Hardware-in-the-Loop Modeling and Simulation Methods for Daylight Systems in Buildings

by

Alex Robert Mead

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering - Civil and Environmental Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Khalid M. Mosalam, Chair
Professor Steven D. Glaser
Assistant Professor Stefano Schiavon
Professor Edward A. Lee

Spring 2017

Hardware-in-the-Loop Modeling and Simulation Methods for Daylight Systems in Buildings

Abstract

Hardware-in-the-Loop Modeling and Simulation Methods for Daylight Systems in Buildings

by

Alex Robert Mead

Doctor of Philosophy in Engineering - Civil and Environmental Engineering

University of California, Berkeley

Professor Khalid M. Mosalam, Chair

This dissertation introduces hardware-in-the-loop modeling and simulation techniques to the daylighting community, with specific application to complex fenestration systems. No such application of this class of techniques, optimally combining mathematical-modeling and physical-modeling experimentation, is known to the author previously in the literature.

Daylighting systems in buildings have a large impact on both the energy usage of a building as well as the occupant experience within a space. As such, a renewed interest has been placed on designing and constructing buildings with an emphasis on daylighting in recent times as part of the "green movement."

Within daylighting systems, a specific subclass of building envelope is receiving much attention: complex fenestration systems (CFSs). CFSs are unique as compared to regular fenestration systems (e.g. glazing) in the regard that they allow for non-specular transmission of daylight into a space. This non-specular nature can be leveraged by designers to "optimize" the times of the day and the days of the year that daylight enters a space. Examples of CFSs include: Venetian blinds, woven fabric shades, and prismatic window coatings. In order to leverage the non-specular transmission properties of CFSs, however, engineering analysis techniques capable of faithfully representing the physics of these systems are needed.

Traditionally, the analysis techniques available to the daylighting community fall broadly into three classes: simplified techniques, mathematical-modeling and simulation, and physical-modeling and experimentation. Simplified techniques use "rules-of-thumb" heuristics to provide insights for simple daylighting systems. Mathematical-modeling and simulation use complex numerical models to provide more detailed insights into system performance. Finally, physical-models can be instrumented and excited using artificial and natural light sources to provide performance insight into a daylighting system. Each class of techniques, broadly speaking however, has advantages and disadvantages with respect to the cost of execution (e.g. money, time, expertise) and the fidelity of the provided insight into the performance of the daylighting system. This varying tradeoff of cost and insight between the techniques determines which techniques are employed for which projects.

Daylighting systems with CFS components, however, when considered for simulation with respect to these traditional technique classes, defy high fidelity analysis. Simplified techniques are clearly not applicable. Mathematical-models must have great complexity in order to capture the non-specular transmission accurately, which greatly limit their applicability. This leaves physical modeling, the most costly, as the preferred method for CFS. While mathematical-modeling and simulation methods do exist, they are in general costly and and still approximations of the underlying CFS behavior. Meaning in fact, measurements of CFSs are currently the only practical method to capture the behavior of CFSs. Traditional measurements of CFSs transmission and reflection properties are conducted using an instrument called a goniophotometer and produce a measurement in the form of a Bidirectional Scatter Distribution Function (BSDF) based on the Klems Basis. This measurement must be executed for each possible state of the CFS, hence only a subset of the possible behaviors can be captured for CFSs with continuously varying configurations. In the current era of rapid prototyping (e.g. 3D printing) and automated control of buildings including daylighting systems, a new analysis technique is needed which can faithfully represent these CFSs which are being designed and constructed at an increasing rate.

Hardware-in-the-loop modeling and simulation is a perfect fit to the current need of analyzing daylighting systems with CFSs. In the proposed hardware-in-the-loop modeling and simulation approach of this dissertation, physical-models of real CFSs are excited using either natural or artificial light. The exiting luminance distribution from these CFSs is measured and used as inputs to a Radiance mathematical-model of the interior of the space, which is proposed to be lit by the CFS containing daylighting system. Hence, the components of the total daylighting and building system which are not mathematically-modeled well, the CFS, are physically excited and measured, while the components which are modeled properly, namely the interior building space, are mathematically-modeled. In order to excite and measure CFSs behavior, a novel parallel goniophotometer, referred to as the CUBE 2.0, is developed in this dissertation. The CUBE 2.0 measures the input illuminance distribution and the output luminance distribution with respect to a CFS under test. Further, the process is fully automated allowing for deployable experiments on proposed building sites, as well as in laboratory based experiments.

In this dissertation, three CFSs, two commercially available and one novel - Twitchell's Textilene 80 Black, Twitchell's Shade View Ebony, and Translucent Concrete Panels (TCP) - are simulated on the CUBE 2.0 system for daylong deployments at one minute time steps. These CFSs are assumed to be placed in the glazing space within the Reference Office Radiance model, for which horizontal illuminance on a work plane of 0.8 m height is calculated for each time step. While Shade View Ebony and TCPs are unmeasured CFSs with respect to BSDF, Textilene 80 Black has been previously measured. As such a validation of the CUBE 2.0 using the goniophotometer measured BSDF is presented, with measurement errors of the horizontal illuminance between +3% and -10%. These error levels are considered to be valid within experimental daylighting investigations. Non-validated results are also presented in full for both Shade View Ebony as well as TCP.

Concluding remarks and future directions for HWiL simulation close the dissertation.

To Nate Shoe

“ . . . a friend who sticks closer than a brother.”

Contents

List of Figures

List of Tables

Acknowledgments

Thank you first and foremost to God, who I know through Jesus Christ, for all I've been given. What I ask, is answered, what I seek, I find, the doors upon which I knock are opened. This thesis work makes sense now.

Thank you to my family. My mother, who loves and supports me beyond anything I deserve. My father, who leads me by an example which still inspires. My brother and sister who are always there, even when I am in the wrong.

With regard to the thesis, thank you to my research advisor, Professor Khalid M. Mosalam, who offered to work with me when I sought opportunities. Thank you as well to my academic advisor, Professor Steven Glaser, who saw me through beginning to end. In addition, I'd like to recognize my remaining dissertation committee, Assistant Professor Stefano Schiavon who taught me of buildings and life, and Professor Edward A. Lee who inspired me to explore electrical engineering and the abstractions this field has to offer. Finally, a special note of thanks to Professor Raja Sengupta who influenced me early in my career and whose unique personally has been deeply impactful and I will never forget.

In closing, thank you to my friends in California who supported me over the last six years. I could not have sustained the load without your support. For those beyond naming, your love is felt. Thank you to Race for your well-reasoned opinions and friendship. Thank you to Jake and Becky, for a door always open, phones always answered.

Funding

This research was funded in part by the Republic of Singapore's National Research Foundation through a grant to the Berkeley Education Alliance for Research in Singapore (BEARS) for the Singapore- Berkeley Building Efficiency and Sustainability in the Tropics (SinBerBEST) program. BEARS has been established by the University of California, Berkeley, as a centre for intellectual excellence in research and education in Singapore.

Chapter 1

Engineering Research and Development

This thesis reports on work in the exciting and highly important field of *building performance simulation* [61]. This field examines how buildings operate, with the ultimate goal of designing and constructing better buildings for both occupants and the planet. This thesis presents research done on daylighting within buildings [128]. Beginning in this chapter, engineering research in general is introduced, with a specific outline of motivation as to why this particular work was conducted. Further, an outline of the thesis, as well as a specific list of its contributions are provided. In chapter 2, a more domain specific introduction to the field of daylighting is given with respect to the terminology introduced here.

1.1 The What: Systems and Experiments

In modern society, engineers hold the privileged position of shaping the physical world in which we live. Whether on the infrastructure scale constructing freeways, dams, and buildings, or on the micro scale building embedded processors, engineers' creations interact with our lives continuously.

The actual objects with which engineers work are referred to as *systems*, defined in this text as "an object or collection of objects whose properties we want to study, the total being a subset of the Universe [1]." Systems in this context can be of natural origin - the Universe, a human heart, the Jet Stream - artificial origin - gas turbines, automobiles, buildings - or a combination - solar irradiance and a photovoltaic panel. While this definition is somewhat arbitrary, clarity is added when considering the *environment* in which the system resides. Here the environment is defined as, "the remaining set when considering the Universe minus some corresponding system under study." System and environment interact through what is called the *interface*, which is, "the inputs from some environment and the outputs of some corresponding system which together define the interaction of the system-environment pair."

[1] The definitions of *system*, *experiment*, *model*, and *simulation* are heavily influenced by Fritzson [49]

An interface in this context is somewhat arbitrary, yet directly reflects the information which is sought by the engineer. Thus, for a given system and environment there may exist several interfaces depending on the desired analysis.

In order to learn the desired information, engineers will perform an *experiment*, "the process of extracting information from a system by exercising its inputs and observing its outputs." In *engineering*, this information typically leads to the construction of a built system for a desired task, while in contrast, *science* seeks information of systems to advance the knowledge of humanity.

As an example of the above definitions, consider an automobile system, including the body, tires, engine, etc. The environment of the automobile is everything we see, minus what is clearly the automobile, including the atmosphere, pavement, buildings, etc. Depending on the analysis which is to be carried out there exist many interfaces, such as (*body panel*, *air*) interface, the (*tire*, *pavement*) interface, or in a collision study, the (*body panel*, *concrete barrier*) interface. An experiment for an aerodynamic study could include various wind tunnel settings (the input) and measurement of the drag on the car panels (the output). There literally exists an infinite amount of interfaces and experiments which could be conducted for a given system-environment pair, however, some are more useful than others and these are the ones engineers focus on when studying systems.

1.2 The How: Models and Simulations

While using experiments to studying systems is effective and provides fundamental information, experimentation on the system is not always possible. Perhaps the engineer is designing a new system and it doesn't exist yet. Maybe it is too expensive or not practical to build the system to study it (e.g. a large dam). Often safety limits can prohibit system construction and study, for example experimental rocket engines or large oil drilling platforms. In these cases, it is often desired to construct a *model* as a representation of the system and study it to gain information.

In this context a *model* is "anything representing a system of interest on which an "experiment" can be applied in order to answer questions about that system." Since an *experiment* refers to a *system*, the term *simulation* or, "an experiment performed on a model," is introduced. The following analogy can be formed: an experiment relates to a system as a simulation relates to a model. Further, a simulation must be defined with respect to the specific model-system pair from which system information would normally be extracted using an experiment.

Conceptually one can think about the relationship between model/simulation and system/experiment by examining Figure 1.1. The X axis represents cost of some model/simulation investigation (e.g. time, money, man-power, computation) and the Y axis represents the realism of the model/simulation investigation in representing a corresponding system/-experiment (e.g. is water's behavior in a flume really representative of the natural system of interest, a stream?). Ideally, a model/simulation resides in the top left, being low in cost, yet high in realism. With the presented definitions of model, many analysis techniques can be considered as “models/simulations”, however, in today’s engineering, models generally fall into two classes: *Physical-Models* and *Mathematical-Models*.

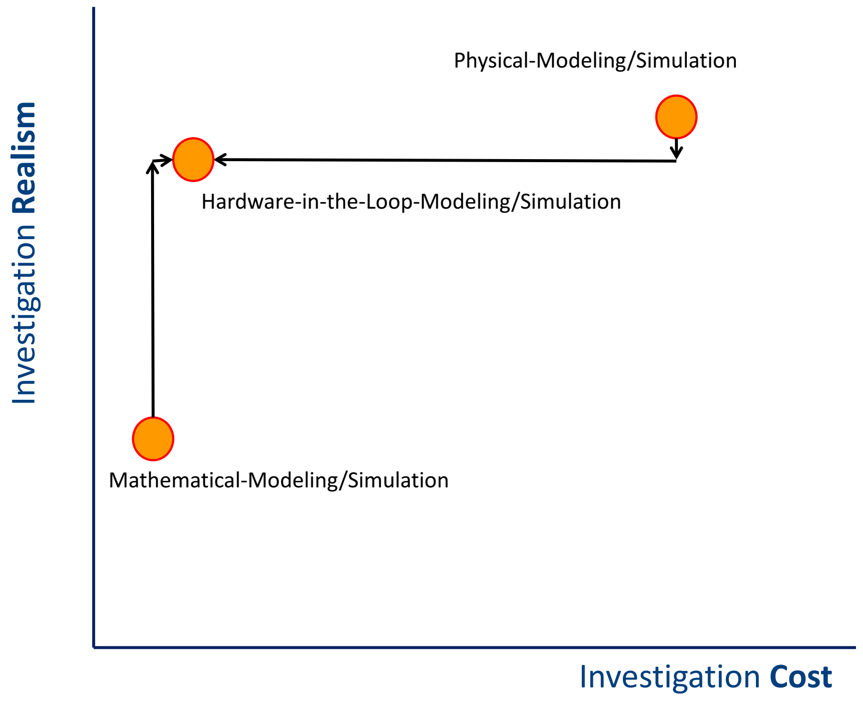

Physical-Modeling/Simulation
Hardware-in-the-Loop-Modeling/Simulation
Mathematical-Modeling/Simulation
Investigation Realism
Investigation Cost

Figure 1.1: Conceptually illustrates the relationship between various modeling/simulation methods (i.e. “investigations”) and the relative realism as compared to the respective system/experiment. Diagram inspired by Selim Gunay.

1.2.1 Physical-Modeling/Simulation

Physical models are those that involve the actual physical construction of some piece of hardware as a representation of a system. While simple geometrically scaled models are ubiquitous (e.g. plastic injection molded planes, trains, and automobiles assembled by hobbyist), the field of physical modeling is hugely complex and varying, including a myriad of application domains from *aerospace* to *civil structural* engineering problems.

Physical-models are governed by the *laws of similitude* [38], which, roughly speaking, mandates the equality between all dimensionless quantities, often called π groups, of both the system and the model as well as any interface components for the given analysis. In reality, fulfilling the similitude requirement is impossible, as there is no physical manner to construct a totally dimensionally consistent model with respect to a system of interest [2]. As such, engineers focus their attention on equating the most pertinent π groups for a given problem. Identifying the significant π groups for various classes of systems is well studied, comprising both rigorous analysis technique and experience gained intuition, what some call "art." Perhaps the most famous π group is the Reynolds number ($Re = \frac{\rho L V}{\mu}$; ρ-fluid density, L-characteristic length, V-fluid velocity, μ-fluid viscosity) from fluid dynamics, with another example from heat transfer in buildings being the Biot number ($Bi = \frac{hL}{k_b}$; h-convective heat transfer coefficient, L-characteristic length, k_b-thermal conductivity).

With the physical model constructed to represent some system, a simulation must be likewise designed and executed to represent the analogous experiment. For this, specialized laboratory facilities around the world have been constructed to exacting tolerances to fulfill similitude requirements of various physical-models/simulations. Examples include the Marine Hydrodynamics Laboratory at the University of Michigan, Ann Arbor [147] for fluid dynamics of ships, the Richmond Field Station at the University of California, Berkeley for civil structural testing [131], and the Transonic Cryogenic Tunnel at NASA's Langley Research Center in Virginia for aerospace system testing [148].

While physical-modeling/simulation is very broad, in general the cost of investigation is relatively high (e.g. time, money, expertise, experimental facility availability); the results also have proportionally high realism when compared to the representative system/experiment. This is illustrated conceptually in Figure 1.1, where physical-modeling/simulation is placed in the upper right corner of the cost versus realism plot.

1.2.2 Mathematical-Modeling/Simulation

Mathematical-modeling/simulation is an established field in engineering for representing systems and experimentation. *Mathematical-models* come in many forms, and all can be summarized as a mathematical description of the relationship between variables making up a system [32]. Mathematical-models vary from simple, as in single degree-of-freedom (DOF) linear equations, to complex, as in highly coupled, differential-algebraic systems comprising millions of DOFs. They may include stochastic relationships, be derived through empirical measurement and fitting, or be arrived at through complex first principle analysis of natural laws. The multitude of mathematical models currently employed in the engineering disciplines is truly staggering.

With respect to simulation for mathematical-models, or as it is commonly referred, *solving the model*, two possibilities result. Firstly, there may be an *analytical solution* to the mathematical-model which can be calculated. This solution is then used to learn information about the corresponding system/experiment. Analytical solutions, however, are very rare for

[2]One exception is the construction of *full scale* physical models in which a complete replica of the system is built for simulation purposes, referred to here as a *prototype*. In this text, however, full scale physical models are classified as actually building the system and thus are considered in the system/experiment classification, not model/simulation

real world engineering applications. Usually, for engineering mathematical-models *numerical solution* techniques are needed which necessitate the usage of computers for solution [33].

With the now near universal availability of digital computers, mathematical-modeling/ simulation has seen rapid growth in its application to the modeling/simulation problem. With a mathematical-model and its associated simulations existing entirely in the digital realms of computers, vast sets of systems and experimental conditions can be explored quite rapidly and at a relatively low cost. While software and the associated computing resources aren't inexpensive, they pale in comparison to the costs of physical testing, hence in general are far to the left in Figure 1.1 with respect to physical-modeling. Yet, many assumptions with respect to their representative systems are made in the generation of mathematical-models (*structural model errors*, *parametric errors* [33]). Still further, the numerical solutions of these models are only approximations of the true solutions (*truncation errors*, *roundoff errors*, *accumulation errors* [33]). Hence, there is a larger shortcoming with respect to realism of the investigation of a mathematical-model/simulation than exists with a physical-model/simulation. It should be noted, some mathematical-models are both low cost and exhibit similar realism to physical-models, however, the above discussion and Figure 1.1 are referring to those systems, or parts of a system, which *are not* well modeled by mathematical-models. Hence, the lower cost of a mathematical-model does in fact result in a distinct lower realism.

1.2.3 Hardware-in-the-Loop-Modeling/Simulation

Given the complexity of systems encountered by engineers today, it is common for a system to be composed of several interacting subcomponents, each of which is also a system. Given the engineer is concerned about the complete system, or super-system, a model/simulation capable of representing its entire behavior with high realism is desired. Knowing each of the subcomponents, or sub-systems, are themselves *systems*, one can imagine the existence of super-systems where some of its sub-systems can be represented by mathematical-models/simulations with high realism, yet others are more optimally represented by physical-models/simulations. When super-systems of this nature are encountered, engineers can in fact use both techniques in a single modeling/simulation framework combining both physical and mathematical models. Modeling/simulation frameworks of this class are referred to as *hardware-in-the-loop-modeling/simulation* (HWiL), more formally defined as, "a representation of a system/experiment with certain subcomponents represented using mathematical-modeling while others are represented using physical-modeling, where all model components are interacted throughout the simulation such that the total model/simulation is representative of the system/experiment under consideration."

Examining Figure 1.1, one can see a disproportional increase in realism with respect to the increase in cost of a HWiL as compared to a fully mathematical-model/simulation. Similarly, there is a disproportional decrease in realism of the the HWiL with respect to the decrease in cost as compared to a fully physical-model/simulation. Hence, HWiL is a synergetic analysis method with respect to physical/mathematical-modeling/simulation for certain complex systems.

While the vision presented above is correct with respect to HWiL modeling/simulation, it is purely conceptual and skirts over many details. For a more detailed treatment of HWiL modeling and simulation, with attention paid to the partitioning of physical and mathematical models and the timing of the simulation, see section 3.1.

1.3 The Where: Building Daylighting Systems

Humanity is constantly striving to improve the state of its existence. Given that engineers are tasked with creating or controlling large portions of the physical world in which humanity exists, much of an engineer's job is working towards this goal. While the priorities of an engineer are vast, changing emphasis depending on the system being considered and the society in which that consideration is taking place, few would disagree that *improving human environmental quality* and *reducing energy use* are main priorities for today's engineers. As basic motivation, one can definitionally consider a better environment quality more favorable than a poorer environment quality. Further, regarding energy use reduction, one can span the gamut from climate change mitigation, to resource conservation, to national defense, to job creation and economic development (perhaps depending on one's priorities or political persuasion) for compelling motivation. With this in mind, many research and development engineers are focusing their efforts on systems in domains with large impacts to both energy use reduction and the increase of human environmental quality.

1.3.1 Buildings as Systems

Considering buildings as systems, one can quickly gather this is a prime area of research and development for both energy use reduction and increases in environmental quality. Examining the important work of energy categorization done by Lawrence Livermore National Laboratory [86], one sees residential and commercial buildings used 20.87 and 18.01 quadrillion BTUs of energy respectively in 2015 (the latest data available). This equates to 39.8% of all primary energy used in the United States. Further, many human activities are executed in these residential and commercial buildings, and thus buildings define the environment which engineers seek to improve. Hence, if engineers can improve the environmental quality of buildings, while reducing the energy they consume, they will be fulfilling two of their major priorities in conducting research.

1.3.2 Building Envelopes

A *building* can be succinctly defined as, "a subspace of the three dimensional world, in which the environment is controlled to be more conducive for some desired task, what architects call the 'building program."' In the definition there exists an implicit delineation of the “inside” and “outside” of the building, physically realized by the *building envelope*. Further, many of the aspects of controlling the environment of the “inside” require energy and create potentials across the building envelope which must be maintained (e.g. thermal differential, relative humidity differential, air pollutant differential). Examining Figure 1.2, it can be seen that ventilation, space heating & cooling, and lighting (i.e. the primary systems used for environmental control of the building “inside” and thus maintain the potentials) have respective totals of 53% and 60% for residential and commercial building energy use [37]. This means that creating and maintaining an “inside” environment conducive to the building program requires signifiant energy resources, with the magnitude of this resource use heavily influenced by the building envelope. As such, better building envelopes will allow for an increase in the quality of the “inside” environment and corresponding reductions in energy use to maintain that quality.

1.3.3 Fenestration Systems

Examining ventilation, space heating & cooling, and lighting, the primary users of energy in buildings, one notices the *Sun* strongly impacts how much energy each of these systems use. A primary mechanism for which this takes place is solar loads on the building itself (e.g. through glazing, warming opaque envelope elements) which form significant heat loads the building HVAC systems must accomodate. In addition, it is well established that exposure to daylight (i.e. sunlight with wavelength of 380-780 nm to which the human eye is sensitive

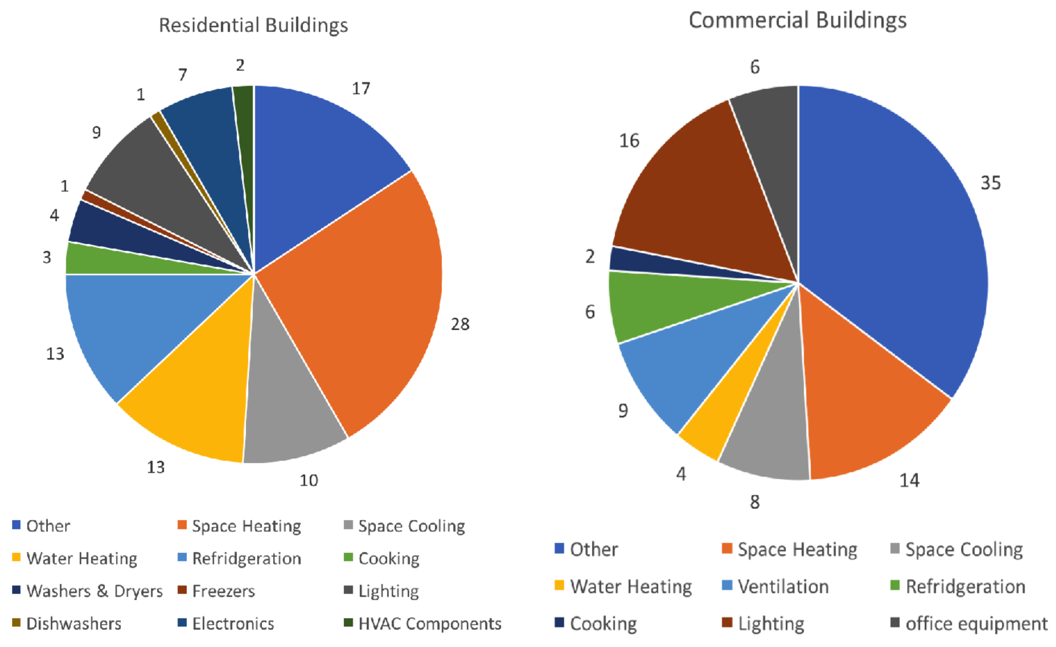


Figure 1.2: Breakdown of energy use by domain in residential buildings [37].

[134]) has health benefits and contributes to productivity gains for building occupants [43, 15]. With this in mind, fenestration systems, specifically the glazing (i.e. windows), occupy a unique position within modern building envelopes and thus building systems in general. This unique position means improvements in fenestration systems have sweeping affects into both the environmental quality present within buildings as well as the magnitude of energy buildings use to maintain their “inside” environments. By focusing attention on improving building fenestration systems, engineers are conducting high impact research both improving environmental quality and reducing energy use.

Given the fact that fenestration systems are a uniquely high impact area for energy use reduction and environmental quality improvement, their study is currently undergoing a “renaissance” as part of the larger “green movement.” Specifically, a class of fenestration systems called *complex fenestration systems* (CFS) are receiving significant attention in building system research communities and are seeing increased use in the industry for real buildings. CFSs distinguish themselves from traditional glazing in that they attempt to optimize the amount of daylight entering a space at any given moment of the year by exhibiting non-specular reflection and transmission properties. For example, allowing the maximum light into the space in winter without causing glare and the minimum light in the summer to provide illumination yet not overheat the space. Examples range from simple window shades which can be deployed by hand [150], to highly complex systems involving automated sensing and actuation of the CFS [143].

1.4 Summary

A brief summary of the entire dissertation is now provided. It begins in chapter 1 with an explanation of engineering research and development. Meaningful terminology (e.g. system, model, and simulation) is initially introduced and defined. In this explanation, what research engineers work on, as well as how they work, is discussed. The topic of this thesis, modeling and simulating building daylighting systems, is then introduced using this framework of reasoning as a justification. To finish, an outline of the whole thesis, as well as the explicit contributions of this thesis are stated.

Next, in chapter 2, a background into daylighting and buildings is provided. This involves giving a formal definition to daylighting, as well as an overview of how building daylighting systems are designed and analyzed. The three major classes of analysis, *simplified techniques*, *mathematical models*, and *physical experimentation* are each explored. Concluding, complex fenestration systems (CFS) are discussed, with emphasis regarding a gap in the current analysis approaches. The possibility of a fourth class of analysis methods, *hardware-in-the-loop* (HWiL) modeling and simulation, is then presented as a possible solution to this gap.

Chapter 3 then introduces HWiL modeling and simulation. This introduction begins with an abstract presentation concerning the method. Next, classification of HWiL models and their simulations are given, first with respect to sensor and actuator dynamics compensation, then to timing limitations. The second part of the HWiL introduction outlines the proposed HWiL model designed to fill the current gap regarding CFS analysis in building daylighting systems. An initially proposed architecture for the HWiL model is presented first, then a second method inspired by the three-phase method is described. The second method is what eventually forms the HWiL model and simulations presented here.

With the idea expressed for the HWiL model, chapter 4 presents the design and construction of that system. Beginning by stating the needs of the system, then moving onto an initial design, the CUBE 2.0 system, the name of the HWiL model presented in this thesis, begins to take form. Various models and simulations constructed concerning the design of the CUBE 2.0 itself are then presented, each ultimately influencing the final design, which is also presented. Next, chapter 4 shows in detail the construction of the CUBE 2.0 system. First, the physical components are realized, followed by the cyber components. Details of this construction can be seen with extensive photographs in the appendices as well as large sections of the code.

With the system built, chapter 5 then turns attention to the detailed steps necessary for calibration. Calibration is necessary to ensure the numbers being produced during the simulations are in fact meaningful SI units. Details of the calibration technique as well as the runs needed to execute this process are presented. With these techniques, the CUBE 2.0 was calibrated and is now ready for use in real simulations.

In chapter 6, three CFSs are tested using the CUBE 2.0. Two commercial products (Textilene 80 Black, Shade View Ebony) and one novel (translucent concrete panels) developed as part of tangential research to the CUBE 2.0. Textilene 80 Black had been measured previously, hence allows for validation of the CUBE 2.0, a task which is executed and presented. In total, results for all three CFSs are presented for daylong simulations at one minute time steps.

Finally, chapter 7 draws conclusions and provided direction for future research ideas involving HWiL and more specifically the CUBE 2.0 system.

1.5 Contributions of this Thesis

As a summary, the key contributions of the CUBE 2.0 system are presented. The CUBE 2.0 is the first known hardware-in-the-loop (HWiL) model within the daylighting community. By partitioning the physics of building daylighting systems between those components modelled faithfully by mathematics and those components still requiring physical experimentation, an optimal model is constructed. The building daylighting community is now introduced to a new modeling technique and the abstract concept of HWiL modeling and simulation has been advanced as an analysis method.

By integrating physical specimens directly into the analysis, very little extra effort must be expanded beyond running the actual simulation to analyze new specimens. Thus, if a new CFS is conceived (perhaps using 3D printing or other rapid manufacturing technique) and analysis is desired, one only needs to attach the specimen to the CUBE 2.0 and execute a simulation. No tedious measurement of material properties or modeling assumptions must be made. This is quite the contrary to the standard daylighting analysis techniques currently in practice, involving extensive material property measurements, modeling assumptions, and often both, for analysis of CFSs.

As a demonstration of this capability, daylong experiments were performed at one minute time steps for three CFSs: Textilene 80 Black, Shade View Ebony, and Translucent Concrete Panels (TCPs). Specifically, Textilene 80 Black has been measured previously for a bidirectional transmission distribution function (BTDFs), allowing a validation of the CUBE 2.0. This validation shows the CUBE 2.0 system is capable of performing HWiL simulations with errors of less than ± 10 %. This is well within the daylighting community standards established in the literature. In addition, meaningful simulation data over the course of a full day was provided for Shade View Ebony and TCPs, results which would be quite laborious to obtain using traditional modeling and simulation techniques.

Further, the CUBE 2.0 offers for the first time a method of testing continuously augmentable CFSs. That is, CFSs which can exist in a very large number of states. For example, consider an electrochromic window type CFS or a dynamic CFS with continuously movable components. In order to characterize these before, bidirectional scatter distribution functions (BSDFs) would need to be measured for each possible state of the CFS. While perhaps practical for a small number of states (i.e. <5), this quickly fails for a real system. By using the actual CFS on the CUBE 2.0 no measurement and modeling is necessary as the real system is being tested.

In addition, the CUBE 2.0 is portable, hence site deployments on current or future building project sites can be executed. By taking the CUBE 2.0 to actual locations, real world conditions can be investigated. Further, due to it's portability, the CUBE 2.0 can also be brought to experimental laboratories where sun simulators can be used to excite the specimens under test. This allows for rapid characterization of the specimens as compared to site investigations and can even be used to explore hypothetical design proposals for locations around the world.

Finally, the CUBE 2.0 system is believed to also be able to measure bidirectional transmission distribution functions (BTDFs) for the front side of the CFS under test. Using the input measurement from the Canon 6D and fisheye lens and the output measurement from the optical fibers and RPiCM, system ID techniques are believed to be useable to identify this property. As of the writing of this thesis, however, the data has not been fully processed, yet promising initial results are presented in Appendix C.

Chapter 2

Daylight Analysis Techniques

In chapter 1 the study of complex fenestration systems (CFSs) by research and development engineers was motivated from "first principles" of engineering. In this chapter, *daylight* and its position in the building systems community will be explained. In context of the classic design-evaluate-iterate cycle, the traditional modeling/simulation techniques associated with daylighting systems will be presented. Shortcomings of these techniques will be pointed out with respect to CFSs, specifically within a rapid prototyping environment, offering motivation for a hardware-in-the-loop modeling/simulation (HWiL) approach. An abstract overview of HWiL, as well as a concrete presentation of the proposed HWiL architecture for building daylighting systems will be presented in chapter 3.

2.1 Daylight and Buildings

Sunlight originates at the Sun and travels through the vacuum of space to reach the top of Earth's atmosphere and eventually its surface. The Sun emits electromagnetic ration (EMR) well approximated by Black Body Radiator Theory and thus provides a continuous, broad banded spectrum of radiant energy. The relative magnitude of this energy per wavelength can be calculated using Plank's Law. The Sun's apparent surface temperature is approximately 5500^o Kalvin. The total energy striking the atmosphere is called the *solar constant* and is about 1367 $[W/m^2]$ [128]. The reflection back to space and the filtering of sunlight as it travels through the Earth's atmosphere further affects the spectral content. These processes depend on many factors within the atmosphere and result in an average of approximately 1000 $[W/m^2]$ of the original 1367 $[W/m^2]$ actually striking the Earth's surface [151]. Traditionally, sunlight is divided into three wavelength defined components, *ultraviolet (UV)* $(10 < \lambda < 380nm)$, *visible* $(380nm < \lambda < 780nm)$, and *near-infrared (NIR)* $(780nm < \lambda < 10^6nm)$ [87]. While humans can perceive the ultraviolet and infrared, for example via sunburns or the warmth of sunlight on the skin, the human eye only perceives the visible portion. The visible subset of sunlight is what is referred to herein as *daylight*.

EMR has been extensively studied for its psychophysical properties, resulting in the luminosity efficacy curve, $V(\lambda)$, used extensively today [134]. In fact, this curve is what defines the UV, visible, and NIR components of sunlight. Given the nature of daylight's impact on the human vision system, its impact on buildings has historically been analyzed independant of the remaining parts of the spectrum, a treatment continued in this thesis. However, as explained above and in chapter 1, daylight is only part of sunlight and its total impact on buildings. As such, a growing number of investigations concerned with the interaction of daylight and sunlight in general have been executed [29, 112, 56, 83, 25].

It should be noted, full sunlight analysis with respect to the techniques presented here is postponed for future work, meaning this thesis will focus exclusively on daylighting physics. A note on terminology, *daylight* refers to the subset of sunlight humans can see, where as, *daylighting* refers to the practice of using daylight to provide illumination within a building. Further, *light* refers to all sources of EMR with wavelength in the visible range, hence, includes natural light from the sun and artificial light from electric luminaries.

2.2 Modeling/Simulation for Daylighting

2.2.1 The Classic Design Process

First, the designer proposes an initial design which is evaluated in some manner to determine it it meets the design requirements. Based on the results of the initial design evaluation, the designer either, accepts the design and the design process ends, or the initial design is updated and re-evaluated iteratively until it is accepted. While the term "evaluation" is used, this is actually the same process of modeling/simulation presented in chapter 1. In the daylighting industry modeling/simulation techniques can be broadly grouped into three classes: physical-models, mathematical-models, and the previously undiscussed *simplified techniques*.

2.2.2 Simplified Techniques

Simplified Techniques make various assumptions about the daylighting analysis both on the model and simulation side. These assumptions reduce accuracy, but make the analysis much more tractable. For example, a simplified method may assume uniform sky conditions (very rarely, if ever, realized in nature) or a fixed position for illumination analysis within a space. Based on so called, "rules-of-thumb" heuristics, these techniques don't rely on first principles of daylight, but rather on experience gathered within the industry to predict the adequacy of a design. A limited number of examples are presented in the sections below.

Window-Head-Height-Rule

A classic example of a simplified daylighting analysis technique is the Window-Head-Height-Rule presented by Reinhart [124]. This rule simply states the depth of daylight penetrating a side lit office space with venetian blinds is approximately one to two times the

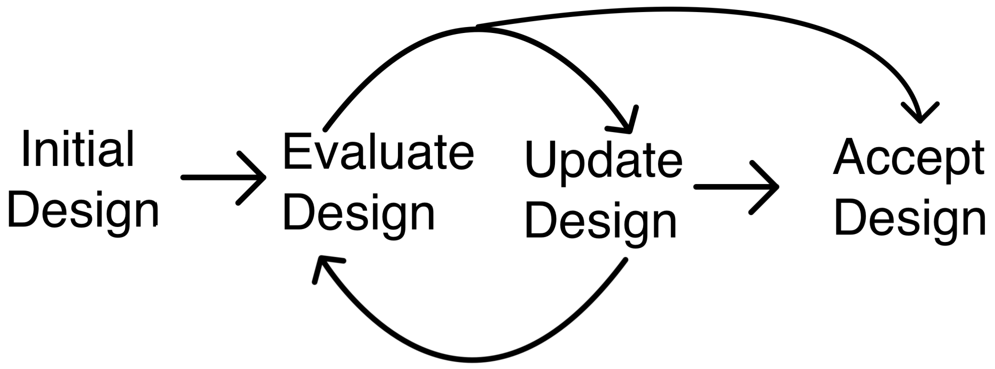


Figure 2.1: Classic Design Cycle: A design is proposed, it is evaluated, the design is updated based on evaluation, then re-evaluated and re-updated until the design is deemed adequate and finally accepted.

window head height, see Figure 2.2. Meaning, activities located within this region of the floor plan can effectively be regarded as occurring in a "day lit" space.

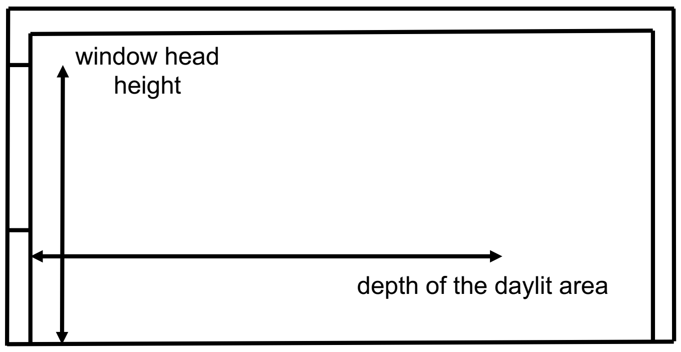


Figure 2.2: Side lit office demonstrating the simplified daylight analysis technique, Window-Head-Height-Rule.

View of the Outdoor Percentage

This metric is defined as the percentage of an occupant's view which is made up of glazing versus other building constructions (e.g. opaque envelope, partition walls) and contents (e.g. desks, people, equipment). That is, the number of steradians which can be defined as glazing divided by the total steradians of the human visual system, approximately 2π. This gives an indication of the magnitude of impact daylight will have on an occupant with respect to views of the outside [45], a meaningful metric for today's buildings. Further, it helps quantify the increasingly important consideration of circadian rhythm preservation which is dependent on exposure to sunlight [95, 129]. This metric has an optimal location, with no view and total view each being negative for occupants. The "best" value is often context dependent.

Advantages and Disadvantages

Advantages of Simplified Techniques include simple and easy to apply calculations or heuristic evaluations which give a base insight into a proposed design's behavior. Disadvantages include the major assumptions involved in these loose calculations and the associated differences from reality in the results. Referring to Figure 1.1, clearly Simplified Techniques are in the lower left section of the plot, offering a very low cost investigation technique and a corresponding low realism return with respect to the actual system.

2.2.3 Mathematical-Modeling/Simulation

Mathematical-models representing light are studied by the branch of physics known as *optics*. Optics has been the pursuit of some of the most famous scientific inquires of all time. Names including Newton, Maxwell, Plank, and Einstein are among the scientists who have explored this space and derived beautifully elegant models describing light's behavior in a myriad of different scenarios. Depending on the scope of explanation, mathematical-models for light can be broken down into three, successively more approximate classes: *quantum optics*, *physical optics*, and *geometric optics*.

Quantum optics uses mathematical principles including *waves* and *particles* from quantum mechanics to faithfully describe complicated light behaviors such as the photoelectric effect. It is a very faithful representation of light, however, comes at a high cost in terms of complexity and abstraction. This complexity is severely limiting for engineering applications, thus quantum optics mathematical-models are rarely used in engineering applications. Physical optics, or *wave optics*, involves approximations in its mathematical models when compared to quantum optics. Still it captures interesting phenomena such as interference and diffraction by modeling light as waves. While an approximation itself, physical optics is still quite complicated in its application to real systems of engineering interest, thus too sees limited applications. Finally, geometric optics, or *ray optics*, makes the most approximations of all the mathematical models associated with light. Using *rays*, vectors representing EMR movement, and bidirectional reflection (mapping between incoming ray direction and outgoing ray reflection) and transmission properties (mapping between incoming ray direction and outgoing ray transmission) at a material surface, it captures the vast majority of observable phenomena in building lighting applications. This usage of far simpler mathematical models, including the *Law of Reflection* and *Law of Refraction*, still, however, create non-trivial computational loads for building lighting applications. Yet these costs are at a reasonable level for engineering analysis and hence see usage.

Geometric Optics for Daylighting

While some simplified building daylighting scenarios (e.g. overcast sky illumination of a horizontal plane) can be evaluated using geometric optics via analytical solutions, these are exceedingly rare. The vast majority of daylighting designs must be modeled and simulated using numerical techniques as employed by digital computers. The actual implementation of these techniques in analysis tools is complex and has seen a long history of development, well characterized by Dutre *et al.* [42] and recently reviewed by Ochoa *et al.* [113].

Currently, the computer program Radiance is by far the industry leading implementation of a geometric optics tool for daylighting analysis [157]. Radiance uses a simple workflow illustrated in Figure 2.3. First, a model of a proposed lighting design is constructed comprising the space geometry, material properties, and light sources (i.e. daylighting and electric luminaires). Next, a specification and execution of the type of desired analysis is completed. For example, one may desire to calculate horizontal illuminance at the workplane height of 80 cm at a vertically oriented grid of points, for a given sky condition. Finally, the results are interpreted to determine if the design is adequate for the purposes of the space. Note, the Illuminating Engineering Society of North America's (IESNA) Lighting Handbook [41] has numerous metrics to evaluate lighting, many of which Radiance can calculate, as well as specifications of acceptable values regarding those metrics for various building types.

Radiance uses the *Central Radiance Equation* (CRE) to model light's behavior and a large collection of over 25 material properties models to capture light's interaction with surfaces. The CRE is solved recursively using a combined deterministic-stochastic algorithm with adjustable parameters based on the particular needs of an analysis [84]. Radiance is an open source modeling/simulation framework originally developed at Lawrence Berkeley National Laboratory (LBNL) by Greg Ward in the late 1980's [157]. While it sees continuous development, the core of the framework is relatively static. The book, "Rendering with Radiance" [84], therefore is still a relavent reference for both learning how to use Radiance as well learning the details of the algorithmic solution it implements for the CRE. There also exist other ray based modeling/simulation tools available. Given the dynamic nature of this field, however, there has been some turnover with respect to which programs have seen continued support and development. As such, the United States chapter of the International Building Performance Simulation Association (IBPSA-USA) keeps a rolling registry of tools which can be explored [26]. It was formally maintained by the United States Department of Energy (US DOE) and as of this writing contained several hundred tools for building performance simulation of many different systems, with lighting a key contingent.

While Radiance is the industry leader, and used exclusively in this thesis, for completeness there also exist two other methods which see meaningful use in the daylighting community. The first of which is called the *radiosity method*, which originated as a heat transfer modeling technique and was adopted for building lighting analysis [160]. The second, is called the *photon map method* and can be applied to special situations such as the calculation of caustics and occurrence in concentrated lighting scenarios [73, 74].

Due to the ubiquitous nature of digital computers in society and engineering offices today, mathematical-modeling/simulation has seen a large increase in use in recent years [127]. Given the trend and ever increasing capabilities of these tools, it seems mathematical-modeling and simulation for daylighting are here to stay and will see ever increasing usage. For a well balanced review of the mathematical models used in daylighting, as well as many other aspects of the field, see Ochoa *et al.* [113].

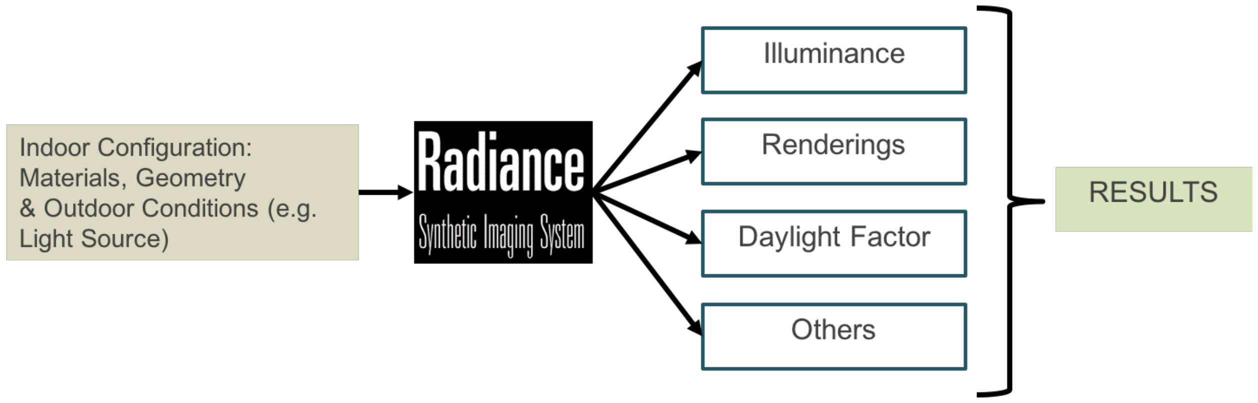

Indoor Configuration: Materials, Geometry & Outdoor Conditions (e.g. Light Source)
Radiance
Synthetic Imaging System
Illuminance
Renderings
Daylight Factor
Others
RESULTS

Figure 2.3: Radiance work flow: construct model including space geometry, material properties, and light sources; specify and execute analysis; evaluate results.

Material Properties in Mathematical Models

While mathematical models with the most realism in representing a system's behavior are critical for engineering analysis (e.g. ray models), without accurate paramaters in those models the analysis is useless. As such, measuring and codifying realistic material properties for mathematical models is a large segment of daylighting research.

In the Radiance program, material properties define how light rays interact with various objects within a scene. This varies significantly from material type to material type, hence Radiance has over 25 built-in types (i.e. mathematical models with different reflection, absorption, transmission, and emission properties) with adjustable parameters to dial in the behavior to the materials which actually make up the system being analyzed. Arguably the most common material type within Radiance modeling is that of *plastic*, which is summarized below, along with two sources of *light* for Radiance models: *light* and *IES file*.

Plastic: Many of the materials in buildings act like plastics, in that they have uncolored highlights when illuminated. The specification requires the *red, green,* and *blue reflectance* (R, G, B), as well as the surface *roughness*, R_s and *fraction of specularity*, S_s. Roughness is defined as the root-mean-square of the surface facets, with a value of 0 indicating a perfectly smooth plane. Fraction of specularity is the amount of reflection from the surface occurring in a specular form versus a diffuse form. This leads to the following material property definitions: $Ref_{diffuse} = (R * 0.265 + G * 0.670 + B * 0.065) * (1 - S_s)$, $Ref_{specular} = S_s$, and $Ref_{total} = Ref_{diffuse} + Ref_{specular}$. Note, realistic values include $R_s < 0.2$ and $S_s < 0.1$ [155].

Further, many methods exist for measuring these properties, some of which are detailed in chapter 5 of Rendering with Radiance [84].

light: A handful of the 25 plus material properties in Radiance actually generate luminous excitation instead of just reflecting, absorbing, and transmitting it. One example is the light material which is specified with simply the *red, green,* and *blue radiance value* $[\frac{W}{sr \times m^2}]$. This material can be used to model objects such as lamps within luminaires, adding *rays* into the model which are propagated throughout the space during simulation.

IES file: While luminous energy producing materials such as *light* listed above can be used for luminaire lamps, many assumptions must be made regarding the angular-varying luminous distribution leaving the material. As such, the distributions often used are very different from what an actual lamp within a luminaire would produce. To offer a more realistic luminous distribution, Radiance can import measured data from lamps and luminaires in the form of IES files, governed by the standard ANSI/IESNA LM-63-02 [12]. These files are generated by measuring each angle of exiting light from a lamp or luminaire using a device called a goniophotometer. Goniophotometers come in three types, A, B, and C, each of which is governed by the standard IESNA LM-75-01 (R2012) [94].

Glazing Material Properties: A Special Case for CFSs

Daylight entering a space must somehow be generated and propagated through a model from the Sun, to where it ultimately lands within the building. Daylight is typically generated by a sky object and a solar disk object of Radiance material type *light*, each of which have special properties and can be automatically generated by a program within Radiance called *gensky*. The daylight must then enter the building, which almost always happens through glazing (i.e. windows) via specular transmission. *Specular transmission* simply means, the rays of daylight entering a glazing pane exit the glazing pane in roughly the same direction they entered, without spreading of the beam of light. Think of a laser shining through a glazing pane where one can see the "dot" on the inside of the building. The amount of incident daylight that enters the space through specular transmission is governed by the *transmissivity* which is calculated via the typically manufacturer measured *transmittance*. The difference in terms has to do with internal pane reflections and how much total daylight leaves the interior side of the pane with respect to that which entered. See the Radiance User Manual for a more complete explanation [155].

While specular transmission of daylight into a building accounts for the vast majority of daylighting, a growing number of fenestration systems are taking advantage of non-specular transmission properties to offer more efficient daylighting than conventional glazing systems. These fenestration systems which rely on non-specular transmission are referred to as *complex fenestration systems* (CFS). Examples include prismatic windows coatings, fritted glass, and movable shading units. With non-specular transmission now needing to be accounted for in mathematical modeling, the measurement of directional behaviour of the incoming rays for transmission and reflection must be quantified.

In order to quantify non-specular transmission, Bartell *et al.* [17] extend the directional reflection distribution function originally put forth by Nicodemus [109] to include all four parts of daylight's interaction with CFSs. This is the outward facing CFS side (i.e. the property of the CFS side facing the outside of the building) reflection and transmission and the inward facing CFS side (i.e. the property of the CFS side facing the inside of the building) reflection and transmission. Collectively these are referred to as the "Transmission Front", "Reflection Front" and "Transmission Back", "Reflection Back" respectively, and abbreviated as BT/RDF Front and BT/RDF Back or also BSDF Front and BSDF Back, where "S" stands for "Scatter."

While many forms of the BSDF exist, the most commonly used today, often called the "WINDOW form" after the software [19], is based on the Klems Basis. BSDFs of this variety take the form of a 145 by 145 matrix, where each column corresponds to an incoming direction of *illuminance*, and the respective rows of that column corespond to the relative fraction of daylight which exits in the corresponding 145 directions in the form of *luminance*. The 145 solid angle partition of the incoming and outgoing hemispheres of a CFS were originally put forth by Klems and Warner [79]. This division was chosen to divide the two half-spaces of which the CFS defines into roughly equal solid angle partitions with the end goal being a new method for calculating solar gains within a building. Their novel method allows for the analysis of multi-layer CFSs via the measurement and combination of individual fenestration system components. The mathematics of the method are first laid out briefly in [80] and then expanded in [81]. Finally, the results of the whole project are presented by Klems *et al.* [82].

Measuring a BSDF is no simple task, as for each of the 145 incident solid angles, a corresponding 145 complementary solid angles must be measured. Meaning, $145 \times 145 = 21025$ individual measurements must be made for each Front and Back, Transmission and Reflection, totaling, $4 \times 21025 = 84100$ measurements[1]. This process is usually executed using goniophotometers of the "scanner" variety. The measurement involves moving a rotating scanner head with a sensor to each of the complementary angles after a light source has been shown on the sample for the corresponding incident angle. Several scanner goniophotometers exist throughout the world including the Cardiff Goniophotometer [22] (technically a goniospectrometer due to wavelength sensitivity of measurements) and the Lawrence Berkeley National Laboratory Goniophotometer [105]. For a large, yet slightly dated, review of goniophotometer, see the second chapter of Andersen's thesis [6]. All scanner goniophotometers, however, suffer from the length of time to conduct a measurement, with typical measurements taking several hours to days to collect a BSDF Front and Back. In response to this long collection time and the need for BSDF characterization of materials, so called "video-based" goniophotometers have seen development.

"Video-based" goniophotometers are based on charged-coupled devices (CCDs) (i.e. the

[1]Often goniophotometers measure based on degrees of *altitude* (θ) and *azimuth* (ϕ), meaning corresponding more or less measurements will actually be executed, however, this is precisely the order of magnitude of measurements regardless of the exact coordinates used.

sensing chips in digital cameras) and projection screens to rapidly capture the complementary 145 solid angle with respect to some incident solid angle. Arguably the most developed and well known “video-based” goniophotometer is the École Polytechnique Fédérale de Lausanne (EPFL) Goniophotometer in Switzerland, originally described by Scartezzini *et al.* [133] and later the primary focus of the Andersen thesis [6]. Fully described by Andersen *et al.* [10], it has also seen validation by Andersen using complementary measurement techniques [7]. Further, validated results from corresponding mathematical models presented by Andersen *et al.* [8] for a prismatic glazing unit and for a Venetian blind [9]. The EPFL Goniophotometer is therefore an established tool for CFS investigation.

The EPFL Goniophotometer, as well as the vast majority of such instruments in the world, measures the response of CFSs with respect to the entire electromagnetic radiation spectrum weighted by the luminous efficacy function, $V(\lambda)$. While visible light as quantified by this method is of primary importance for lighting analysis within buildings, wavelengths corresponding to other portions of the solar spectrum (e.g. UV, near-infrared) also influence building performance. Hence a spectrally quantified BSDF Front and Back of a CFS being considered for use is needed for a realistic analysis beyond simple illumination. With this in mind, Stokes *et al.* [141] created a spectral-goniophotometer at the Massachusetts Institute of Technology (MIT) capable of measuring spectrally resolved BSDFs. Nine seperate spectral bands are independently detectable by the machine [52], and further, a new innovative projection “screen” is realized by a hemi-spheroid allowing for the entire corresponding reflectance or transmittance distribution to be captured in one high-dynamic-range-image (HDRI) using a fisheye lens. The machine is very similar to the previously discussed EPFL Goniophotometer, yet has further capabilities which have been demonstrated by Andersen *et al.* [11].

The MIT Goniophotometer described above uses a hemi-spheroid projection screen and is thus slightly different in the manner in which it captures the corresponding light behavior, either transmission or reflection, compared to the EPFL Goniophotometer. This hemi-spheroid usage was inspired by Ward [156] and is known as a *parallel goniophotometer* [76]. Parallel goniophotometers hold much promise, and come in a few different types, however, also have limitations in terms of size and accuracy well characterized by Karamata *et al.* [77, 78]. Parallel Goniophotometers are a large inspiration of the *CUBE 2.0* system proposed and developed in this thesis, and are hence mentioned here as a primer.

Advantages and Disadvantages

There are many advantages of mathematical-modeling/simulation and they continue to see more usage as time goes on. Specifically Radiance, has seen many validation efforts [98, 63], adding significantly to the confidence in the results it produces. Further, Radiance [157] is free and opensource, with several other packages competitively priced for the building lighting/daylighting community. Within the Radiance opensource community there are also several listserv and online-forums which make learning and debugging the tools easier.

There is even a Radiance specific conference which meets annually concerning the latest developments of the software [161].

Along with these advantages, however, shortcomings also exist. While validated, there are still errors associated with these tools, with 20% error between predicted and reality being a reasonable goal for real world applications [125]. Less than 5% error is exceedingly difficult to accomplish [113]. Further, there is a limit to the complexity of shapes and scenes which these tools can handle, often as a result of the limited number of "bounces" which can be used in the numerical simulation of these models. Also, if a novel building system for daylighting isn't implemented in these tools, they either can't be analyzed at all, or significant modeling assumptions must be made to represent the component in the tool. Material property measurements and database management of these properties is also an issue, with assumptions regarding material behavior commonplace in practice. Further, measuring the material properties themselves is still an open question for many materials [35] and at the very least is an expensive proposition. Still further, in the world of 3D printing, numerous CFSs are being developed rapidly which can't be tested quickly enough with a goniophotometer [44]. Moreover, many CFS have augmentable components which vary via continuously based on measured data coming from sensors within building spaces. Hence these CFSs can't even be quantified in the discrete representation of a BSDF as it would require infinite data at the worst and very large amounts of data at the very least to store all the possible behaviors of the CFS. Finally, the actual solution algorithms used for the models introduce errors which cannot be eliminated, for example, the well known limitation of geometric optical models inability to model diffraction.

2.2.4 Physical-Modeling/Simulation

As stated above, developing mathematical-models/simulation techniques that accurately describe light's behavior is a well established and challenging problem. As such, building physical models of proposed daylighting systems is still common practice. Daylight models are constructed at either full scale or reduced scale, and depending on the purpose of the model, will exhibit different properties.

Full Scale

Full scale daylighting tests are quite rare, even with the overall size of the daylighting community being quite large. This is a direct result of the extensive investments required to make such models, instrument them, and finally perform simulations to examine their response. An example of a facility capable of such full scale daylight testing is the FLEXLAB at Lawrence Berkeley National Laboratory (LBNL) in Berkeley, California [47]. This laboratory offers many cutting edge facilities such as side-by-side test bays, a rotating test bay capable of "following" the sun, as well as extensive collections of sensors and instruments to monitor experiments. Many experiments have been conducted in the laboratory, however, true to form, they are quite expensive and took a large effort to implement.

Another example of a full scale model and simulation conducted for daylighting, as well as hygrothermal analysis, was done in Singapore in preparation for the famous Gardens by the Bay [50]. Greenhouses 1, 2, 3, & 4 located at Horticulture Park were developed in order to ensure the complicated building controls and systems would actually perform as intended. The climate of Singapore is a harsh tropical one which frequents 80-90% humidity and temperatures in the 90°'s Fahrenheit. Seeing actual full functioning greenhouses were constructed for this full scale model, the costs were quite significant, however, given the scope and national prominence of the final project, it was a good investment to ensure it preformed as planned.

Reduced Scale: Model Types

Scale models for daylighting analysis come in many forms, with variations tailored to different types of analysis. In general, these analysis can be grouped into the classes of qualitative (i.e. understanding general behavior, yet not concerned with specific values) or quantitative (i.e. concerned with accurate phenomena valuations). Regardless of model type and analysis, a large body of best practices has been developed with Bodat *et al.* [21] and Reinhart [128] offering both practical and theoretical advice. A selected list of reduced scale daylighting models are presented below.

Massing Model: These models are simply shaped, opaque masses which are used to qualitatively study how light falls within a space. Generally, constructed of foam, cardboard, or even 3D printed, they are done on the building scale, when considering multiple buildings in close proximity where their shadows will interact. These models can be used on heliodons, outdoors, as well as in sky domes.

Daylight Penetration: Arguably the most complicated, these models aim to determine how much, and where, daylighting will fall within a floor plan or on the walls and ceiling of a proposed building. Both qualitative and quantitate variations exist, with the material properties and geometry of the model replicating that of the real system critical for the later [145]. Even small deviations in reflective properties of the model versus system can have significant affects as multiple bounces of light compound the effect. Illuminance meters can be used with these models to predict actual indoor illuminance, however, errors must be accounted for in the results analysis.

Shadow Analysis: These are simple geometrically precise models which are put on heliodons in order to study how the direct beam radiation from the sun will interact with the building geometry. For these studies, the most important factor is to ensure no *parasitic light* enters the model through transparent walls or other light leaks. One of the main advantages of this analysis is actually seeing unique light patterns present in the space. *Erradicts*, or random beams of light within a building only visible for certains periods of the year, are one daylighting feature which can be effectively studied with these models.

Combinations: Different model types can be used in conjunction. Take for example, using a *massing model* next to a *daylight penetration model* in order to represent an opaque building adjacent to a proposed building under analysis. The massing model will influence the daylighting penetration model significantly, however, doesn't need to have the corresponding level of detail. This model combination may be used in an overcast sky simulator or on a heliodon.

Reduced Scale: Excitation Generators

While various reduced scale model types exist for the study of daylighting, they typically need to be subjected to some lighting excitation in order to learn about their behavior. Several specialized instruments exist for scaled model daylighting simulations, a selection of which are listed below.

Overcast Sky Simulator: These instruments, also known as *mirror boxes*, have mirrored walls and an overhead source of light. The overhead light source is turned on, causing light to be reflected off the mirror walls, creating a "uniform" source of luminous excitation. The goal is to reproduce the CIE Uniform Sky [39] which is used to excite a scale model. An example of an Overcast Sky Simulator can be found at the University of California, Berkeley at the Center for the Built Environment (CBE) in the College of Environmental Design. Another exists at the University of Idaho. These tools typically are used with illuminance meters within the model and on top of the model to calculate the metric *Daylight Factor (DF)* [87].

Sky Dome: Also known as artificial skies, these instruments attempt to reproduce the entire non-uniform sky vault as closely as possible with respect to realistic luminous excitation sources. This is accomplished through the use of thousands of individually controlled lamps oriented in a hemisphere. Often realizing one of the 15 standard CIE Sky Models [39], these instruments too excite scale models, however, are significantly more expensive and specialized than Overcast Sky Simulators. Important considerations for sky domes are parallax errors, well described by Mardaljevic [99]. A well known example is the Sky Dome at Bartenbach LichtLabor, Innsbruck, Austria with adjustable brightness, color temperature, and light distribution [146].

Heliodon: These instruments consists of an adjustable table surface, called the "table," to which models are attached, and a collimated light beam produced to represent the Sun. The table can then be adjusted such that the collimated beam is in the same position as the Sun would be for any given day of the year and time of that day. This allows for mostly qualitative analysis of the direct beam component of the Sun and how it interacts with the building. Cameras can be used to capture the light distribution within a space, helping designers and owners visualize what the final building will look like at different times of the year. It should be noted, however, traditional heliodons only study direct beam radiation, as their light is simply a collimated beam. Another type of heliodon, the *scanning heliodon* is changing this, and making a heliodon capable of measuring phenomena well beyond that the direct beam.

Scanning Heliodon: Very new on the daylighting scene, and relatively very expensive, scanning heliodons combine a section of a Sky Dome, called a Sky Patch, and a computer controlled traditional heliodon table. By continuously moving the table and adjusting the

Sky Patch lighting distribution, a simulation of the entire sky can be produced during a scan. These tools are newer on the market, however, and the daylighting community is still adjusting to their use [138].

Outdoor Sunlight: Often simply taking a model of any type outside and allowing real daylight to excite it can offer "insight" and "inspiration." Further, devices exist for heliodon like studies under real sun, sometimes called "pocket heliodons." Aligning the "pocket heliodon" in the proper orientation then take the currently available direct sunlight and uses it as a heliodon. Even bringing a material sample under real daylight can change a designer's perspective with respect to how it will behave in real conditions. Finally, design sessions outdoors often are communal type events, where brainstorming occurs, producing innovations in the design.

Sensing Model Response

Once the model is built, either full or reduced scale, and excited, measurements are the final step in simulation. Recalling the two major uses of daylighting models, qualitative and quantitative, sensors fall into different categories and can be used in different ways.

Illuminance Meters: These sensors are ubiquitous in the engineering world and are used to measure the total incident electromagnetic radiation falling on a given point, cosine-weighted for incident angle and luminous efficacy ($V(\lambda)$) weighted for wavelength. More intuitively, illuminance can be thought of as the total light falling on some point from all directions, quantified in the unit of [lux]=[$\frac{lumens}{m^2}$]. Illuminance meters come in many varieties [5], with the most accurate accounting for cosine effects and properly adjusted luminous efficacy filters traceable to National Institute of Standard and Technology (NIST) sources [92, 106]. These sensors can be used to both measure stand along illuminance or daylight factors within a space. Prices range from tens of US Dollars for approximate sensors, to over five hundred US Dollars for best available.

Luminance Meters: Luminance sensors are specialized "gun" type sensors which can be aimed at points within a space and which calculate the light leaving that point and arriving at the observer. Informally, these sensors measure the "brightness" of a surface, which is done quantifiably via units of [$\frac{cd}{m^2}$]=[$\frac{lumens}{sr \cdot m^2}$]. Luminance meters, however, are limited to measuring one spot of a scene at a time, typically with a field-of-view of 1°, making measurements of the luminance of several points exceedingly slow. To capture large areas of luminance, digital cameras can also be used, where each pixel is an individual luminance meter [64].

Digital Cameras: Digital cameras of various types can be used to capture the behavior of a scale daylighting model for future analysis or to share with owners. These cameras can be used inside a space to capture a simple qualitative photograph, or used to capture high-dynamic-range-images (HDRI) which can be calibrated for actual luminance measurement [$\frac{cd}{m^2}$] [64, 104]. Small cameras can also be placed inside models, offering fields-of-view humans can't access without adding parasitic light. Cameras can be used inside models on heliodons, overcast sky simulators, and sky domes.

Qualitative Feedback: The impression given of a proposed design when actually viewed via a scale model illuminated with real light can often be quite telling with respect to the ultimate human experience goals of a design. This non-quantifiable measurement often has confirmed designs as a final choice or offered new insight into a design, improving the final system in a way mathematical modeling and simulation currently can't do.

Advantages and Disadvantages

The advantages of physical modeling on both the full and reduced scale are many. First off, the actual physics of the system being investigated are actually being used, offering confidence in the fact that so called "modeling errors" and "computational errors" are not possible. Further, assumptions within the model tend to be well understood, for example geometrical scaling. In addition, physical modeling offers the ability to examine systems where the currently implemented mathematical models (i.e. ray models) employed in daylighting are known to fail, for example daylighting systems involving diffraction or interference.

In conjunction with these and other advantages, disadvantages do come into play with physical modeling for daylighting analysis. The first and most obvious is cost. Cost comes into play well beyond monetary considerations and involves the extensive time, materials, and needed technical expertise to conduct a physical model investigation. Note, these can even involve seperate building construction projects as mentioned in the Singapore full scale model section above. Further, instruments to excite scale models tend to be rare and must be maintained with consistent calibration of sensors and illuminant sources, often on time scales as short as every two years. Also, the laws of similitude, and practical model construction practices, limit the materials which can be used for model construction, often limiting the critical equating of material properties between model and system. For a well done comparison with meaningful commentary of full scale, reduced scale, and mathematical model of a daylighting application, see Thanachareonkit *et al* [145], Thanachareonkit and Scartezzini [144], and the accompanying thesis by Thanachareonkit [142].

2.2.5 Hardware-in-the-loop-Modeling/Simulation

Knowing energy use and environmental quality are top priorities of today's research engineers, chapter 1 established *complex fenestration systems* (CFS) as a key area of research within buildings. This establishment comes from knowing CFSs affect buildings in many facets via their control of sunlight's impact. As such, with respect to the subset of daylighting, CFSs are undergoing an explosion in innovation and development. Using computational tools such as Grasshopper [55] for Rhino [130], CFSs of intriguing novelty are being conceived. Coupled with the increasing presence of 3D printing technology (i.e. additive manufacturing), these conceptions are actually being realized in physical form [44]. With this large number of new building systems currently under development, designers and engineers need analysis methods which can be used to faithfully predict the behavior of such systems in real buildings.

Examining the currently available daylighting analysis techniques listed above, one will notice a shortcoming for analyzing CFSs being rapidly produced via computational design and 3D printing/rapid manufacturing workflows. Simplified techniques show obvious shortcomings, as the very nature of the operation of CFSs contradict the conditions of simplicity necessary. Considering mathematical modeling techniques, such as Radiance, one too finds limitations. For example, one could use a goniophotometer to measure a BSDF Front and Back of a newly 3D printed CFS, but perhaps the nearest instrument is thousands of miles away, requiring long shipment times and cost. Further, BSDF's are static measurements, making it necessary for augmentable CFSs (which are growing in number year by year) to be measured for each possible configuration. For CFSs with continuously varying configuration via control inputs, BSDF measurement then becomes untenable. Further, one could use *genBSDF* from the Radiance suite [102], but it too requires many material property parameters, and while validated for some examples [107], could very possibly fall short of capturing the true behavior of the new CFSs. Physical modeling involving the new CFS is thus the most practical for rapidly developed CFSs, especially those with continuously augmentable components, or unique daylight modifying elements. One then must decide regarding full or reduced scale for the CFS investigation.

While mathematical models, such as Radiance, struggle to handle daylight transfer through a CFS, they have been well validated for daylight transfer from both the sky to the outside of the CFS and also from the inside of the CSF to the interior surfaces within a building. In fact, this is the premise of the well known three-phase method [101], see Figure 3.5. With this reasoning in mind, the analysis of rapidly produced, novel CFSs for daylighting analysis lends itself well to a *hardware-in-the-loop modeling*/*simulation* (HWiL) investigation. For this analysis, the daylighting excitation of the CFS and the transfer of light through the CFS will occur in the physical modeling domain. The complementary propagation of light exiting the physically realized CFS within the build environment will occur not in a physical model of the space, but rather in a mathematical model of the space realized as a Radiance model. Radiance takes as input the measured light leaving the CFS and calculates any desired daylighting behavior given the CFS input. This HWiL model and the subsequent simulations will then be linked via measurement of the luminance distribution exiting the CFS and any control inputs to the CFS which are based on Radiance calculated response metrics. Examining Figure 3.6, one can see the various components of the HWiL model with explicit demarkation of the physical and computational components needed for the subsequent simulations.

With the HWiL modeling/simulation framework well motivated above, further details on HWiL models and simulations will be provided in chapter 3. First, an abstract overview of the HWiL method will be provided, then an indepth component wise breakdown of the proposed HWiL model for this thesis, the *CUBE 2.0*, will be provided.

Chapter 3

Hardware-in-the-Loop Architecture

In chapter 2, daylight as it relates to buildings was overviewed, with a motivation for the study of *complex fenestration systems* (CFS) using a hardware-in-the-loop (HWiL) model presented. In this chapter, HWiL modeling/simulation will be explained in an abstract form. HWiL as it relates to the proposed HWiL model including CFS for daylighting in buildings will also be presented. The components of both the physical and computational domains, and the interface between these domains, will be delineated, all of which together are referred to as the *CUBE 2.0*. In chapter 4 a more in depth treatment of the design and construction of the herein proposed CUBE 2.0 HWiL model will be given.

3.1 HWiL as an Abstraction

As stated earlier in chapter 1, section 1.2, hardware-in-the-loop modeling/simulation is an analysis technique combining both physical-models and mathematical-models into a new model which is then simulated to ultimately investigate some system.

Physical-models exist in the *physical-domain* - the physical world in which all observable matter and phenomena occur (i.e. the *Universe*), whereas mathematical-models exist in the *cyber-domain* - the world of computation as exists in the abstraction of integrated processors. A system with components in both of these domains forms a *cyber-physical system* [89]. *Actuators* are devices which take input from a digital state value in the cyber-domain, either directly or in analog via a digital-to-analog converter, and based on that value cause some excitation in the physical-domain. Examples include hydraulic and pneumatic cylinders and servo motors. *Sensors* are devices which convert some physical excitation from the physical-domain into a digital state value in the cyber-domain through the use of an analog-to-digital converter (ADC). Examples include charged couple devices (CCD) (i.e. digital camera sensing chips), thermocouples, and the classic strain gage.

Together, actuators and sensors bridge the cyber-physical domain divide, and allow for the linking of the mathematical and physical models which comprise a HWiL model.

3.1.1 HWiL Conceptual Architectures

Depending on the background of the practitioner involved, a hardware-in-the-loop model can take one of two equivalent conceptual architectures [139]:

1) **Numerical-Analysis Concept (NA)** [136]: Here an appropriate numerical model (i.e. mathematical model) is constructed of the entire system, including “special elements” with boundaries coincident with the physical models. These “special elements” are then replaced by the physical model’s behavior with regard to imposed and measured boundary conditions during simulation. A popular choice is a Finite-Element-Method model of the system, with physical specimens replacing the select elements. Hence, during each step of the solution of the FEM model, measurements and actuations are directly involved in the numerical solution technique being used to “step” the model forward in time.

The NA conceptual architecture is preferred by engineers who are familiar with spatially continuous dynamics models, for example structural engineers using Finite-Element-Methods in civil, mechanical, and aerospace.

2) **Controller-Plant Concept (CP)** [51]: In this conceptual architecture, the mathematical model is simulated on the “controller,” with *sensor inputs* and *actuator outputs* connecting it to the physical model which is representative of the “plant.” As opposed to the NA architecture, the measurements are part of “inputs” to the numerical solver and the “outputs” of the numerical solver are valued used for the actuations. The CP conceptual approach is common in the modeling and controls community, often more familiar with controlled systems like internal combustion engines (ICEs) and oil refining processes than Finite-Element-Methods.

As such, one can generally think of the NA architecture involving the measurements and actuations in the numerical solution technique itself, while the CP architecture uses a more traditional numerical simulation technique which uses the measurements as inputs and the outputs as the next actuation value.

Both the Numerical-Analysis and Controller-Plant conceptual architectures offer different perspectives, yet are equivalent in modeling and simulation semantics [139]. This thesis will focus on the Controller-Plant conceptual architecture, as many principles are easier to express given this approach and it was used when designing the CUBE 2.0 system in this thesis. As such, Figure 3.1, shows a general schematic representing the main components of a HWiL from the Controller-Plant perspective.

3.1.2 Actuator and Sensor Dynamics Compensation

While a real system is by its nature continuously linked through all components, a HWiL model is distinctly discontinuous between the components residing in the physical and cyber domains. This seperation within the total HWiL model is made continuous, like the system it represents, through actuators and sensors linking the two modeling domains, see Figure 3.1.

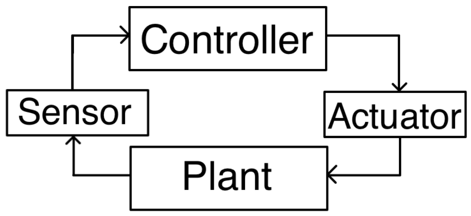


Figure 3.1: Controller-Plant framework based schematic representing a HWiL model.

In an ideal world, this linking of the cyber and physical domains would be seamless, with actuators imposing exact excitation in the physical-domain and sensors delivering precise readings to the cyber-domain. The introduction of these additional components in the model, however, brings additional dynamics when the model is eventually simulated. With respect to these dynamics, HWiL models fall into two general categories: those whose actuator and sensor dynamics are negligible, and those whose are not.

Actuator and Sensor Dynamics Negligible:

Many examples exist within the HWiL modeling and simulation community whose actuator and sensor dynamics are negligible with respect to the system dynamics. When actuator and sensor dynamics are negligible, the HWiL model architecture presented in Figure 3.1 can be implemented directly.

Examples include the entire class of *pseudodynamic HWiL modeling and simulation* found in the civil-structural engineering community [135]. Here, the physical model exhibits a time independant response (see the section below for more details on timing), hence, the actuators and sensors are controlled in a slow and precise manner, keeping the addition of unwanted dynamics out of the model.

Another example comes from the classic application of HWiL modeling for *engine controllers* (e.g. diesel and gas internal combustion engines, jet turbines), with the embedded control unit (ECU) being physical and the engine being a mathematical model [65]. Two factors in this case make the actuator and sensor dynamics negligible. First, the physical model only interfaces with the actuator at discrete times (i.e. when it samples a sensor on the engine), hence the actuator only needs to be "correct" for a short period of sampling during each cycle of the simulation. Second, the actuators tend to be digital-to-analog converters with fast response times and good control, as opposed to hydraulic cylinders which are slow and exhibit many difficult control phenomena such as *overshoot* and *relaxation*.

Actuator and Sensor Dynamics Not Negligible:

Contrary to the simpler examples listed above, the actuator and sensor dynamics of many HWil models are substantial with respect to system response (e.g. hydraulic cylinders, dynamometers), and hence require compensation to produce accurate simulation results.

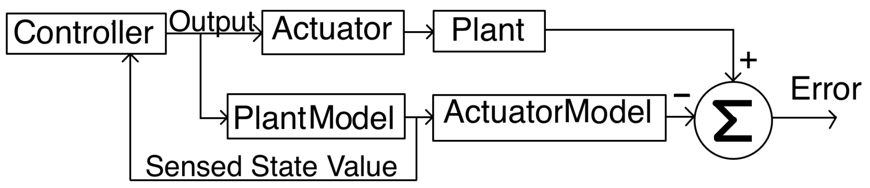


Figure 3.2: Controller-Plant Conceptual Architecture with no Actuator and Sensor Dynamics Compensation. Diagram inspired by Sivaselvan [139]

Compensation for this unwanted addition of dynamics comes in different forms, with the most popular all taking the flavor of a *state-estimation framework* [139].

State-estimation frameworks combine a *mathematical model representation* of system states and *sensor measurements* of system states to estimate the value of state variables of interest. This combine estimation then takes on a higher confidence than either the mathematical model estimate or sensor estimate would have alone[1]. Examples of state-estimators are many and comprise a rich field of study in themselves, with perhaps the most well known being the Kalman Filter [158].

Examining Figure 3.1, we see the entire system we desire to represent is accounted for in the union of the plant and controller. Thus, we seek the goal of estimating the behavior of the plant alone and rejecting the behavior of the actuators and sensors. The estimated plant state values can then be used to interface with the mathematical model, forming a seamless system model which can be studied for engineering purposes. In order to accomplish this, mathematical models of the physical model, actuators, and sensors are simulated along side the computational model. Further, the actual response of the physical models, actuators, and sensors are measured. Figure 3.2 shows this process, at which point the modeling errors have been quantified.

These modeling errors can now be used in different manners to account for the undesirable actuator and sensor dynamics introduced into the HWiL model. One manner in which they can be used, is to directly correct the mathematical models of the physical specimen's response before it is feed back into the mathematical partition of the system. This is known as a Smith Predictor architecture and is common in hydraulic cylinder control and can be seen in Figure 3.3. The modeling errors could likewise be handled in many other forms, for example a Luenberger Observer form, not shown here.

Regardless of the exact manner in which modeling errors are handled, the end result is a HWiL model architecture which accounts for actuator and sensor dynamics. With these compensation methods in mind, and seeing the *pseudodynamic structural models* HWiL simulation avoided compensation for these added dynamics by adjusting the HWiL simulation timing, one may ask why don't all HWiL simulations do the same thing.

[1]Knowing the "true value" of a state is impossible, as sensors and mathematical models will always have error, leaving only better estimates of state variables as a practical engineering goal.

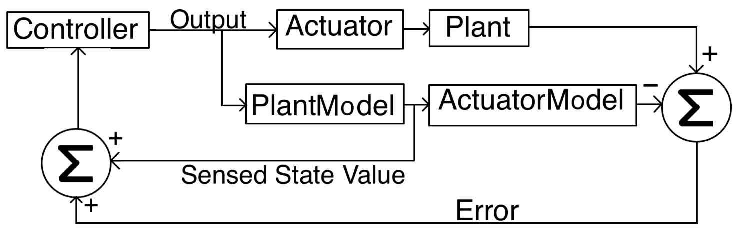


Figure 3.3: Controller-Plant Conceptual Architecture with Actuator and Sensor Dynamics Compensation in Smith Predictor architecture. Diagram inspired by Sivaselvan [139]

While it seems appealing, this suggestion won't succeed in general. For pseudodynamic testing of some physical structural models, the physical model's behavior is independant of time, in general this is not true. In fact, many physical models exhibit behaviors that are highly time dependent, for example visco-elastic properties of polymers or thermodynamic processes in building heat transfer. As such, the timing protocol of a HWiL simulation is highly important and falls into one of three categories: 1) slower-than-real-time, 2) equal-to-real-time, or 3) faster-than-real-time.

3.1.3 Timing of a HWiL Simulation

The rate at which a HWiL simulation is executed is governed by the needs of the physical model subcomponents of the HWiL model being investigated. This fact stems from the simple idea that *real time* - the rate at which time procedes in the physical domain - cannot be altered in a practical manner and as such, all other HWiL model components must bend around its mandates on physical reality. With respect to timing, a HWiL simulation will be executed either: (i) Faster-then-real-time, (ii) Slower-than-real-time, or (iii) Equal-to-real-time. Timing with respect to hardware-in-the-loop is a deep topic relavent to many fields [117, 135, 65], hence a short, far from exhaustive, overview of the possible configurations is given below.

Faster-then-real-time:

In these HWiL models/simulations, the process which is being modeled is typically a slow one (e.g. thermodynamics in buildings, oil refining batch plants, hydroelectric dams [108]). As such, advantages are gained by simulating *faster* than the real phenomena would occur if the whole system were actually being experimented on. This allows for the investigation of more of the possible behaviors of a system in the same amount of time.

Another example, is when using a scaled physical model subcomponent in the HWiL model whose *similitude* requirements mandate increasing the rate of time to get the proper alignment to the full scale system's behavior [59].

Limitations for these investigations include how much the physical components can be speed up to meet the increased time speed. For example, the analog-to-digital converter, digital-to-analog converter, and physical controller must react quickly enough to keep pace with the simulation. Or in another experiment, the simulation must keep pace with a faster than real time physical timing.

Slower-than-real-time:

In some circumstances, a process will be run *slower* than the real phenomena. This may be done as the physical model will exhibit limited timing dependency of behavior, or the simulation is too complicated to solve in the time step needed to maintain real time HWiL simulation speeds. As such, the simulation is slowed down, giving more computation, actuation, and sensing time, yet not introducing timing errors because the physical model doesn't exhibit timing dependent behavior.

Here, issues can arise including maintaining boundary conditions in the physical model as phenomena such as *actuator slip* and *specimen relaxation* can occur.

Equal-to-real-time:

HWiL simulations running at real time are common and are used when the physical components need to be simulated as if they were behaving as they would in the real system. Here maintaining pace of the mathematical model simulated in the cyber domain to that of real time is critical in order to stay paced with the physical domain's notion of "real time." In reality, due to actuation and sensing time requirements, the mathematical model simulation in fact must run faster than the allocated time step size to allow time for sensing and actuation. As such, coordinating the timing of cyber and physical domains is the main issue, requiring certain error tolerances which must be maintained through out the simulation.

3.2 HWiL Applied to Building CFS

In chapter 2 section Hardware-in-the-loop-Modeling/Simulation, a hardware-in-the-loop (HWiL) model was motivated for the analysis of *complex fenestration systems* (CFS) as components in innovative building daylighting systems. The architecture of this proposed HWiL model will now be expanded upon in more detail.

The notion of using hardware-in-the-loop techniques with respect to CFS as part of building daylighting systems is the key contribution of this thesis and is seen as an innovation in the daylighting community as no known examples of HWiL modeling and simulation are known.

The HWiL idea was originally outlined in Mead *et al.* [103] and is overviewed first. In the course of executing this thesis work, however, the three-phase method was discovered by the author [101]. The three-phase method offers a more succinct input interface for the measured luminance data with respect to the Radiance model than the originally proposed solution.

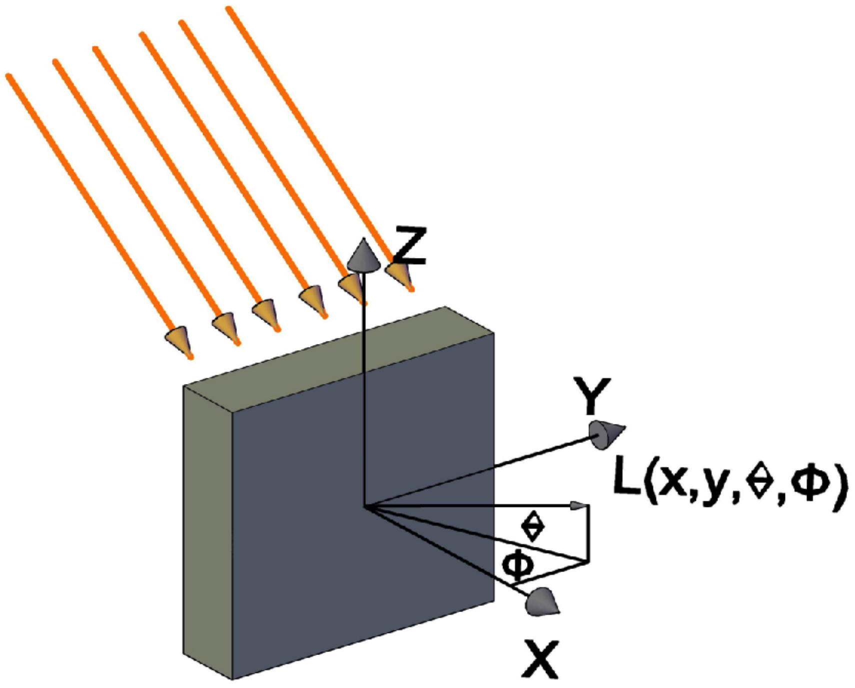


Figure 3.4: Diagram showing the exiting luminance distribution in functional form relative to an abstract CFS panel.

Hence, an adaptation of the three-method was adopted as the implemented architecture of the HWiL model for this thesis, which is presented below.

3.2.1 Original Proposal Based on IES files

While no longer the implemented solution in this thesis work, the original architecture of the HWiL model is presented here for completness. A more detailed examination of the system can be found in Mead *et al.* [103].

Beginning with the physical domain, the CFS panels which make up the building envelope for a daylighting system are first excited using some form of luminance energy on their outside face. This luminous energy can be either natural (i.e. sunlight in outdoor testing) or artificial (i.e. electric luminaires in a laboratory). Then, the exiting luminance distribution associated with this excitation will need to be measured leaving the inside face. This measurement is proposed to be executed by an array of complementary metal-oxide semiconductor (CMOS) sensors, organized spatially and coupled with an analysis program to produce a measurement of $L(X, Y, \theta, \phi)$ shown below in Figure 3.4. The measurement as conducted by the CMOS sensors is the transition from the physical domain to the cyber domain.

Now in the cyber domain, the measured exiting luminance, $L(X, Y, \theta, \phi)$, will then need to be input into a Radiance model. For this, a well established interface of luminous energy to Radiance, the IES file, is proposed. IES files, formally called, "Standard File Format for the Electronic Transfer of Photometric Data" and defined by the Illuminating Engineering Society standard LM-63-02 [12], are the typical mechanism to introduce luminous energy into a Radiance model. Ordinarily, these files encode the exiting luminance distribution leaving electric luminaires, yet are abstract enough to represent any luminous energy source codified by a luminance distribution.

As such, to introduce the measured exiting luminance distribution from a CFS panel under test into the Radiance model (Note: the Radiance model can be any arbitrary space), an array of IES objects can be instantiated in an array. These objects are defined through the IES files and the Radiance program *ies2rad*. This array thus introduces what would be the luminous energy exiting the building enclosure comprised of the CFS undertest into the modeled space. This Radiance model can then be simulated to solve for various luminous metrics as desired in the design process.

One draw back of this setup, however, which was key to the choice to adopt the three-phase inspired method, is with what geometrical properties to instantiate the array of IES file generated objects. This array must somehow represent the measured exiting luminance in such a manner as to have it represent the real CFS of a building enclosure. Many ideas were conceived, such as an evenly space grid or a point for point instantiation of the proposed CFS enclosure wall. In the solution proposed below and inspired by the three-phase method, the Radiance programs *rtcontrib* and *genklemsamp* allow for the Radiance model to do the "heavy lifting" of defining this array, making this challenging problem a non-issue. This is because the three-phase method doesn't use an array all, but rather assumes a homogeneous CFS. Thus, the measured exiting luminance distribution, $L(X, Y, \theta, \phi)$, for a small CFS sample can simply be assumed to represent that of the whole CFS constructed enclosure section. Further, the three-phase method allows for precomputing a large portion of the Radiance simulation, saving valuable time per time step in the HWiL simulation.

3.2.2 The Three-Phase Method

While the above architecture will certainly work for a HWiL model, a different architecture inspired by the three-phase method was conceived and developed during this thesis work. This new conception addresses some shortcomings of the above proposal, most prominently, the organization of the the IES files to faithfully represent the CFS containing enclosure.

Examining Figure 3.5, one sees a simplified schematic view of a building which is designed using daylighting. There are three distinct zones, "outside," "inside," and the "building envelope" in this diagram, each corresponding to a different segment of the path daylight takes from the sun and sky into a building. Below the schematic, the mathematical representation of this path is expressed using the notation from McNeil *et al.* [101].

Looking closer, $s \in \mathbb{R}^{145\times 1}$ is a vector representing luminance $[\frac{cd}{m^2}]$ coming from the sky vault including the solar disk. The discretization scheme of the sky vault for this vector uses the Tregenza Basis, originally proposed from the Daylight Coefficient method [149]. Each element in the vector is an average of the luminance over a certain θ and ϕ range as related

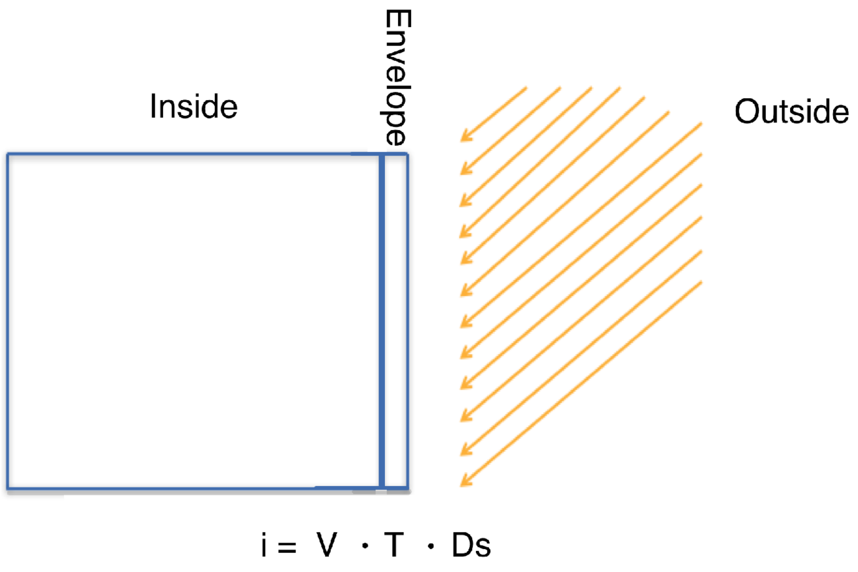


Figure 3.5: Conceptualization of the three-phase method used to model building daylighting systems.

to the global coordinate system of the model. Next, $D \in \mathbb{R}^{145 \times 145}$ is called the "daylight matrix," and corresponds to a mapping of each individual Tregenza sky division to the incoming Klems Basis discretization of the CFS under consideration. Hence, $Ds \in \mathbb{R}^{145 \times 1}$ is the total illuminance $[\frac{lumens}{m^2}]$ per incident steradian on the *outside* of the CFS. The $T \in \mathbb{R}^{145 \times 145}$ is none other than the *bidirectional transmission distribution function* discussed in the Glazing Material Properties section above. T therefore requires either measurement or modeling and simulation to determine. Finally, $V \in \mathbb{R}^{p \times 145}$ is called the "view matrix" and represents the path from the 145 exiting solid angles of the CFS under study, to the inside of the building to whatever p instantiations of the illuminant metric Radiance is calculating. An example maybe a grid of virtual illuminance meters some a by b in size. This grid, such that $a \times b = p$ with $a, b \in \mathbb{N} \backslash \{0\}$, is thus represented in the final value $i \in \mathbb{R}^{p \times 1}$ in the diagram.

3.2.3 HWiL Architecture: Cyber and Physical Partitions

With this framework established, the proposed hardware-in-the-loop model can be described explicitly with respect to building daylighting systems. First, a physical complex fenestration system (CFS) will be constructed and excited with real luminous energy. Considering the luminance distribution leaving the CFS, this collectively represents elements s, D, and T from the above diagram multiplied together, TDs. The remaining term, namely V is what is modeled mathematically in Radiance, resulting in the final illuminant met-

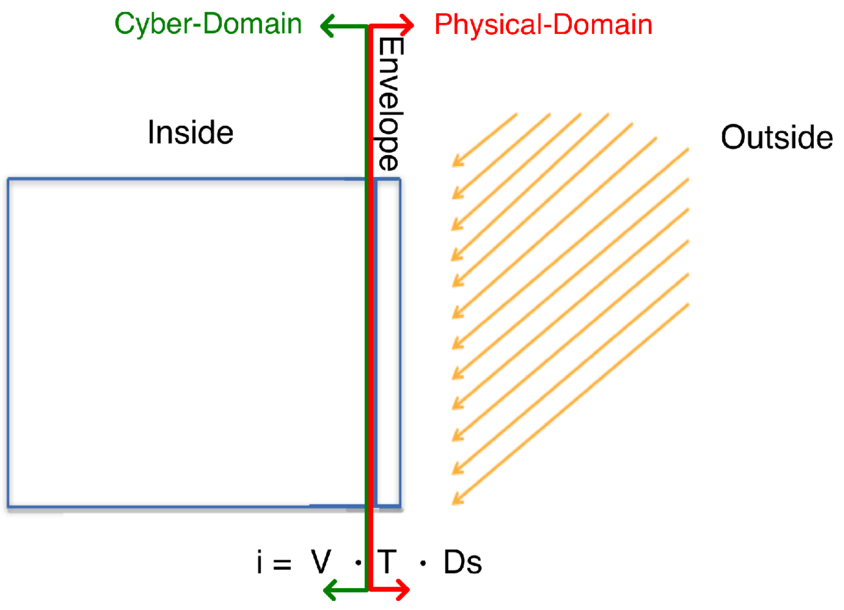


Figure 3.6: Three-phase method delineated with cyber and physical domains as required by the proposed HWiL model of this thesis.

rics which are desired. This partitioning of the daylighting system between the cyber and physical domains is shown in Figure 3.6.

This daylight system partitioning in a hardware-in-the-loop architecture is the first of its kind within daylighting analysis known to the author. By using physical specimens for the difficult to mathematically model components (i.e. the complex fenestration system), and a mathematical model for the distribution of light within the space, an optimal combination of model fidelity is reached.

Extending beyond the three phase method, the proposed HWiL will also “close-the-loop” of the HWiL model. This is accomplished by taking the values of the illuminant metrics, calculated by Radiance, and feeding them into a control law which can adjust the CFS based on simulated conditions. While not all CFS are augmentable, some are and this feature is critical to engaging their behavior for *in situ* conditions in buildings.

3.2.4 HWiL Architecture: Sensing and Actuation

With the cyber-physical partitioning of the daylighting system now established, attention will now focus on linking the two through sensing and actuation.

Sensing

Examining Figure 3.6, it becomes clear the light exiting the CFS undertest, will need to be measured. More specifically, the directionally varying luminance distribution, $L(x, y, \theta, \phi)[\frac{cd}{m^2}]$, exiting the back of the CFS when excited by a light source is the needed quantity. Further, this luminance distribution needs to be expressed in terms of a discretized Klems Basis measurement, denoted here as L_v. This measurement will take the physical-domain model's response and transform it into the cyber-domain. See Figure 3.4 for a visual representation of the luminance distribution which will be measured in the proposed HWiL model.

Considering the three-phase framework established above, this luminance distribution will take the form of a vector as if it was TDs. Thus, the measurement will be the function $L(x, y, \theta, \phi)$ discretized by the Klems Basis into a vector $L_v \in \mathbb{R}^{145\times 1}$. With this vector of luminance measurements, L_v can be directly inserted into the mathematical model constructed in Radiance. Implementation details of this sensing, as well as the entire HWiL model, will be presented in detail in chapter 4.

Actuation

While not applicable to all CFS, those with augmentable components can in theory be adjusted based on the results of the mathematical model's simulation, that is i_m in the above formalisms. Examples of CFS augmentation include the simple lowering of a roller shade or Venetian blind, to complicated geometrically altering CFS with electronic motors and detailed control laws moving rigid components to transform the transmission properties [143].

Seeing augmentation is CFS dependent, the proposed HWiL model leaves this notion abstract, however, in chapter 4 implementation details will be given regarding different options available for "closing-the-loop."

Chapter 4

Hardware-in-the-Loop System Implementation

In chapter 3, hardware-in-the-loop modeling/simulation (HWiL) was presented in both an abstract form, as well as, a concrete application to building daylighting systems. The building daylighting application is the main focus of this thesis and is refereed to as the *CUBE 2.0* system. The CUBE 2.0 system includes computational as well as physical sub-systems, which interact to form the super-system, making CUBE 2.0 a *cyber-physical system*. In turn, CUBE 2.0 is itself a model of the building daylighting system under study.

In this chapter, the required tasks of the CUBE 2.0 system are first presented in plain English. Next, the design process of the CUBE 2.0 cyber-physical system is presented. From initial design proposal, through modeling and simulation, redesign, and then concluding with the final design proposal for CUBE 2.0. Closing out the chapter is an overview of the actual construction process from both a cyber and physical domain, component by component perspective. Next, in chapter 5, the calibration process for the CUBE 2.0 will be presented in detail.

4.1 CUBE 2.0 System Specifications

Arguably the easiest manner to describe a complex engineered system is to state its operational goals in plain English sentences. While this has known limitations with respect to expressing objective statements, a task better left to tools such as *linear temporal logic* [89], it is a convenient method utilized here to describe the tasks needed by the CUBE 2.0 system. These tasks, both physical and cyber in nature, are discussed next in roughly the order of a HWiL simulation step. For more detail on the necessity of these tasks, see chapter 3 section HWiL Applied to Building CFS.

Light Excitation Source

The complex fenestration system (CFS) panels need to be excited in some manner in the physical domain. In a real building this is the Sun and the daylight it produces. In the HWiL model this will also need to involve physical electromagnetic radiation (EMR) as the CFS panels are physical models which need physical excitation.

Measure Input Illuminance Distribution

Measuring the directionally varying illuminance falling on the outside of the CFS undertest is needed for calibration of the system and validation efforts using previously quantified CFS. With respect to the three-phase method presented in chapter 3, this is a vector quantity $E_v \in \mathbb{R}^{145\times 1}$ representing $E_v = Ds$. Further, the proposed system identification techniques described in Appendix C will use this measurement as the "input" to the CFS system under test.

Measure Exiting Luminance Distribution

The directionally varying luminance exiting the CFS must be measured as this is the input for the cyber domain model of the interior space of the building. With respect to the three-phase method previously mentioned in chapter 3, this is a vector quantity $L_v \in \mathbb{R}^{145\times 1}$ representing $L_v = TDs$.

With E_v and L_v representing Ds and TDs respectively, the *bidirectional transmission distribution function* (BTDF), T, can be "learned" using system identification techniques, see Appendix C.

Simulation Execution, Component Coordination and Communication

Initiation and coordination of the above two measurements, as well as communication of the measured data to a computational platform for processing and storage is needed. Further, the measured luminance distribution leaving the panel, L_v, must be used as input into the mathematical model of the interior space, which is subsequently simulated.

The modeling/simulation package to be used is Radiance, hence it must be a UNIX platform. With respect to the three-phase method presented in chapter 3, this is the view matrix, V, representing the interior space which transforms luminance leaving the CFS, L_v, into illuminant metrics of interest to the designer and engineer, i_m, via the matrix multiplication, $i_m = V \cdot L_v$.

Next, with the results of the simulated model, various control laws can be computed for an appropriate augmentable CFS response. These control law values then need to be output to the CFS undertest in some usable format (e.g. pulse width modulation (PWM), analog voltage). In this thesis, the actual augmentation should be part of the CFS, with the CUBE 2.0 system providing a platform for computing control laws and outputting those signals in a number of standard protocols.

Physical Structure

Seeing there are actual physical components to this system and the processors must exist in real space, a structure is needed to define the entire system. This structure will hold the CFS undertest, allow for positioning of the specimens relative to light excitation, as well as hold together the various sensors and computational resources which comprise the entire system.

Electrical Power

Many of the components of the CUBE 2.0 system will require electrical power to operate. Two modes of power are needed, stationary deployments with access to grid power and mobile deployments with enough power to run the system for a complete 24 hour cycle.

4.2 CUBE 1.0

In the course of the hardware-in-the-loop engineering project described in this thesis, the CUBE 2.0 system is the end result of a long path of exploratory investigations. This path began with the abstract goal of "studying energy use in buildings," and ultimately landed on the very focused investigation of complex fenestration systems with respect to daylighting.

On this journey, the CUBE 1.0 system was originally conceived to measure the light transmission properties of various building enclosure panels, see Figure 4.1. While the CUBE 2.0 measures the directionally varying exiting luminance distribution, $L_v \in \mathbb{R}^{145\times1}$, CUBE 1.0 measures simply the *total transmission of light* through the panel being tested. This measurement is accomplished by placing two illuminance sensors vertically in parallel with the panel under test. The first sensor is on the outside and the second is on the inside of a small enclosed space behind the panel. The CUBE 1.0 is then exposed to excitation, using both real world sunlight and daylight simulators. By taking the second sensor measurement, E_v^{in}, and dividing it by the first, E_v^{out}, a simple uniform transmission, τ, can be calculated for the panel: $\tau = \frac{E_v^{in}}{E_v^{out}}$. This method of measurement is refereed to in this thesis as the transmission method of measurement. The results derived from using the CUBE 1.0 are the subject of the Casquero-Modrego Thesis [30].

While the *transmission method* offers an initial investigation into the properties of building enclosure panels, when it comes to CFSs it falls short. Specifically, the directionally varying nature of the exiting luminance distribution is completely ignored. Given the nature of the proposed HWiL model, this directionally varying nature must be characterized and used as input into a computational model. As such, the CUBE 2.0 was conceived and is explored below in extensive detail. The CUBE 1.0 is presented here for historical reasons and to explain the "2.0" suffix.

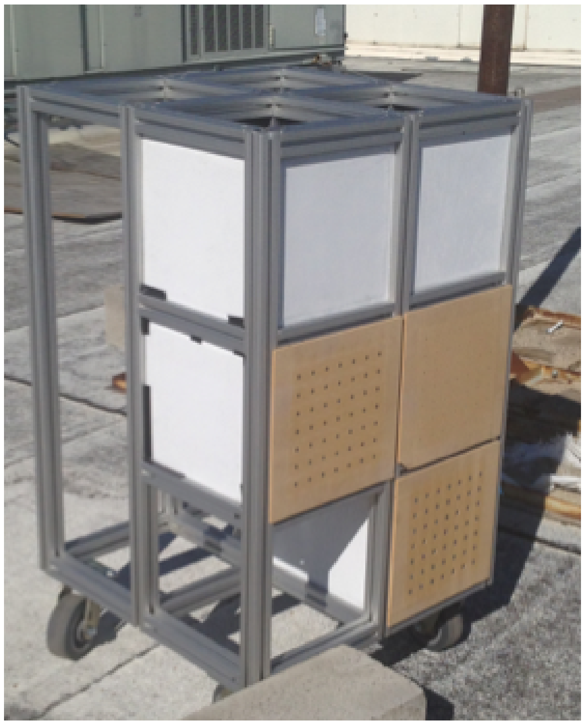


Figure 4.1: CUBE 1.0 which uses the *transmission method* to investigate building envelope panels.

4.3 CUBE 2.0 System Design

In this section, the design process of the CUBE 2.0 system is presented. While the CUBE 2.0 system is a HWiL model of a building daylighting system using CFS, it is itself a *system*. Hence, the modeling and simulation tools used in the analysis of the CUBE 2.0 are also presented. Concluding, the final design of the CUBE 2.0 system resulting from the insights learned in modeling and simulation is presented. This section organizes the discussion of the system components roughly in the chronological order of a HWiL simulation step.

4.3.1 Initial CUBE 2.0 System Design

Every system design process must start with an initial design proposal. Below is an overview description of an initial design proposal for the CUBE 2.0 system which is designed to be able to fulfill the above stated system specifications. Subsequently, various analyses are applied to the proposed system design, both as a whole and to individual components. These analyses ultimately inform the proposed design, resulting in the final design.

Light Excitation Source

Two operational modes are envisioned for the CUBE 2.0 system with respect to luminous excitation. First, extended outdoor testing under real daylight (i.e. one to several days continuously (1, 5, 15, or 60 minute time steps)). This is the ultimate light excitation source as it is the real source the system will also undergo in practice. While the sun varies continuously from minute to minute and location to location around the world, seeing real luminous system excitation is being used, daylighting system behavior can be studied using this excitation with great confidence in results. Second, daylight simulators for specific and repeatable excitation conditions will also be used. These could be overcast sky simulators, sky domes, or even heliodons for direct lighting component studies.

Measure Input Illuminance Distribution

In order to measure the input illuminance distribution a **Canon 6D** digital single lens reflex (DSLR) [28] with a **Sigma 8mm F3.5 EX DF** equisolid angle projection lens [137] is proposed. A Canon 6D is proposed as one is available for use and also Canon provides a software development kit (SDK), the Canon: Digital Image Developer Programme [27]. This SDK has a C API which will allow full control of the camera from a program running on the MacBook Pro. A Sigma 8mm F3.5 EX DF fisheye is proposed due to it's previous usage in the literature [63] and reasonable price of ∼$900.00 USD and availability.

The Canon 6D is remotely controlled through USB from the MacBook Pro, see below, to take a series of *exposure compensation* (EC) varying photographs, with EC=-5,..,0,..,+5 at one stop intervals. These 11 photographs, once downloaded to the MacBook Pro, are combined into a single *high dynamic range image* (HDRI) using *hdrgen*, the engine behind the popular Photoshere [13]. *hdrgen* is used as it is well validated and available free of charge. Further, it is considered to be the industry standard for building daylighting study. Once the HDRI has been formed, the XYZE formatted file [154] is parsed for exact pixel information of the luminance, CIE-Y [58]. The total collection of pixels will then be used to calculate the average luminance for each of the incoming *Klems Basis* solid angles [80, 81]. This averaging will produce a vector of measurements, $E_v \in \mathbb{R}^{145\times 1}$, equivalent to the three-phase method term $E_v = Ds$.

To ensure accurate measurements are obtained for the HDRI produced above, a vertical illuminance measurement is taken coincident with the Sigma Fisheye lens using a **Li-Cor LI-210** illuminance meter [92] for absolute luminance calibration. The LI-210 is a robust, outdoor illuminance sensor, with decades of use in meteorological measurements, with continuous design improvements, and hence is chosen as a dependable instrument here.

The LI-210, however, is a current producing sensor, hence, a 1% precision 604 ohm resister is used according to manufacturer specifications to convert it to a voltage [93]. This voltage is measured using a **LabJack T7-Pro** data acquisition system and will measure illuminance when signaled by the MacBook Pro main() program. The LabJack T7-Pro has an Ethernet jack and interacts with a proprietary TCP/IP protocol, hence it will plug directly into the

CUBE 2.0 network and can communicate with the MacBook Pro main() program seamlessly. Further, a freely provided Python Module from the manufacturer packages nearly the entire functionality of the LabJack T7-Pro into an object. Simplifying further the communication with the MacBook Pro. Moreover, the input range and bit-depth of the analog-to-digital converter (ADC) has more than enough coverage and resolution for the anticipated signal.

Measure Exiting Luminance Distribution

Raspberry Pi Camera Module and Optical Fibers: Measuring the exiting luminance distribution from the CFS undertest is accomplished using optical fibers and a Raspberry Pi Camera Module (RPiCM) (i.e. complementary metal-oxide-semiconductor (CMOS) sensor). The proposed design uses an acrylic hemisphere painted with matte black paint on the inside, that which faces the exiting side of the CFS undertest. At specific locations on the hemisphere corresponding to the central positions of the outgoing Klems Basis solid angles [80, 81], holes are drilled through the hemisphere and optical fibers are inserted. These optical fibers act as a discretized measurement of the exit luminance distribution. The other end of the optical fibers are bent around and inserted into a planer plywood board with their tip exposed to a "light proof" enclosure, called the *measurement cone*. Total internal reflection of the optical fibers thus transforms the three dimensional measurement of the exit luminance distribution into a two dimensional measurement using a projection of shorts through the optical fibers.

Optical fibers with an outer diameter of 1/4 inches and a core made of Poly(methyl methacrylate) (PMMA) are proposed, as large quantities are available, left over from the Translucent Concrete Panel (TCP) project [3]. In addition, several RPiCM, that is Raspberry Pi 2 Model Bs [120] and the associated Camera Modules [121], were acquired free of charge by the author. Further, the open source PiCamera Python Module [115] grants straight forward, Python level access to full camera module control, including RAW image and custom microsecond (μs) precision shutter speed.

The ends of the optical fibers for all 145 Klems Basis solid angles will point in the same direction, as they are secured in position by the plywood board. A digital camera can be used to capture a high dynamic range image (HDRI) of the ends of all the optical fibers at once, similarly to the Canon 6D with the fisheye lens. This photographic measurement will then be used to measure the corresponding light leaving the optical fiber ends, which in turn corresponds to the light leaving the CFS undertest for the specific directions. Each optical fiber end thus corresponds to one of the 145 elements of the vector $L_v \in \mathbb{R}^{145\times 1}$, equivalent to the $L_v = TDs$ term from the three-phase method.

In the design stage, it is uncertain how much light the optical fiber ends will transmit, hence the exact *exposure values* of the photographs to be taken are left open to determination. Further, the RPiCM is a stand along computing platform running its own version of Linux, called NOOBS [110]. The proposed design thus has the RPiCM performing all photograph gathering and processing locally when signaled by the MacBook Pro main() program. Communication is handled over Ethernet using TCP/IP sockets, hence by doing the computation locally, the network sees much lower throughput. The MacBook Pro sends one byte indicating it is time to take a measurement, and the RPiCM responds only with the 145 numbers corresponding to the final measurement[1]. Negating the need to transfer a set of EC varying photographs or even one HDRI photograph.

Simulation Execution, Component Coordination and Communication

MacBook Pro: A MacBook Pro running OSX 10.11 (UNIX based OS) will run the program orchestrating all the CUBE 2.0 system components. This main program, called "main()," is written in Python 2.7 [118]. Primary reasons motivating this include: i) the Sockets module will make TCP/IP communication easy, ii) data file management is exceptionally easy for simulation data reading and writing, iii) there exists an intuitive Python API for the LabJack T7-Pro over Ethernet, iv) the SciPy module, specifically NumPy which is used for matrix manipulation, is well validated and effective, amongst several others.

The MacBook Pro will also provide the needed computational resources for processing the Canon 6D fisheye generated photographs into a HDRI and simulate the Radiance model of the interior of the space. Further, any control laws based on the Radiance results can also be computed locally. The main() program will also control timing of each round of the simulation, as well as when a given simulation starts and stops.

A single "round" or "step" will consist of the following chronological events, execution of which are to be controlled by main():

1. Idle time in case the simulation timing requires it
2. Notify the RPiCM to take an exiting luminance distribution measurement
3. Take an illuminance measurement using the LabJack T7-Pro and the LI-210
4. Engage the Canon 6D with fisheye lens to take an EC varying LDRI sequence
5. Process the Canon 6D fisheye photographs and illuminance measurements into an input illuminance measurement
6. Get the RPiCM generated exiting luminance distribution measurement
7. Simulate the Radiance model using the exiting luminance distribution measurement
8. If required, calculate and impose any control law values
9. Save all data and return to Idle until next "step"

[1]In the real deployment several "housing keeping" bytes are also sent back and forth for state communication, however, these require negligible amounts of network resources compared to HDRI photographs.

The CUBE 2.0 Network: Given the distributed nature of the sensing and computing resources in the CUBE 2.0 (e.g. Canon 6D, LabJack, RPiCM), two communications protocols are proposed to be used. First, a local area network (LAN) over Ethernet is constructed. Over this network the various computational resources are able to communicate over TCP/IP sockets. The quick speeds and established protocols will aid greatly in building the system. The MacBook Pro, LabJack T7-Pro, and the RPiCM will all communicate via the LAN.

Second, the Canon produced EOS SDK [27] offers the ability to control a Canon 6D camera via universal serial bus (USB). Thus, a custom program is written using this SDK to run on the MacBook Pro which, when activated by main(), is used to control the Canon 6D with the fisheye lens.

In the design stage, the exact computing versus networking timing performance of these various components are unknown. While timing estimates are available for general computing and network transfer, a network model and simulation is used to explore the proposed design of the network.

Closing the loop - CFS Augmentation: The LabJack T7-Pro data acquisition system, mentioned above, also has various output mechanisms such as analog out (0-5 Volts) and digital out capable of pulse width modulation (PWM) and I^{2}C. Analog out, PWM, I^{2}C, and others offer possibilities for "closing-the-loop" with respect to the hardware-in-the-loop model.

Physical Structure

The CUBE 2.0 system is a physical object, hence a frame is proposed to be made from **8020 Aluminium sections** [1] to hold the various component together. This lite weight yet strong aluminum alloy material comes in standard lengths and profiles and is easily cut and machined using widely available band saws, drill presses, and mills. 8020 also has a large selection of standard fittings for easy mechanical connection of components. This avoids welds or other connection techniques (e.g. adhesives, standard nuts and bolts) which are expensive, permanent, and fully custom.

Beyond a simple frame, the various components of CUBE 2.0 must be shielded from the elements. As such, "underlayment quality" **plywood** is proposed to be used to contain the elements of the system. Precision cuts at the needed joints will also be made for "light proof" conditions. Further, aluminum tape, a totally opaque material common in daylighting studies, is placed over all joints of the system. White paint is used to limit the solar heat gain of the CUBE 2.0, as well as help prevent rotting of the outward facing sheets. Matte finish black paint is used on the inside of the enclosure to ensure a low reflectivity space in which measurement of the exiting luminance distribution, L_v, can occur.

Electrical Power

Five components in the CUBE 2.0 will require electrical power: MacBook Pro, Canon 6D, LabJack T7-Pro, RPiCM, and the Ethernet Router. Two of these, the MacBook Pro and the Canon 6D have self contained battery power. Further, the LabJack T7-Pro, RPiCM, and Ethernet router can be powered off of a standard USB drive (i.e. 5V DC). As such, a USB splitter is used to power the LabJack T7-Pro, RPiCM, and Ethernet router by plugging into the MacBook Pro. To gain more simulation time, the MacBook Pro can be plugged into a standard 120 V AC outlet. Further, the Canon 6D also has a standard 120 V AC power outlet which can be used for extended testing.

4.3.2 Modeling and Simulation of the CUBE 2.0 System

The above design proposal for the CUBE 2.0 system is believed to be able to meet the needs of the HWiL model put forth above in section 4.1. To add confidence to this intuition, as well as calculate several parameters of the system (e.g. physical dimensions) needed for implementation, a modeling/simulation analysis for many of the sub-system components is presented below. Further, these models, much of which exist in the language of set theory, offer concrete definitions to the actual tasks to be completed.

These analyses provided much insight for the CUBE 2.0 system, which ultimately informed the initial CUBE 2.0 design presented above, resulting in the final CUBE 2.0 design discussed below. Using the grouping of the tasks for the CUBE 2.0 originally put forth in the specification section, the various models and analysis used are presented below.

Background to High Dynamic Range Imaging

Both the input illuminance measurement and the exiting luminance measurement will use high dynamic range imaging (HDRI) techniques. With this in mind, this section is dedicated to providing a short background introduction to HDRI techniques using notation that is used throughout the remainder of this thesis.

To begin, the sensing chips used in modern digital cameras come in two types, either a charged-couple device (CCD) or a complementary metal-oxide semiconductor (CMOS). For the purpose of HDRIs in building daylighting systems, however, these sensors are interchangeable, thus are further referred to here as simply sensors. Depending on the quality of the camera, these sensors have dynamic ranges in the order of 8 to 14 bits per pixel. This means, for a given exposure (i.e. a single picture with fixed settings of aperture, shutter speed, ISO, etc.) each pixel in the sensor must map the amount of light striking the sensor, and thus the luminance of the object in the field of view of the camera, to a value between 0 and either $2^8 - 1 = 255$ or $2^{14} - 1 = 16383$ respectively. At the sensor level, without artistic driven post processing (often referred to as RAW), this is well modeled by a linear function. This type of luminance measurement within a scene is known as low dynamic range image (LDRI) photography. By controlling the shutter speed (i.e. the time the shutter is open allowing light to hit the sensor) and aperture size (i.e. the size of the hole through which light travels to strike the sensor when the shutter is open) of the digital camera, the user can move this limited range of the sensor to be sensitive to the magnitudes of luminance in the field of view of the scene to which it is desired to capture an image. This can be conceptualized as changing the slope and intercept of the linear sensor model.

Properly moving the range of the sensor on a camera via adjustments of aperture and shutter speed has seen much work, with the most famous perhaps being the Zone System proposed by the photographer Ansel Adams [2]. While the system was originally devised for film based cameras, the principle remains applicable for digital cameras. Roughly speaking, the system suggests setting the range of the camera to be sensitive to the luminance range of the desired subject of the picture. While this may seem obvious, due to the complexities of subject luminance not being uniform, the desire for proper contrast within the picture, and numerous other technical and aesthetic driven factors, the process is truly part science, part art, and somewhat subjective.

In today's digital cameras, equipped with embedded processors and built in light meters, automatic suggestion for a good range placement for a particular scene is typical. This suggested exposure, defined as a combination of shutter speed and aperture setting[2], is a best guess of how to capture the desired luminance range the camera believes the user to be aiming. To simplify the understanding of varying shutter speed (t, expressed in seconds) and aperture (N, using standard f-number values), the exposure value (EV) system was devised in Germany in the 1950s [122]. Any shutter speed and aperture setting value combination with an equal EV, assuming equal scene luminance between EV settings, equates to an equal amount of light striking the sensor. EV is defined as,

$$EV = log_2\left[\frac{N^2}{\frac{t}{\frac{ISO}{100}}}\right]. \tag{4.1}$$

EV is simply a characterization of shutter speed and aperture settings in which an increase or decrease of one unit, referred to in photography as a "stop", corresponds to a doubling or halving of the total light striking the sensor.

A common method in photography to attain a recommended EV value, for a known subject luminance and a given camera manufacturer, can be calculated by,

$$EV = log_2\left[\frac{L \cdot S}{K}\right], \tag{4.2}$$

where L is some characteristic subject luminance (e.g. total average, spot metering), K is a manufacturer meter calibration constant (e.g. K=12.5 for Canon, Nikon), and S is the ISO speed, usually reported at 100 [67, 36].

Thus, a typical exposure sequence would be: 1) User picks a scene, 2) User activates built-in camera light meter to estimate L, 3) Based on the selected ISO and manufacturer, the cameras computer calculates EV using Eq. 4.2, then using Eq. 4.1 suggests some shutter

[2]Technically this also includes the ISO, however, often an ISO=100 is assumed.

speed, t, and aperture, N, setting, 4) User takes a LDRI using the suggested settings. While elegant, more complicated algorithms are the norm nowadays in commercial digital cameras.

Formalizing LDRI: To better understand the process with real data, the above can be formally expressed as follows. A LDRI is defined as,

$$LDRI\colon = (N, t, \mathcal{P}^{x \times y}), \tag{4.3}$$

where N and t define the EV as calculated using equation 4.1, assuming ISO=100, $\mathcal{P} = [0, 2^{\mathcal{B}}] \subset \mathbb{N}$, define the actual pixel values, all for a $\mathcal{B} = \{8, 10, 12, 14\}$ bit sensor of dimensions $x, y \in \mathbb{N} \backslash \{0\}$. This assumes usage of the same camera to gather all LDRIs. Anchoring this notation to SI units of luminance measurement, a function is defined such that,

$$DR : LDRI \longrightarrow [min, max], \tag{4.4}$$

where LDRI is define as above and $[min, max] = [\mathbb{R}_+, \mathbb{R}_+]$, represents the minimum and maximum luminance values which some LDRI can capture, called here the range of a LDRI. For an arbitrary scene, the resulting LDRI is shown conceptually in Fig. 4.2 in its ability to capture the luminance range of a scene.

Even with an "optimal" or "perfect" exposure as defined with some EV above, there will most likely still be sections of the photograph with luminances respectively above and below the *max* and *min* (i.e. out of the LDRIs range) as reported by the function *DR*. These areas are reported by the sensor as white or black respectively, yet if the range of the sensor were dialed into each respective area's luminance by adjusting the EV, a meaningful value would be reported by the sensor. Knowing the only factor separating the sensor from getting meaningful information from the underexposed black pixels and overexposed white pixels is the shutter and aperture settings, uniquely specified by EV, one might ask, "Why not capture multiple images at varying EV, effectively increasing the range of the sensor, to capture more of the luminance values present in the scene?" This notion precisely is what lies behind high dynamic range image (HDRI) photography [70, 123]. An HDRI is thus a single photograph which has been constructed from a set of varying exposure valued LDRI photographs, such that its range of luminance values is higher than any single LDRI. Note, for luminously dynamic scenes (e.g. those with moving objects) the images must be collected quickly with respect to each other, thus it can be assumed they are exposed to the same luminance distribution. Put another way, a HDRI is the resulting photograph from a set of LDRIs such that when considered together, the respective luminance ranges for which each LDRI is sensitive, span the complete range of the total luminance values of the desired scene.

Positioning of the dynamic range of the individual LDRIs within the luminance distribution of a scene is most typically defined using the same EV expression, i.e. a base two logarithmic system with respect to the light exposure magnitude of the sensor. However, due to their relative spacing to an "optimal" exposure, a new labelling system is used, the so called exposure compensation (EC). The EC values are the same quantity defined by N and t in Equation 4.1, but by definition EC=0 corresponds to the optimal exposure of some scene, not any particular luminance value as does an EV=0. Thus, a set of LDRI, $LDRI^j$, is

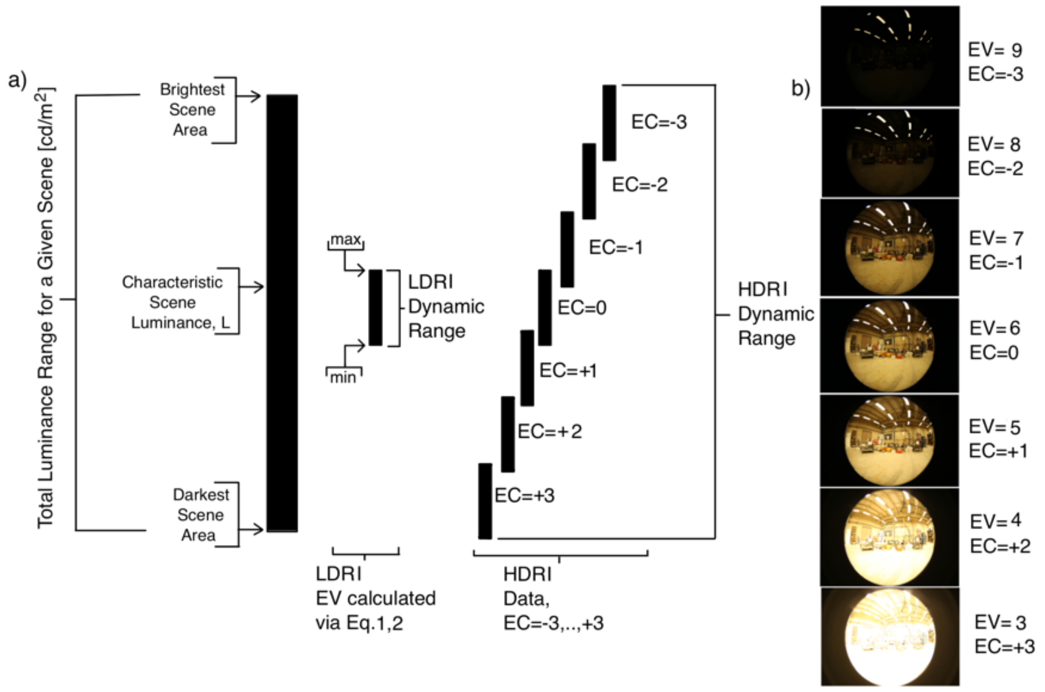


Figure 4.2: a) Conceptual comparison of the span of dynamic range for an LDRI as compared to an HDRI consisting of several LDRIs. The LDRIs of the HDRI are spaced such that both plus and minus exposure compensation (EC) values are included until the entire luminance range has been spanned. b) Real scene LDRIs of an artificially lit laboratory corresponding to the conceptual images presented in (a).

a group of j LDRIs whose total dynamic range span an extended range of luminance values within a scene as compared to a single LDRI. Please note, the EV and EC concepts have been incorrectly presented in the building lighting literature, thus caution is urged to ensure other resources have this distinction correct. An example of an $LDRI^j$ with three images is,

$$LDRI^3 = \{LDRI_{EC=-1}, LDRI_{EC=0}, LDRI_{EC=+1}\}, \tag{4.5}$$

where the respective LDRIs are expected to adjust their EVs (either aperture, N, shutter speed, t, or both N and t) accordingly to best fit the EC to which they correspond. The EC=0 LDRI is calculated as stated above using Equations 4.1 and 4.2 to give the suggested EV value, which is the "optimal" exposure for the scene. The other EC values correspond to being exposed to half the light of optimum exposure (i.e. underexposed and sensitive to the white pixels in the optimal exposure) and being exposed to twice the light of the optimum exposure (i.e. overexposed and sensitive to the black pixels in the optimal exposure). The needed spread of the EC values, and thus number of the LDRIs comprising the set $LDRI^j$, is scene-dependent such that all luminance values within the scene fall within the dynamic

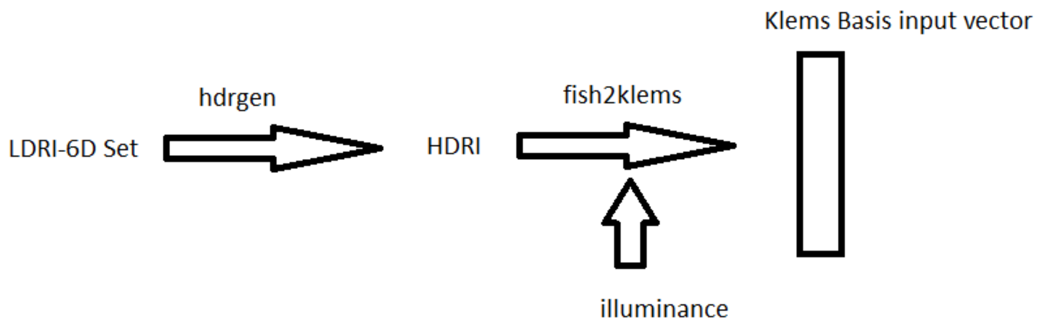


Figure 4.3: Conceptual overview of the input illuminance measurement process.

range of one of the LDRIs. This notion is show conceptually in Figure 4.2. There a set of LDRIs, $LDRI^7$ is presented.

Given the above background in HDRI photography, specific details of how the Canon 6D and RPiCM will capture the set $LDRI^j$ needed to construct their respective HDRI measurements, and the actual construction of that HDRI, are presented individually below.

Measure Input Illuminance Distribution

Creating an E_v Measurement: Given the anticipated luminance distributions which the CUBE 2.0 is exposed to during deployment, high dynamic range imaging (HDRI) is required to capture the entire distribution. As such, a set of low dynamic range images (LDRI), $LDRI^j$ from above, which span the space, must be captured and processed into a HDRI. Further, the images must be corrected for vignetting, and finally anchored to absolute SI units of cadela per meter squared $[\frac{cd}{m^2}]$. The final step then involves averaging the Klems Basis input solid angle values to get a vector of the input illuminance. This process is presented conceptually in Figure 4.3, with the individual steps expanded formally in the sections below.

It should be noted, direct sunlight from the solar disk exhibits a very large luminance, on the order of one billion $[\frac{cd}{m^2}]$. As of the beginning of this thesis work, the process of capturing this luminance magnitude using digital cameras is still an open question [72, 71]. As such, this thesis focuses on diffuse sky luminance, hence no direct component is measured. However, with the use of neutral density filters, as has been demonstrated in the literature [72, 71], it is believed a recalibration of the CUBE 2.0 system could be realized allowing for direct solar components. Limits of time in this thesis project are the reason for not extending into direct solar components.

LDRI Set Capture: As stated above, in order to create a HDRI which can be used to faithfully measure the input luminance distribution for which the CUBE 2.0 system is being exposed, a set of LDRIs must be captured and processed into an HDRI. This set of LDRI images, denoted $LDRI^j$, will need to consist of images whose individual dynamic ranges, when considered in total, span the entire luminance range of the scene, shown conceptually in Figure 4.2.

The method planned for use in the CUBE 2.0 is to vary the exposure compensation (EC) value of a set of subsequent images taken immediately back-to-back in a controlled and consistent manner. By taking the images quickly, it can be assumed they are exposed to the same luminance distribution. As stated above in Equations 4.1 and 4.2, EC is a function of two variables, aperture (N) and shutter speed (t). It is common practice in HDRI measurements to fix the aperture, N, and simply vary the shutter speed, t, in order to adjust the EC value [71, 64]. This method of photography, known as *aperture priority mode*, has several advantages, including: i) fixing the depth of field, ii) more shutter speed options to choose from than apertures, thus better matching of the set EV to the suggested EV, and iii) fixed vignetting effect, discussed further below. Hence, aperture priority is planned for use here to vary EC for the set $LDRI^j$.

The exact EV values which are gathered using the Canon 6D are calculated as follows. First, the EC=0 exposure is calculated using the inbuilt sensor of the Canon 6D. Next, a command is issued to the Canon 6D to adjust the shutter speed to such a value that EC=-5 and take a picture. This is repeated 11 times, at 1 EC increments (known in photography as a "stop") so the LDRI set $LDRI^{11}$ is formed. This LDRI set will then be downloaded onto the MacBook Pro, with the original photographs deleted from the SD card in the camera. The entire EC capturing and downloading process is planned to be completed by a C program implementing the Canon EOS SDK [27]. It should be noted, the computer control allows for a much faster collection speed of $LDRI^{11}$ than is possible "by hand." Further, the process can be automated for long term deployments of the system.

The exact number of 11 photographs, EV$=-5, \cdots, +5$, is planned due to this being the largest computer controllable setting on the Canon 6D. It should be noted, refined EV spanning information (i.e. how far above and below optimal exposure the $LDRI^j$ set will span) can be done for more efficient processing time (i.e. less LDRI photographs), however, 11 is planned for now as it fulfills the requirements for the CUBE 2.0. Further, it ensures mistakes aren't made with respect to the $LDRI^j$ which are gathered, as this is the maximum possible set to gather. It is thus assumed, $DR(LDRI_{EC=-5})_2 > bp$ and $DR(LDRI_{EC=+5})_1 < dp$, where bp and dp are the respective values of highest and lowest luminance within the hemisphere in which the CUBE 2.0 is exposed.

HDRI Formation: Once the set of LDRIs are on the MacBook Pro, the freeware program *hdrgen* will fuse the set $LDRI^{11}$ into an HDRI. This process is known as radiometric self-calibration [40], and can be expressed formally as a mapping,

$$hdrgen : LDRI^{11} \longrightarrow HDRI_{input}, \tag{4.6}$$

where,

$$HDRI_{input} = (N, \mathcal{P}_h^{x \times y}). \tag{4.7}$$

Here, N represents the aperture setting of the camera which is fixed for each of the LDRIs in the set $LDRI^{11}$ at $N = f/4.0$. $\mathcal{P}_h \in \mathbb{R}_+$ is the luminance value associated with each

Table 4.1: Settings used for the Canon 6D.

Feature	**Configuration**
White Balance	Daylight (5200 K)
Image Size (pixels)	720 × 480
Sensitivity	ISO 100
Aperture Priority (Av)	f/4.0
Lens Aberration	OFF
Auto Lighting Optimizer	OFF
Noise Reduction	OFF
Highlight Tone Priority	OFF

individual pixel. Note, this pixel value results from the properly exposed LDRI $\in LDRI^{11}$, hence each pixel in the $HDRI_{input}$ is properly exposed. Again, this assumes the set $LDRI^{11}$ exceeds the luminance range of the scene with respect to it's ability to measure luminance.

It should be stated, however, *hdrgen* is the engine behind the freeware Photosphere [13]. Hence, the planned mapping can be described with the actual file formats used for clarity, rather than mathematical abstractions of this thesis. Here, each LDRI is in fact a "Joint Photographic Experts Group" (JPEG for short) file formated image [68], with the least lossy compression setting used to preserve the measurement data. JPEG is used over RAW as capturing consist images in RAW format is more difficult and offers little extra in terms of measurement information. While this is counter intuitive, as one would think gamma encoding and other post processing done on the camera reduces measurement information, experience shows this is true [60]. For consistency JPEG photographs is captured with the camera settings shown in Table 4.1.

The resulting $HDRI_{input}$ is planned to have $x = 720$ and $y = 480$ pixels, each of which is encoded using the CIE-XYZ color space [69] and the Radiance XYZE floating point encoding [154]. This can be accomplished practically by using the "-c" flag with XYZ. Meaning each pixel has a four byte representation. The interest of the HDRI is luminance, hence the value sought is the "Y" component for the pixel being considered. Luminance can thus be extracted from the $HDRI_{input}$ using the second (Y_b) and fourth byte (E_b) as,

$$\mathcal{P}_h = \frac{Y_b}{255} 2^{(E_b - 128)}. \tag{4.8}$$

This process can be repeated for all pixels ($480 \times 720 = 345600$), in the $HDRI_{input}$ image, resulting in a measurement of the luminance which has been calculated from the set of LDRIs, $LDRI^{11}$.

It should be noted, *hdrgen* also requires a camera response function to form the HDRI. For the CUBE 2.0, a camera specific response function is formed using a scene with large, smooth gradients of radiance in accordance with standard practice. This response will then be reused for each $LDRI^{11}$ set by activating the "-r" flag in *hdrgen*. If no response function is available, *hdrgen* will attempt to create one for the $LDRI^{11}$ set being analyzed. If there is enough variations in the image set, it is successful, yet this mode of operation for *hdrgen* will not be used here.

Vignetting Correction: After the above proposed processing, $HDRI_{input}$ is in proper high dynamic range image form, however, it must still be corrected for a lens artifact called vignetting. *Vignetting*, a form of "lens shading," is the phenomena of a lens to attenuate a given luminance excitation near the edge of the image it forms with respect to the center. Hence, the values of the image must be corrected for this effect in order to report accurate measurement information. The vignetting effect is well understood in optics, and while closed form corrections can be calculated using Fourier optics techniques [54], the daylighting community has preferred to measure these corrections empirically [64].

Vignetting corrections have been shown to be consistent across a single lens model from a given manufacture [31]. Hence, vignetting corrections functions can be looked up in the literature for a specific lens being used, with the open source LensFun project [90] being a growing source. For completeness, the vignetting correction function is measured in the calibration process of the CUBE 2.0, with the results compared to Inanici [64] from the literature presented in Chapter 5 as part of the calibration. Moving forward, the vignetting function is referred to notationally as $v(\theta)$ with θ referring to the incident angle with respect to the optical axis of the lens.

Hence, for each pixel (i, j), $i = \{1, \cdots, x\}$ and $j = \{1, \cdots, y\}$, the value of which is denoted $HDRI_{input}(i, j)$, the new vignetting corrected value is,

$$HDRI_{input}(i,j) = \begin{cases} HDRI_{input}(i,j)/v(DC(i,j)_1) & for \quad DC(i,j)_1 > 0 \\ 0 & else \end{cases} \tag{4.9}$$

Where the function DC, is defined below as Equation 4.10 which allows for a transformation from (i, j) to (θ, ϕ), making $DC(i, j)_1 = \theta_e$ of the pixel (i, j) being considered. Note here, the vignetting correction function is assumed to be that for an aperture of f/4.0.

Absolute SI Unit Anchoring: At this stage, the $HDRI_{input}$ is formed from the set of LDRIs, $LDRI^{11}$, and corrected for vignetting, meaning the relative magnitudes of the individual pixel's luminance values are correct. However, these values may in fact all be off by a scaler multiple from true SI units of $[\frac{cd}{m^2}]$. As a result, a scaling factor must be calculated and applied to each pixel within the HDRI. Calculating this scaling factor can be done in one of two manners, depicted graphically in Figure 4.4: i) spot-luminance calibration, or ii) vertical-illuminance calibration.

Spot-Luminance Calibration: The first and most common of which, spot-luminance calibration, involves using a hand-held luminance meter (e.g. Konica Minolta LS-160, Gossen Mavo-Spot 2) to measure a specific point of luminance in the scene captured by the HDRI concurrently with the LDRI set capture. This specific point should be a relatively uniform luminance emitter (e.g. a grey surface or ideally a well-defined target such as the Macbeth ColorChecker). Here, the luminance measurement value, $\mathcal{L}_{i,j}$, is associated with some subset of pixels in the $HDRI_{input}$, $l_{i,j} \subset \mathcal{P}_h^{x \times y}$, such that the average value of $l_{i,j}$ should equal

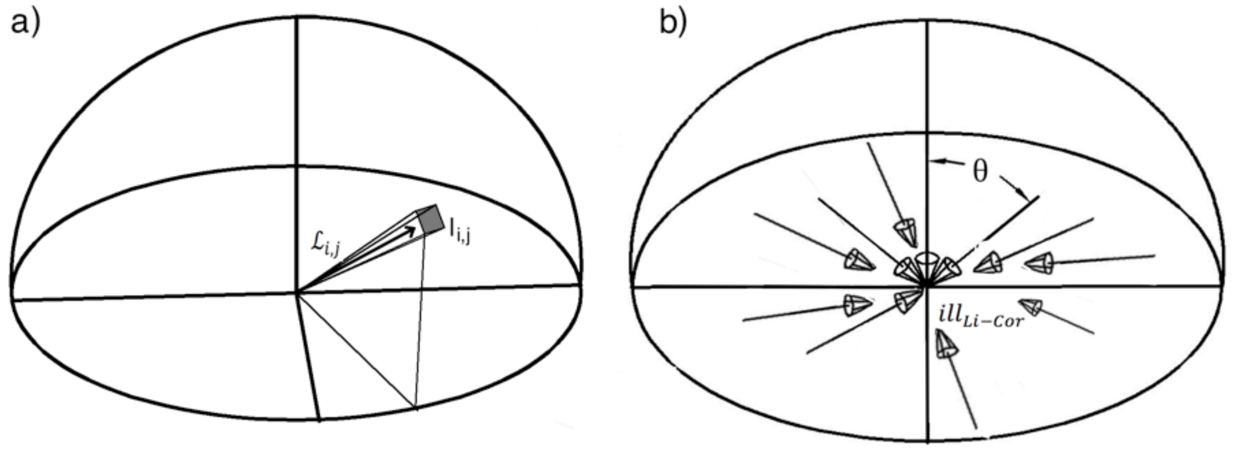


Figure 4.4: Conceptual diagram of HDRI calibration techniques: a) spot-luminance, b) vertical-illuminance

$\mathcal{L}_{i,j}$, see Figure 4.4. Most likely, this will not occur. Thus, a correction factor is calculated, $K_L = \mathcal{L}_{i,j}/mean(l_{i,j})$, and due to the relative proportions of each pixel being correct, the correction, K_L, is applied through multiplication to all pixels in $HDRI_{input}$, resulting in an $HDRI_{input}$ whose values are a proper measurement of the luminance conditions of the originally captured scene [64]. This method, however, requires not only a diffusely reflecting uniform target within the image, it also requires the pixel locations of that target. Having a fixed target with respect to the CUBE 2.0 input measuring Canon 6D would be difficult, thus attention is turned to the second method for calculating the scaling correction factor, vertical-illuminance calibration.

Vertical-Illuminance Calibration: In vertical-illuminance calibration, an illuminance measurement coplanar and in close proximity laterally with the fisheye lens, ill_{LC}, can be used to calculate the correction factor, K_I. This notion comes from the fact that using an $HDRI_{input}$, the function $L(\theta, \phi)$, which defines the luminance distribution being experienced by the fisheye lens, can be defined through the function,

$$DC : [i] \times [j] \longrightarrow [\theta] \times [\phi], \tag{4.10}$$

which maps (i, j) pairs (i.e. pixels) to (θ, ϕ) pairs ($\theta \in [0, \frac{\pi}{2}], \phi \in [0, 2\pi]; i \in \{1, \ldots, x\}; j \in \{1, \ldots, y\}$). Note, if the pixel is outside the circular image, the function returns $(-1, -1)$ indicating this is part of the mask of the image and not contributing to illuminance. The exact formation of the function DC will depend on the specific type of fisheye lens used (e.g. equidistant, equi-solid angle, orthographic, or stereographic projection), each of which have a different projection formula [20, 31].

With the luminance distribution known, it can be integrated to provide illuminance as follows,

$$ill_{HDRI} = \int_0^{2\pi} \int_0^{\frac{\pi}{2}} L(\theta, \phi) cos(\theta) sin(\theta) d\theta d\phi. \tag{4.11}$$

Or with respect to the picture dimensions,

$$ill_{HDRI} = \sum_{pixels} HDRI_{input}(i, j) * cos(DC(i, j)_1) * \omega(i, j), \tag{4.12}$$

where $DC(i,j)_1$ is the calculated θ_e, $HDRI_{input}(i,j)$ is the actual luminance value as originally calculated in Equation 4.8 and vignetting corrected in Equation 4.9, and $\omega(i,j)$ is the solid angle, each of which are associated with the pixel (i,j), which are defined explicitly below for the fisheye lens used in this work.

The correction factor to K_I can then be calculated and applied to the $HDRI_{input}$ as,

$$K_I = ill_{LC}/ill_{HDRI}. \tag{4.13}$$

In theory, both the spot-luminance and vertical-illuminance calibration methods are equivalent (i.e. $K_L = K_I$) and can thus be used interchangeably. For reasons of not using a target with the CUBE 2.0, K_I is planned for use here. Thus, each value in the $HDRI_{input}$ is scaled by K_I, meaning the pixels are fully corrected and calibrated and can be used for scientific grade measurements.

Details of Equation 4.12: Each fisheye lens type (e.g. equidistant, equi-solid angle, orthographic, or stereographic projection) and the chosen image pixel dimensions (i.e. x, y) will result in a different formation for $DC(i,j)$, and $\omega(i,j)$. For this thesis work, the derivation of these is now shown for arbitrary x and y and an equi-solid angle fisheye lens.

The Sigma 8mm F3.5 EX DG fisheye lens is an equi-solid angle projection lens, thus its projection formula is,

$$r = 2ftan(\theta/2). \tag{4.14}$$

Here, θ is the incident angle with respect to the lens central axis, f is the focal length of the lens, and r is the distance on the image plane measured from the optical axis. For this thesis work, an object was located at $\theta = 90°$ with respect to the fisheye lens and image was taken. This object was then identified in the image, allowing for a determination of r_{90} in pixels from the optical axis. r_{90} can be thought of as the radius in pixels from the center of the image corresponding to the perpendicular direction from the optical axis at the lens surface. Meaning any pixel with an effective radius $r_e > r_{90}$, defined below, is outside the hemisphere of input and can be disregarded. With this, Equation 4.14 was used to calculate a focal length in pixels, f_{pixel}, for the fisheye lens. The details of these measurements and calculation can be found in chapter 5. Now, using basic trigonometry the function DC can be stated explicitly, with θ_e its first output and ϕ_e its second output.

With the center of the image known, (i_c, j_c), and using the bottom left corner of the image as the origin[3], each pixel in the image is given a new effective coordinate, $(i_e = i - i_c, j_e =$

[3]The top left corner is often used as the origin, however, it is hard to conceptualize this origin as it is counter to the way the vast majority of people learn mathematics. Both are equivalent, however, must be implemented in code correctly, a task easier said than done.

$j - j_c$). With these new coordinates, each pixel is given an effective image plane distance, $r_e = \sqrt{i_e^2 + j_e^2}$, which by inverting Equation 4.14, gives the θ_e value for each pixel,

$$\theta_e = \begin{cases} 2arctan\left(\frac{r_e}{2f_{pixel}}\right) & for \quad r_e \leq r_{90} \\ -1 & else \end{cases} . \tag{4.15}$$

The ϕ_e value is calculated simply using the cartesian to polar coordinate transformation, where,

$$\phi_p = \begin{cases} arctan\left(\frac{i_e}{j_e}\right) & for \quad r_e \leq r_{90} \\ -1 & else \end{cases} \tag{4.16}$$

and shifting to the proper quadrant,

$$\phi_e = \begin{cases} if(\phi_p \neq -1)\{ & \\ \quad if(i_e > 0 \ \& \ j_e > 0) & \{\phi_p\} \\ \quad elseif(i_e < 0 \ \& \ j_e > 0) & \{\phi_p + 180\} \\ \quad elseif(i_e < 0 \ \& \ j_e < 0) & \{\phi_p + 180\} \\ \quad else & \{\phi_p + 360\} \\ \} & \\ else\{ & \\ \quad -1 & \\ \} & \end{cases} \tag{4.17}$$

The $\omega(i,j)$ factor can too be calculated via the *projection formula* and an understanding of solid angles. As background, a solid angle, Ω, is measured in steradians, $[sr]$, which are analogous to the radians used in planer angles. A solid angle is characterized by the area on a sphere which it subtends, A_s, divided by the square of the radius of that sphere, r_s, such that $\Omega = \frac{A_s}{r_s^2}$. Hence, if the area of a certain patch on a unit hemisphere being captured by a equi-solid angle fisheye lens is calculated, then that area divided by the number of pixels which represent that area, the solid angle for which those pixels represent can be calculated. Assuming a small enough patch is utilized, this can be repeated and the solid angle values for all the pixels can be calculated.

Due to radial symmetry of the fisheye lens, pixels here are assigned solid angle values in batches based on their θ_e. Beginning with micro-theta band one, $\theta_{up} = 0° \leq \theta_e < 1° = \theta_{down}$, the area on the unit sphere of this micro-theta band is calculated as $A_{\mu\theta} = 2\pi(cos(\theta_{up}) - cos(\theta_{down}))$. This area is converted to a solid angle by dividing by the radius squared, meaning $\Omega_{\mu\theta} = \frac{A_{\mu\theta}}{r^2} = \frac{A_{\mu\theta}}{1^2} = A_{\mu\theta}$. Now, the pixels in the image corresponding to this area are grouped,

$$pix_{\mu\theta} = \{(i,j) \mid \theta_{up} \leq DC(i,j)_1 < \theta_{down} \ \}, \tag{4.18}$$

then counted,

$$p_{\mu\theta} = |pix_{\mu\theta}|, \tag{4.19}$$

Table 4.2: Definition of the Klems Basis Patches.

Theta Band	Patches	Theta Range (°)	Solid Angle (steradians)
1	1	0-5	0.0239
2	2-9	5-15	0.0238
3	10-25	15-25	0.0234
4	26-45	25-35	0.0274
5	46-69	35-45	0.0293
6	70-93	45-55	0.0350
7	94-117	55-65	0.0395
8	118-133	65-75	0.0643
9	134-145	75-90	0.1355

where $|\cdot|$ denotes the cardinality operator. Finally, the solid angle for this group of pixels is calculated as

$$\omega(i,j) = \frac{\Omega_{\mu\theta}}{p_{\mu\theta}},\ \forall\ (i,j) \in pix_{\mu\theta}. \tag{4.20}$$

This process is then repeated for the next micro-theta band two, $\theta_{up} = 1° \leq \theta_e < 2° = \theta_{down}$, until all micro-theta bands have been covered, 0° to 90°. It should also be noted, there are other methods to calculate $\omega(i,j)$, yet this method is shown in chapter 5 to be accurate to 14 digits with respect to its theoretically derived value for $x = 720$ and $y = 480$.

Details of ill_{LC}: For completeness, it should be stated the illuminance measurement, ill_{LC}, used in the calibration of the $HDRI_{input}$ is measured by the LabJack T7-Pro via a Li-Cor LI-210. This sensor is well modeled as a linear system, hence,

$$ill_{LC} = \mathcal{C} \cdot (V_{LC} - V_{dv}), \tag{4.21}$$

where $\mathcal{C}$ is the manufacturer provided calibration constant, V_{LC} is the voltage across a 604 ohm precision resistor used to convert the current to a voltage, and V_{dv} is the dark voltage associated with a measurement of no illuminance. Both V_{LC} and V_{dv} is queried via the LabJack provided library and $\mathcal{C}$ is provided by the manufacturer. Hence, the system can get the vertical illuminance measurement, ill_{LC}, totally programmatically.

Fisheye Hemisphere View to Klems Basis Input Vector: The final step which is needed for the CUBE 2.0 system with respect to the input illuminance measurement is the aggregation of the hence forth constructed, finely discretized input illuminance measurement provided by $HDRI_{input}$, into the Klems Basis input. The Klems Basis is a division of the total incoming hemisphere into 145 discrete "patches," each of which have roughly the same solid angle. Table 4.2 defines the Klems Basis patches using the standard θ and ϕ coordinate system. Note, each theta band is evenly distributed with patches, hence the ϕ divisions are each evenly spread out within the band. For more detail see the orignal papers by Klems [80, 81], or the related paper by McNeil *et al.* [101].

Thus, for each Klems Basis Patch, $n = \{1, \cdots, 145\}$, a solid angle weighted averaging of the cosine weighted luminance values from the $HDRI_{input}$ is made. Put formally,

$$k_n = \{(i,j) \mid DC(i,j) \in K_n\}, \tag{4.22}$$

is the set of pixels in Klems Basis Patch n, where $K_n = ([\theta_{min}, \theta_{max}], [\phi_{min}, \phi_{max}])$ is the θ and ϕ limits associated with Klems Basis Patch n. Meaning,

$$E_v^n = \frac{\sum_{k_n} HDRI_{input}(i,j) cos(DC(i,j)_1)\omega(i,j)}{\sum_{k_n} cos(DC(i,j)_1)\omega(i,j)}, \tag{4.23}$$

is the input illuminance for patch n. This calculation is repeated for all patches, ending with the creation of the total $E_v \in \mathbb{R}^{145\times 1}$.

The above process, resulting in E_v is repeated for each round of a CUBE 2.0 simulation, where E_v^t corresponds to the input illuminance measurement for round t.

Canon 6D and Fisheye Lens Aperture Alignment: The careful reader at this point may inquire as to how the input measurement is made by the Canon 6D and Sigma Fisheye lens as it cannot be coincident with the *CFS test aperture* which will hold the CFS undertest. This is indeed correct, as the input illuminance measuring Canon 6D and fisheye lens would block the CFS test aperture. As such, the Canon 6D and fisheye lens is offset vertically from the CFS test aperture at a height of 16 inches. 16 inches comes from a 12 inch radius of the acrylic hemisphere, plus 80/20 frame, plus plywood enclosure, plus construction tolerance, and finally half the fisheye lens diameter. However, given this 16 inch offset the luminance distribution seen by the fisheye lens, $L_{fish}(\theta, \phi)$, and CFS test aperture, $L_{CFS}(\theta, \phi)$, are now different. Further, the luminance distribution seen by the LI-210 for SI unit anchoring, $L_{LI-210}(\theta, \phi)$, is different still. This difference, that is $L_{fish}(\theta, \phi) \neq L_{CFS}(\theta, \phi) \neq L_{LI-210}(\theta, \phi)$, is known as parallax error.

It is believed using similar reasoning to the "**5× Rule**," the fisheye measurement can be considered coincident with the CFS test aperture if the source of light excitation is greater than five times the 16 inch figure. $5 \times 16 = 80$ inches, 80 inches $\times \frac{1}{12}\frac{\text{foot}}{\text{inch}} = 6.\overline{6}$ feet. In practical terms regarding testing, this means the light sources (i.e. any object reflecting light onto the CFS test aperture) must be a distance of 6 feet 8 inches or more away.

To investigate this issue definitively beyond the **5× Rule** intuition would require a separate analysis for every location the CUBE 2.0 system could be deployed. Given this is not possible, as there exist infinite possible scenarios, Radiance is used to analyze a generic testing area consisting of a simple plane surface on which the CUBE 2.0 system is deployed. This analysis consisted of calculating the vertical illuminance falling onto the CFS test aperture, E_T^a, fisheye lens, E_T^b, and the LI-210 illuminance meter, E_T^c. Beyond the simple vertical illuminance, denoted by the "T" for total, the directionally varying illuminance distributions with respect to the Klems Basis was also calculated. Meaning, each total vertical illuminance value is divided into 145 components which correspond to the illuminance contributed by the respective Klems Basis solid angles. The Klems Basis divided illuminance measures are denoted $E_K^a, E_K^b, E_K^c \in \mathbb{R}^{145\times 1}$ respectively for the CFS test aperture, fisheye lens, and

LI-210. Further still, as inspired by Mardaljevic [99], the illuminance distributions were calculated with clear sky excitation using 13 representative sun locations. These $r = 1, \ldots, 13$ locations are designed to be representative of the total solar cycle with respect to Berkeley, California, making the final set of measurements, $\{(E_K^{a,r}, E_K^{b,r}, E_K^{c,r}) \mid r = 1, \ldots, 13\}$.

In addition to the mathematical model and simulation exercise using Radiance shown above, the actual Canon 6D and fisheye lens were used to gather input illuminance measurements as well. These measurements consisted of two $HDRI_{inputs}$ that were vertically displaced from each other by 14 inches and were analyzed similarly with respect to input illuminance errors with respect to Klems Basis Patch averaging. While not 16 inches as in the proposed CUBE 2.0 design due to tripod limitations, the analysis offers reasonable insight into the proposed design performance. In this thesis, two sets of input measurements are included for review.The total results of the input analysis, both in simulation as well as physical measurement form, can be seen in Appendix F.

In examining both the simulation and experimental analysis with respect to Canon 6D fisheye lens, LI-210, and CFS test aperture alignment for the Klems Basis divided input illuminance distribution, it is concluded the large majority of klems patches are below 5% difference. This means, $L(\theta,\phi)_{cfs} = L(\theta,\phi)_{fish} = L(\theta,\phi)_{LI-210}$, is true for errors below 5% for the vast majority of Klems Basis Patch cases. With the reported error on the datasheet for the LI-210 illuminance meter at ±5% in mind, this is deemed acceptable for the CUBE 2.0. While some patches demonstrate errors above this level, ranging into 20% (and even a small few above this), the general behavior is determined acceptable for the CUBE 2.0 setup. Further, as noted in the appendix, the qualitative behavior is very similar between the two inputs, hence adds confidence to at least the relative behavior of the CFS systems being studied. Considering further the reasoning put forth in the design stage (i.e. no viable other options even exist to measure the input illuminance distribution) the 16 inch vertical displacement proposal is adopted whole heartedly for the CUBE 2.0 system.

Measure Exiting Luminance Distribution

Hemisphere Radius (5× Rule): In photometry, measurement of the luminance distribution from a light source falls into two categories: *Far Field* and *Near Field*. Far Field means the light source can be approximated as a point source and still be modeled correctly within some desired error band (e.g. 1% or 0.1%). Thus, $L(\theta, \phi)$ is the desired function in a Far Field Photometry measurement paradigm. Alternatively, Near Field means the light source in question cannot be modeled as a point source and thus must be modeled to have actual two or three dimensional geometry. As such, $L(x, y, \theta, \phi)$, where x and y are locations on the light source, is the desired function in a Near Field Photometry measurement paradigm.

With this in mind, the division between Near and Far Field Photometry is not constant, but rather varies between application. In the field of lighting and daylighting in buildings, Far Field Photometry is considered valid for an error percentage of 1% [16]. Modeling a finite dimensional luminous object as a point source is valid if the luminance distribution

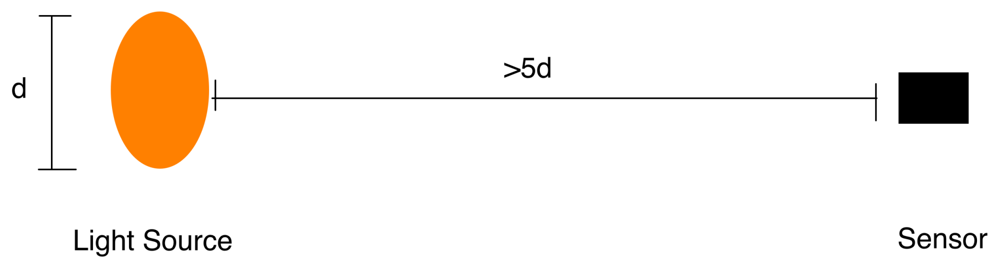


Figure 4.5: Diagram showing the conceptual principle of the 5× rule for far and near field photometry.

measurements out of the light source is taken at a distance 5× or greater the largest dimension of that light source [85, 140, 114], see Figure 4.5. It is also illustrated in the standard defining the definition of goniophotometers, the devices used to gather luminance distribution data [94]. This convention in the building lighting community is referred to as the "**5× Rule**."

With the "**5× Rule**" in mind, measuring the exiting luminance distribution, L, from the inside surface of a complex fenestration system (CFS) undertest is now considered. Due to the mathematics of Far Field Photometry being much simpler than Near Field, Far Field Photometry is the desired approach whenever possible, and is used here. Recalling the proposed design of optical fibers acting as "samplers" of the luminance distribution, the position of the optical fiber ends must be greater than $5 \times d$ where d is the largest dimension of the CFS section under test (i.e. the light source being measured), if a point source approximation is to be valid. Seeing the optical fiber ends are held in place by an acrylic hemisphere, the radius of that hemisphere, $r_{hemisphere}$, must be great than or equal to five times the CFS sample diameter (i.e. $r_{hemisphere} >= 5 \times CFS_{diamter}$), see Figure 4.10.

Considering the above Far Field Photometry consideration, and examining the current literature with respect to goniophotometers [6], a $CFS_{diamter} = 1$ inch was settled upon. This 1 inch whole is subsequently referred to as the *CFS test aperture*. Complementarily, $r_{hemisphere} = 12$ inches was also settled upon, easily satisfying the "**5× Rule**," meaning point source approximation errors are well below 1%. When considered with respect to the portability requirements of the entire CUBE 2.0 system, a two foot wide measurement space gives additional room for the CUBE 2.0 frame and enclosure to be extended beyond the 24 inch total hemisphere width, yet still allows the CUBE 2.0 to fit through standard doorways of approximately 30 inches. This is a critical advantage for portability.

Optical Fiber Spectral Attenuation: Optical fibers are used in the proposed CUBE 2.0 design to "bend" the three dimensional measurement of the exiting luminance into a two dimensional measurement. Because the light is transfered through the optical fibers, the proposed of which have a core consisting of Poly(methyl methacrylate) (PMMA), one

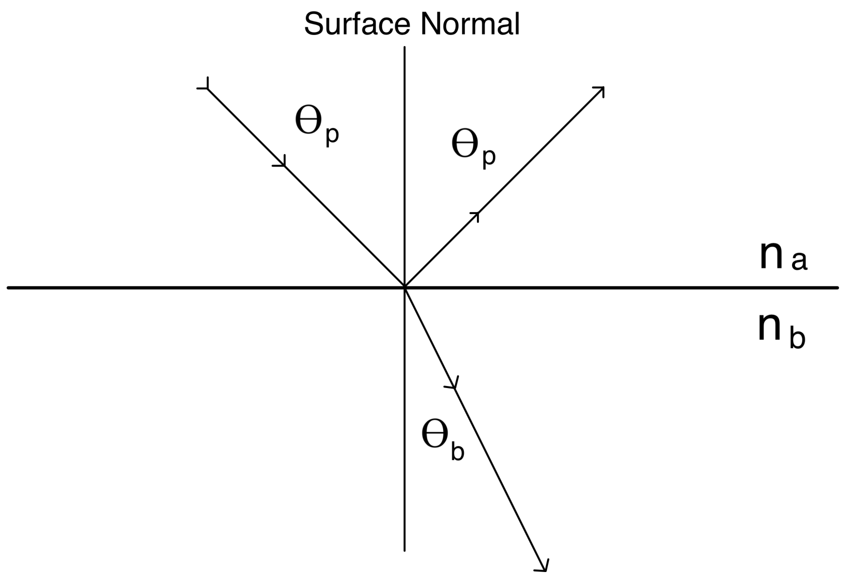


Figure 4.6: Diagram showing incident angle for which polarization of the reflected light will occur, θ_p.

must ensure it is not modified spectrally in a non-uniform manner. This is in fact the case, as PMMA optical fibers transmit all visible wavelengths, those being measured, with equal magnitude of a little over 90 % [75]. Hence, the relative magnitude of the spectral content of the light leaving the optical fiber is the same entering the optical fiber.

Optical Fiber Polarization: Reflection at the surface of an optical boundary between two materials of varying Index of Refraction can cause partial to full polarization of the reflected light according to Brewster's Law. Brewster's Law predicts the incident angle with respect to normal of the surface at which total polarization of the reflected component of the incident ray occurs. Examining Figure 4.6 one can see how θ_p relates to the respective Index of Fractions (n_a, n_b), which are related by Brawster's Law as follow: $\theta_p = \arctan(\frac{n_b}{n_a})$ [48]. For the proposed optical fibers made of PMMA, $n_b = 1.49$, with that of air, $n_a \cong 1.0$, meaning $\theta_p = 54.5°$ Seeing each optical fiber end is oriented with the normal of its surface pointed directly at the CFS, which is modeled as a point source, the $\theta_{incident}$ is very small compared to θ_p, meaning polarization is of little effect and can thus be ignored. Further, the similarity of the orientation of each optical fiber end to the point source will cause equal polarization among the optical fibers, allowing for a valid measurement as each Klems Basis direction has been modified in a similar manner.

Optical Fiber Total Internal Reflection: Within the optical fiber core it is assumed total internal reflection will occur as the exiting luminance distribution is "bent" by the optical fibers. Given the incident rays of interest is entering the optical fiber near 0° (i.e. in parallel with the optical fiber longitudinal axis), only bending of the optical fiber would

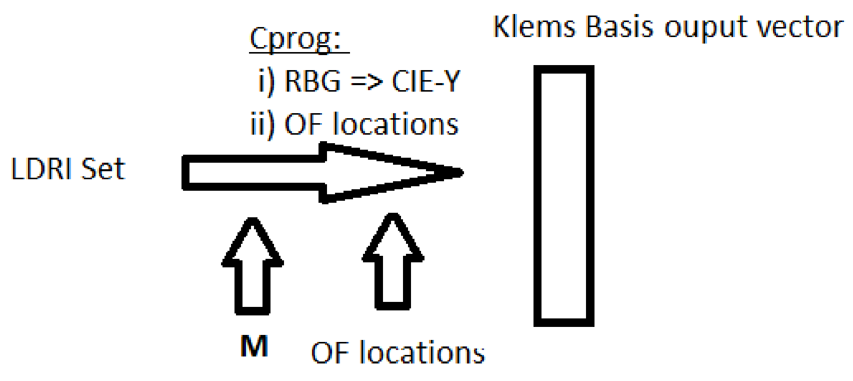


Figure 4.7: Conceptual overview of the input illuminance measurement process.

cause non-transmission of the total sampled values. The incident angle of these rays with the optical fiber core boundary once in the optical fiber must then be less than the critical angle, θ_{crit}, which is defined as,

$$\theta_{crit} = arcsin(\frac{n_{cladding}}{n_{core}}) = arcsin(\frac{1.3}{1.49}) = 60°. \tag{4.24}$$

As such, the optical fibers must be kept to bending radiuses such that they don't induce incident angles within their cores for the traveling rays greater than θ_{crit}, [48]. This is attainable in the CUBE 2.0 system and is noted for how optical fiber lengths are cut and arranged.

Creating an L_v Measurement: While the luminance levels of the "measurement ends" of the optical fibers within the measurement cone are far smaller than those of the CFS test aperture, their dynamic range is also anticipated to require high dynamic range imaging (HDRI) for full capture. As such, a LDRI set will also be captured of the optical fiber ends within the measurement cone. Each optical fiber end will then be processed in a HDRI manner in parallel, and directly transformed into an SI anchored luminance value. The process is conceptually very similar to the Canon 6D process presented above, however, differences arise due to the containment of the measurement cone, fixed orientation of the optical fiber ends, and the calibration process employed.

Conceptually presented in Figure 4.7, the following sections expand the proposed process with mathematical clarity. First, the gathering of the LDRI set is presented. Then processing that set with respect to color space, or changing the actual pixel values from camera specific ones to the CIE-XYZ space, is presented. Next, the parsing of the individual optical fiber ends, and thus their respective Klems Basis output patches, in terms of an HDRI formation from the set of LDRIs is covered. Finally, the anchoring of the unitless "bit values" is made with respect to known luminance excitation.

LDRI Set Capture: The exposure compensation (EC) method presented above, and used by the Canon 6D, is a straight forward approach for gathering a set of low dynamic range images (LDRIs) for processing into a high dynamic range image (HDRI). To set the EC values, the inbuilt camera meter is used to correctly gage the exposures of the scene. For the input illuminance measurement, this is a manageable task, as taking *landscape like* photographs is a common application of digital single-lens reflex (DSLR) camera like the Canon 6D. However, based on experience using the RPiCM, the conditions within the measurement cone are anticipated to make using the RPiCM's inbuilt meter problematic. This stems from the fact that luminance gradients are very large, ranging from near 0 to several thousand $[\frac{cd}{m^2}]$ at the optical fiber "measurement ends" in a single field of view, and thus, occurring in a distributed "hilly" type manner. These large differences, spatially distributed in the field of view, typically make consistent EC calculation difficult for inbuilt meters as the pixel locations chosen to calculate luminance can be very different than the ones which are desired.

With this in mind, a different method of gathering the LDRI set to create a HDRI is used. Rather than starting from an arbitrary "optimal" exposure and expanding downward and upward to capture the luminance distribution, the RPiCM will start with the longest exposure (i.e. measuring the darkest regions) and proceed upward to the shortest exposure (i.e. measuring the brightest regions). This process will create a set of LDRIs captured by the RPiCM within the measurement cone, and is referred to here as,

$$LDRI_{RPi}^{j} = \{LDRI_{RPi,1}, \ldots, LDRI_{RPi,j}\}. \tag{4.25}$$

With,

$$LDRI_{RPi,e} = (N, t, \mathcal{P}_c^{x \times y}), \tag{4.26}$$

where $\mathcal{P}_c = (\mathcal{P}, \mathcal{P}, \mathcal{P})$ is a three channeled pixel for "red", "green," and "blue," $N = f/2.4$ is a fixed aperture, and t is the shutter speed in microseconds $[\mu s]$. As with the Canon 6D, the fixed aperture of the RPiCM allows for aperture priority mode selection of the EC values, hence, only shutter speed, t, is varied between the $LDRI_{RPi,e}$.

The notion of multiple channels now only slightly complicates the idea of exposure. One can simply think of a new function DR_3 consisting of three channels,

$$\begin{aligned} DR_3 : LDRI_{RPi} \longrightarrow (&max(DR(\mathcal{P}_c^1)_1, DR(\mathcal{P}_c^2)_1, DR(\mathcal{P}_c^3)_1), \\ &min(DR(\mathcal{P}_c^1)_2, DR(\mathcal{P}_c^2)_2, DR(\mathcal{P}_c^3)_2)), \end{aligned}$$

stated simply in words, "the channel with the largest *min* governs the bottom of the exposure range, and the channel with the smallest *max* governs the top of the exposure range."

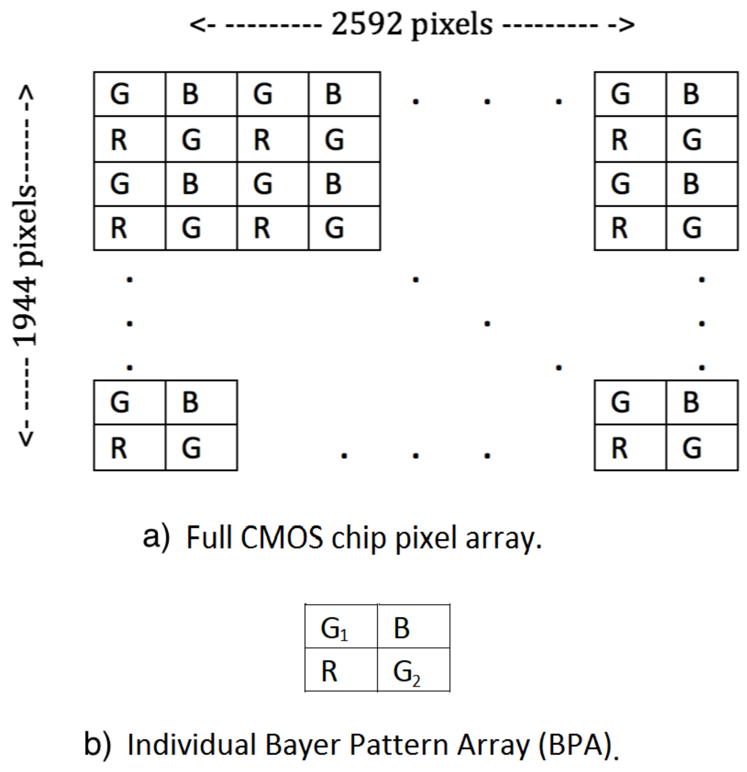
The processes of determining the actual EC values for use by the RPiCM in the measurement cone is proposed to be started by assigning $j = 6$ with nominal shutter speeds from 10^6 $[\mu s]$ to 10 $[\mu s]$ decreasing by one order of magnitude between photographs. This scheme is chosen as the true RAW pixel values is used from the sensor, hence they will exhibit very linear behavior. The RPiCM sensor is a 10-bit chip (OmniVision OV5647), $\mathbb{B} = 10$, and thus $\mathcal{P} = [0, 1023] \subset \mathbb{N}$. Meaning roughly as each $LDRI_{RPi}$ saturates at a sensor returned value of approximately 1000, a reduction in the shutter speed of $10\times$ will bring that same luminance value down to a sensor returned value of approximately 100. Meaning, the dynamic range of each $LDRI_{RPi}$ is well utilized, yet allow for some overlap to prevent saturation or noise floor interaction. The absolute placement of these overlapping $LDRI_{RPi}$ are believed to be well grounded to low luminance values as a shutter speed of $t = 10^6$ $[\mu s]$, or one second, is a long exposure being able to measure luminance values in the single digits of $[\frac{cd}{m^2}]$. Put more simply, the longest shutter speed will capture the lowest luminance conditions expected, and the rest of the $LDRI_{RPi}s$ are spaced above this in a very efficient manner with sufficient but not wasteful placement of the dynamic range. The actual exposure values eventually used are derived empirically as part of the calibration process and are presented in chapter 5.

Color Space Transformation: In typical usage, the final product of modern day digital cameras is a three channel, "red", "green," and "blue" value for each physical pixel on the chip. These values, however, are the result of complicated *demosaicing algorithms*, as in reality each individual pixel on a sensing chip can only record a single color channel, either "red," "green," or "blue." This stems from the filters applied to the pixels which allow them to sense limited wavelength bands. The sensor in the RPiCM is an OmniVision OV5647 complementary metal-oxide semiconductor (CMOS) chip. It has $1944 \times 2592 = 5\,038\,848$ base pixels, over which a combination of filters are placed in what is known as a Bayer Pattern. As can be seen in Figure 4.8, the total Bayer Pattern consists of groups of four base pixels, arranged two by two, which are covered with green, blue, red, and green filters. These groups of four pixels are here called Bayer Pattern Arrays (BPA) and consist of G_1, B, R, and G_2 pixels, see Figure 4.8. In this application, it is proposed that each BPA be considered its own pixel, complete with four samplers. With this assumption, the resolution of the RPiCM will drop by a factor of two, hence the $LDRI_{RPi}$ described above will have $x = 648$, $y = 486$, and $\mathcal{P}_c = (\mathcal{P}, \mathcal{P}, \mathcal{P}) = (R, \lfloor \frac{G_1+G_2}{2} \rfloor, B)$ with each R, G_1, G_2, B $\in [0, 1023] \subset \mathbb{N}$.

Regardless of the shutter speed or aperture setting, the actual pixel values in each of the $\mathcal{P}_c$ channels is specific to the device on which they are caught. To signify this, the three channels in $\mathcal{P}_c$ are referred to with quotes, "red," "green," and "blue" as there is no objective standard to which these colors refer. These device specific colors, that is the complete set of measurable colors, are hereby referred to as the RGB_{RPi} color space [153]. It should be noted, this space varies even from device to device within the same manufacturer. However, the measurement of light and reproduction of color is a very import task and has hence spurred the development of so called standard color spaces. In a standard color space the 3-tuple values which characterize a color are in fact an objective specification of the color which can be used in other applications.

Perhaps the most famous standard color space is the International Commission on Illumination (CIE) XYZ space [134, 69]. This space was developed around 1931 as an improvement over the CIE RGB space, with the major change being the use of imaginary primaries versus the original real primaries. The CIE RGB space is the product of experimental work done by Wright [162, 163] and Guild [57] and is thus based on empirical observation of human eyes. A major advantage of the CIE-XYZ space is the Y channel has been cleverly designed to in fact be luminance measured in $[\frac{cd}{m^2}]$.

With the fact that CIE-Y is luminance, which is sought in the measurement of the optical



<- --------- 2592 pixels --------- ->
<- ------ 1944 pixels------- ->
G B G B G B
R G R G R G
G B G B G B
R G R G R G
G B G B
R G R G
a) Full CMOS chip pixel array.
G1 B
R G2
b) Individual Bayer Pattern Array (BPA).

Figure 4.8: a) Schematic of the RPiCM Bayer Pattern Filter. b) An individual BPA, considered a pixel in this thesis.

fiber "measurement ends" inside the measurement cone, a mapping of the RGB_{RPi} 3-tuples to CIE-XYZ 3-tuples is considered. In fact, this mapping does exit and takes the form of a linear transformation encoded in a three by three matrix [100]. Put formally, consider some $\mathcal{P}_c \in RGB_{RPi}$ the equivalent representation of this measurement is $\mathcal{P}_{XYZ} = \mathbb{M}\mathcal{P}_c$.

The construction of this mapping, $\mathbb{M}$, is known as spectral characterization. At this stage it is assumed to exist and is hence used to describe the needed transformations which are proposed for the exiting luminance measurement. Looking ahead to the calibration stage, a least squares regression methodology is used [153] with the data presented in chapter 5 for the actual calculation of $\mathbb{M}$.

Individual Optical Fiber HDRI Formation: Given the above stated formalisms, the set of $LDRI^6_{RPi}$ is collected, and attention is turned to the transformation of them into a HDRI. Unlike the input measurement, however, only a small portion of the field of the view (i.e. total pixels) needs to be analyzed and formed into an HDRI, namely the optical fiber end locations. As such, the identification of the optical fiber end locations is critically important as it allows the association of a given set of pixels to a given optical fiber and ultimately a given Klems Basis exiting luminance.

The relation between optical fiber ends and pixel location is defined through a function OF of the form,

$$OF : \{1, \cdots, 145\} \longrightarrow [i_{of}] \times [j_{of}]. \tag{4.27}$$

Here, given an optical fiber, recall there are 145 corresponding to the Klems Basis patches, OF will return the pixel location for the center of the optical fiber. The definition of the function OF when actually implemented in code, however, will admittedly be tough to realize. It will link the physical world of space with the computational abstraction of matrix position, hence is the topic of a dedicated section of calibration presented in chapter 5.

With the center pixel location of a given optical fiber $n \in \{1, \cdots, 145\}$ known, (i^n_{of}, j^n_{of}), a surrounding subset of pixels is considered to be a capturing of the optical fiber end. This subset is assumed to be square in shape for simplicity, with a dimension of roughly q BPAs on a side, where q is a positive, even integer.

The proposed BPA subset can now be formally defined for a given optical fiber, $n \in \{1, \cdots, 145\}$, within a given $LDRI_{RPi,e} \in LDRI^6_{RPi}$ as

$$\mathcal{OF}^e_n = \{LDRI_{RPi,e}(i,j) \mid abs(i - i^n_{of}) > \frac{q}{2} \ \& \ abs(j - j^n_{of}) > \frac{q}{2}\}, \tag{4.28}$$

where $LDRI_{RPi,e}(i,j) = \mathcal{P}_c$ located at (i,j). Note, $\forall\ i \in \{1, \cdots, x\}$ and $\forall\ j \in \{1, \cdots, y\}$.

With an ability to identify optical fiber ends via their pixel locations, processing of the set $LDRI^6_{RPi}$ into an HDRI can occur. The proposed process will start by locating $\mathcal{OF}^1_1$ and examining if any of the pixel values have been saturated. Saturation is indicated simply by a pixel value equal to 1023, meaning the magnitude of light excitation was greater than the dynamic range of the first exposure. Put formally, consider the the set,

$$sat^1_1 = \{\mathcal{P}_c \mid \mathcal{P}_c \in \mathcal{OF}^1_1 \ \& \ (\mathcal{P}^1_c == 1023 \ || \ \mathcal{P}^2_c == 1023 \ || \ \mathcal{P}^3_c == 1023)\}, \tag{4.29}$$

and determine if,

$$|sat^1_1| \ > \ 0, \tag{4.30}$$

where $| \ \cdot \ |$ is cardinality of the set. If yes, then saturation occurred, if no, then saturation did not occur . If saturation occurs, $\mathcal{OF}^1_1$ is discarded and $\mathcal{OF}^2_1$ is located. Again saturation is checked, with this processing repeating until the $\mathcal{OF}^e_1$ is located in which no saturation has occurred.

This proposed process of moving to shorter and shorter exposure times can be thought of as allowing for a larger and larger luminance measurement to take place. Hence, when a non-saturated $\mathcal{OF}^e_1$ is found, one can be assured this is the proper exposure for a given optical fiber and thus a given Klems Basis exiting patch [96].

Assuming a non-saturated measurement is located, $\mathcal{OF}^e_1$, in $LDRI_{RPi,e} \in LDRI^6_{RPi}$, a metric must now be calculated to transform the $\sim q^2$ BPAs, which will undoubtedly exhibit their own distribution, into a single number which represents the optical fiber measurement. This process is proposed to begin by transforming all the BPAs from the RPiCM color space, RGB_{RPi}, to CIE-Y values using the method described above in Color Space Transformation.

This will result in the set $\mathcal{OFY}_1^e$ which contains only CIE-Y values in place of the RGB_{RPi} values. Meaning,

$$\mathcal{OFY}_1^e = \{\mathcal{P}_Y \mid \forall\ \mathcal{P}_c \in \mathcal{OF}_1^e\ \exists\ \mathcal{P}_Y = \mathbb{M}\mathcal{P}_c\ \}, \tag{4.31}$$

where $\mathcal{P}_Y$ is the luminance channel from the total $\mathcal{P}_{XYZ}$. From this set, an arbitrary metric is proposed to be calculated and which is considered the measurement of the optical fiber $n = 1$. Stated formally,

$$\mathcal{OFM}_1^e = metric_{of}(\mathcal{OFY}_1^e), \tag{4.32}$$

the actual mathematical manipulation of which is left abstract at this point, because the actual data will need to be explored to determine an appropriate measure. Skipping ahead to the calibration stage, an exploration of possible metrics is covered in chapter 5. The value, $\mathcal{OFM}_1^e$, will then be stored and the process outlined above is repeated for optical fiber two, beginning with the location of $\mathcal{OF}_2^1$. Further it is repeated for all 145 optical fibers, $\mathcal{OF}_3^e, \cdots, \mathcal{OF}_{145}^e$ with $e \in \{1, \cdots, 6\}$.

Absolute SI Unit Anchoring: With the above process completed, a set of 145 optical fiber measurements, $\mathcal{OFM}_n^e \in \mathbb{R}$, $n = \{1, \cdots, 145\}$, $e \in \{1, \cdots, 6\}$, have been processed out from the set $LDRI_{RPi}^6$. Notice, each optical fiber measurement, denoted by n, is free to come from any of the $e \in \{1, \cdots, 6\}$, $LDRI_{RPi,e} \in LDRI_{RPi}^6$. With this in mind, one will notice, all the values are still in a form of bits, scaled arbitrarily based on the output of the CMOS chip and matrix tranformation from $\mathbb{M}$. Thus, one final scaling remains, transforming the $\mathcal{OFM}_n^e$ to a meaningful value of luminance based on which RPiCM generated exposure the measurement was generated.

Knowing that CMOS chips are modeled very well by a linear model, transforming the $\mathcal{OFM}_n^e$ measurement to one of actual luminance will simply involve a *scaler multiple* and *scaler addition*. These scalers are both optical fiber dependent and exposure dependent, hence there will $6 \times 145 \times 2$ parameters which need to be estimated. Their organization can be defined by the functions,

$$\mathcal{S} : n \times e \longrightarrow \mathbb{R}, \tag{4.33}$$

for slope and,

$$\mathcal{I} : n \times e \longrightarrow \mathbb{R}, \tag{4.34}$$

for intercept, where $n \in \{1, \cdots, 145\}$ is the optical fiber and $e \in \{1, \cdots, 6\}$ is the exposure. Hence, for some optical fiber n, which is exhibiting some bit value $\mathcal{OFM}_n^e$ from an exposure e, the corresponding luminance value is,

$$L_v^n = \mathcal{S}(n, e) \cdot \mathcal{OFM}_n^e + \mathcal{I}(n, e), \tag{4.35}$$

with units of $[\frac{cd}{m^2}]$. When all optical fibers are considered, the resulting value is the exiting luminance distribution, $L_v \in \mathbb{R}^{145\times 1}$.

Simulation Execution, Component Coordination and Communication

Cyber-Physical System Architecture: The proposed CUBE 2.0 is a combination of physical and computational processes, hence is a *cyber-physical system* [89]. These computational and physical processes reside in seperate "worlds," and are linked via sensors and actuators, hence their operational behavior is non-trivial, making formal analysis well justified.

To model the entire system operation the system modeling and simulation tool Ptolemy II was utilized [117]. Ptolemy II is an open source, actor-oriented modeling tool which has seen continuous development at the University of California, Berkeley since 1996, under the guidance of Professor Edward Lee [116]. Actors communicate via ports and message passing, with the actual model semantics being governed by a special actor which implements a Model of Computation (MoC). The main strength of Ptolemy II is the well defined, deterministic interaction of actors not only with in individual MoCs, but various MoCs arrange in hierarchal structures. For example, modeling discrete semantics with in a continuous model, akin to a classic digital controller for a physical process.

In this analysis, extended finite-state-machines (EFSM) are used to model the key components of the system: main() running on the MacBook Pro, the RPiCM, and the LabJack T7-Pro equipped with a LiCor Li-210. A Discrete Event (DE) MoC was used to simulate what an actual HWiL simulation would entail [88]. Further, to model communication over the Ethernet LAN, model *aspects* in the form of a *Bus* object are also added to the model.

The background knowledge and subtleties which are accounted for in this type of system modeling and simulation are well beyond the contributions of this thesis, and hence are not reviewed here. They are, however, very valuable skills and can add greatly to better system design. Select Ptolemy II model files, encoded in eXtensible Markup Language (XML), are available for review in Appendix D. These files make up several of the system components included in the CUBE 2.0 system. As a sample, Ptolemy II's graphical user interface (GUI), Vergil, is representing a block diagram of the total system model in Figure 4.9. In this figure, the MacBook Pro, RPiCM, and LabJack T7-Pro can clearly be seen. With in these actors, the EFSMs reside which are representations of both the cyber and physical components in the CUBE 2.0 system.

With this model, may subtle interactions between the various components over the network, as well as intra-component behavior, were explored in the simulation environment. These analyses took place well before any code was written and interaction with actual hardware was yet to be undertaken. This design time line allowed for system design lessons to be learned without needlessly exploring the design space through actual software and hardware development. Examples include system operational steps like whether the RPiCM should wait to hear from main() or if main() should wait to hear from RPiCM when they are communicating. With TCP/IP Sockets, these subtle differences can be implemented exactly as modeled, hence, by working out the operational semantics down to the packet level, much quicker implementation of the actual system was realized.

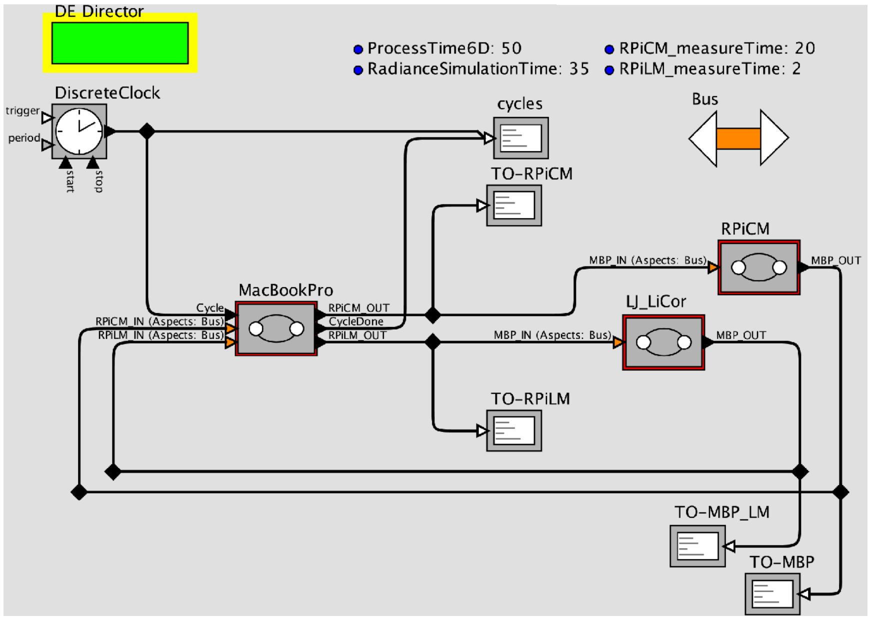


Figure 4.9: Ptolemy II graphical user interface (GUI) representation of the high level model of the total CUBE 2.0 system.

Table 4.3: Timing implementation variables used in improving system architecture.

t^P_{RPiCM}	required time to process $LDRI^j_{RPi}$ into L_v on RPiCM
t_{145}	required time to send processed $LDRI^j_{RPi}$, $LDRI^j_{RPi}$, over LAN
t^P_{MBP}	required time to process $LDRI^j_{RPi}$ into $LDRI^j_{RPi}$ on MacBook Pro
t_{LRI}	required time to send unprocessed $LDRI^j_{RPi}$ over LAN

Example Design Change from System Operational Modeling: One significant system architecture change which was realized in the analysis of the total system via the DE Ptolemy II model, was where the exiting luminance measure data from the RPiCM should be processed. The original design proposed for all processing of the *low dynamic range image*, denoted $LDRI^j_{RPi}$ above in Equation 4.25, set of the optical fiber ends to be done locally on the Raspberry Pi itself, allowing for minimum data transmission over the LAN. However, it became apparent through design space exploration using Ptolemy II, that if the transmission rate of the LAN was sufficient to transport low dynamic range images fast enough (a very reasonable possibility), the entire system may require less processing and transmission time per step if the LDRI set was transported to the MacBook Pro and processed there.

This condition can be stated formally considering the terms defined in Table 4.3. Given the much larger size of $LDRI^j_{RPi}$ when compared to the processed L_v measurement, we assume, $t_{LDRI} > t_{145}$. Thus, the extra time needed to send $LDRI^j_{RPi}$ is $t_{extra} = t_{LDRI} - t_{145}$. Similarly, considering the processing power of the MBP compared to the RPiCM allows for assuming, $t^P_{RPiCM} > t^P_{MBP}$. Meaning the saved processing time can be expressed as, $t^P_{less} = t^P_{RPiCM} - t^P_{MBP}$. Thus, the design decision can be framed as follows:

```
if $(t^P_{less} > t_{extra})$
    {transport $LDRI^j_{RPi}$ and process on MBP}
else
    {process $LDRI^j_{RPi}$ on RPiCM and transport $L_v$}.
```

Seeing the exact timing quantity of these processes is unknown at the design stage, actual deployment was used to test this design alternative. In fact, t^P_{RPiCM} and t^P_{MBP} are $LDRI^j_{RPi}$ dependent given the nature of HDRI processing. As such, several representative $LDRI^j_{RPi}$s were used to test this design consideration.

Using the actual system implementations described below in CUBE 2.0 System Construction, it was determined it is far faster to transport $LDRI^j_{RPi}$ and process L_v on the MacBook Pro. In actual deployment $t^P_{RPiCM} + t_{145} \cong 15$ seconds and $t^P_{MBP} + t_{LDRI} \cong 5$ seconds, making the choice obvious.

Other examples of more subtle system component interaction were also determined via analyses similar to that presented above. While meaningful for designing the system, they are more procedural in nature and therefore not discussed in detail here. The results of these analyses did, however, strongly influence the final implemented system.

Mathematical-Model Simulation of HWiL: Beyond the interaction of physical components and computer code, recall, the CUBE 2.0 system is in fact a hardware-in-the-loop (HWiL) model of a building daylighting system consisting of a complex fenestration system component (CFS). Knowing that so far the physical domain will have been measured for the current time step, focus is drawn to computing the mathematical-model component of the system. That is the Radiance model of the interior of the space.

Radiance and IES Files: At the current stage, the CUBE 2.0 system will have a directionally varying measurement of the exiting luminance distribution, L_v, from the CFS under test. Within Radiance, there exists a well defined entry mechanism for a construct of this type to be inserted into a model as a source of luminous energy, that is the IES file. Typically IES files are used to model electric luminaires [84], however, the principle is the exact same for some CFS transmitting daylight into a space. These files, defined by the IES standard LM-63-02 [12], offer the ability to input a directionally varying luminance source directly into a model by simply including a new object inside the space. Multiple of these objects can be constructed in array type geometries to in fact represent entire CFS units within a space.

By dynamically creating IES files to represent the measured L_v components produced by the RPiCM system, a Radiance model can be simulated with these IES files which represents the interior as if it where being excited by the CFS under test which is being excited by the same luminance distribution which is exciting the CFS on the CUBE 2.0 system. That is, the CFS under test were actually installed in the room Radiance is modeling.

To explore this idea, the Reference Office Radiance model [126] was augmented, replacing the glazing unit with an array of IES file objects which could be dynamically altered between simulation executions. In this office, horizontal illuminance was calculated on the working plane of 0.8 meters over the entire floor plane in a grid spaced at 10 cm intervals in both dimensions. This calculation is a simple call of the Radiance program *rtrace* with the appropriate parameters defined, of which [126] give suggestions. It should be noted, defining simulation parameters within Radiance is a notoriously difficult exercise, where experience pays and small perturbations can be very impactful to results.

The result of this trial showed that in fact, IES files can be dynamically created (eventually from L_v measurements), from which Radiance models can be recompiled (i.e. formed into octree's using *oconv*) and simulated (*rtrace* on a grid of points), from which meaningful data can be derived (working plane horizontal illuminance). Due to the nature of Radiance, near limitless other luminous metrics could also be computed from the compiled octree's. While using IES files was the originally proposed idea for the linking of physical measurements and the mathematical-model, the discovery of the three-phase method in the course of this thesis work offered a new idea.

Three Phase: View Matrix To begin, notice the exiting luminance distribution, L_v, which is being measured is the same quantity as the term $T \cdot D \cdot s$. Thus, with this understanding one can compute any luminous metrics which the three-phase method can simulate using this HWiL architecture by precomputing the appropriate view matrix V and simply multiplying it by L_v. With this new idea, the Radiance model and simulation are precomputed, meaning, instead of calling a Radiance system call each step, only a matrix multiple is needed. Further, instead of relying on the appropriate modeling of the total CFS via an array of IES file defined objects, the view matrix, V, which is computing using advanced stochastic methods with respect to ray origins and angles of travel, can be constructed for a given CFS installation [101]. This offers more confidence in the introduction of the measured luminance into the Radiance model. As an added system operational benefit, matrix multiplication has a very low chance of crashing, yet Radiance, an open source freeware program, is not so robust. Hence, by using the three-phase method, added confidence is placed in the mathematical-model execution as compared to system calls to execute Radiance.

Considering the above analysis, the three-phase Radiance model simulation is used in the CUBE 2.0 system. As such, a view matrix, V_{cfs}, was computed using the Radiance program *rtcontrib* for the Klems Basis exiting the CFS and a 10 cm spaced grid of horizontal illuminance measurements at the work plane height of 0.8 meters. Thus, the dimensions of V_{cfs} are 2835×145, with the 2835 factor being reshaped into a grid with dimensions 81×35, which represents the horizontal illuminance at the workplane height of 0.8 meters.

Physical Structure

Frame: A static structural analysis of the frame was conducted using the structural analysis software RISA-2D and showed the proposed design had approximately 10 times the needed capacity for even the highest predicted gravity loads [132]. The 10× figure itself

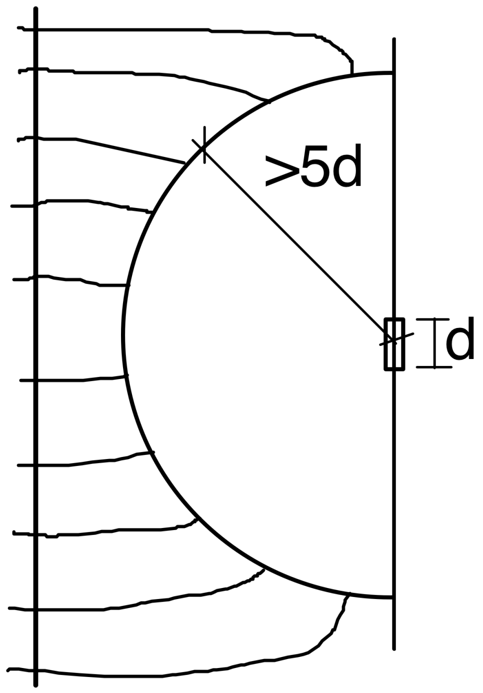


Figure 4.10: Proposed design for the CUBE 2.0 enclosure made from painted plywood.

is conservative as connection strengths were taken from manufacturer specifications with reduction factors applied. This analysis is beyond the scope of the thesis, however, and not presented here. Often 8020 sections are used in "unengineered" applications, hence, the design was approved with a high confidence of meeting the requirements.

"Light Proof" Enclosure: Another consideration for the CUBE 2.0 system enclosure, to be made of plywood, is how large to make the space containing the optical fiber ends, called subsequently the *measurement cone*. Given the RPiCM camera has a fixed focus of 1 meter to infinity, the closest the optical fiber ends can be to the camera lens, and remain in focus, is 1 meter. With this, as well as the field-of-view (horizontal: 53.50±0.13°, vertical: 41.41±0.11° [121]) of the camera in mind a distance of 1.1 meters is chosen. The field-of-view is important as all of the optical fiber ends must fit within the captured image inside the CUBE 2.0 *measurement cone*. Basic geometry reveals the image size at 1.1 meters is 1.11 meters wide and 0.83 meters tall. This is more than enough space to capture all the optical fiber ends.

Electrical Power

To test the proposed electrical power design, a USB splitter was purchased and all the devices where connected and powered on. The USB splitter had no issue delivering power to the LabJack T7-Pro, RPiCM, and Ethernet Router off the USB power grid of the MacBook Pro. Further, the Canon 6D was tested and can take several thousand photographs on a single charge.

At this stage, the length of time the MacBook Pro battery can run the CUBE 2.0 is unknown. It is believed if more energy is needed than can be provided by the MacBook Pro battery for a given experiment, supplemental power can be delivered to the MacBook Pro through its standard 120 V AC power cord. This cord can be either plugged into grid power or connected to a portable generator or battery bank such as a Goal Zero 23000 Yeti 400 Solar Generator [53]. Further, the Canon 6D can also be plugged into a standard 120 V AC outlet for limitless power and simulation time.

4.3.3 Final CUBE 2.0 System Design

The final design of the CUBE 2.0 system presented next is the result of the initial proposed design being modified after various analyses were conducted both at the super-system level as well as the sub-system level. Thus, presented here is a summary recap of the initial design proposed above. For extensive details of each system component as they are implemented, see below.

Light Excitation Source

As with the initial design, both natural (i.e. outdoor sunlight) as well as artificial light sources is used as excitation for the CFS undertest.

Measure Input Illuminance Distribution

Measuring the input illuminance distribution is achieved using a Canon 6D equipped with a Sigma 8mm F3.5 EX DF fisheye lens. The camera and lens is controlled via a custom program running on the MacBook Pro which uses the EOS SDK. Further, a LabJack T7-Pro and a LiCor LI-210 illuminance meter, also queried by the MacBook Pro, is used to calibrate the gathered input illuminance distribution to absolute SI units of candela per meter squared $[\frac{cd}{m^2}]$. After downloading the low dynamic range image (LDRI) set to the MacBook Pro, it is formed into $HDRI_{input}$ by *hdrgen*. Then a custom C program will first calibrate the HDRI using the gathered illuminance measurement, after which the average illuminance corresponding to the Klems Basis input solid angles is calculated, E_v, and saved to a file.

Measure Exiting Luminance Distribution

An acrylic hemisphere holding optical fiber ends at the various Klems Basis angles centroids will “bend” the exiting luminance distribution from a three dimensional to a two dimensional measurement. A RPiCM will then capture a series of LDRI of the optical fiber ends. This LDRI set, $LDRI^{j}_{RPi}$, is sent to the MacBook Pro where is is processed using a custom C program into a measurement corresponding to the appropriate Klems Basis angle exiting the CFS undertest, L_v. These values is saved to a file for future analysis as well as input into the Radiance simulation.

Simulation Execution, Component Coordination and Communication

A Python 2.7 program, called main(), will run on a MacBook Pro coordinating the entire CUBE 2.0 cyber-physical system during simulation. The program will control the various aspects via both Ethernet and USB as well as output signals using pulse width modulation (PWM), analog voltage, and I^2C among others. Main() will also execute the mathematical model (i.e. Radiance in a Three-Method type architecture) corresponding to the "cyber" domain which represents the inside of the building daylighting system.

Physical Structure

The physical structure of the CUBE 2.0 system is made form 8020 aluminium sections and plywood. It will hold the CFS undertest as well the acrylic hemisphere, optical fibers, "light proof" enclosure, and the various sensors (e.g. Canon 6D, RPiCM). The frame and enclosure are rigid enough to provide structure, yet light and small enough to allow for easy portability through standard doors.

Electrical Power

As stated above, the MacBook Pro and Canon 6D will run off self contained battery power. The LabJack T7-Pro, RPiCM, and Ethernet Router will run off the USB power of the MacBook Pro. If more energy is needed for longer deployments, the MacBook Pro and the Canon 6D can be plugged into grid power, a portable generator, or a battery bank.

4.4 CUBE 2.0 System Construction

Here the details of the construction process of the CUBE 2.0 system are presented. Starting from raw materials, the step-by-step construction process for each component is covered. Supplementary pictures in the appendices provide further visual clarity. While the preceding two sections, CUBE 2.0 System Specifications and CUBE 2.0 System Design, ordered their discussion based roughly on the chronological ordering of tasks in a single HWiL simulation step, this section orders the discussion chronologically with respect to the actual component construction which was executed in completion of this thesis.

It is hoped this section will clear up any details of the CUBE 2.0 system implementation which were presented in a more abstract manner above.

4.4.1 Physical Components

Documenting the construction of the CUBE 2.0 system is organized into two parts. First, the physical components are covered, followed by the cyber components.

CUBE Frame, Enclosure, Measurement Cone

Frame: 8020 1.50”×1.50” T-Slotted Profile aluminum sections were ordered in their standard lengths of 144 inches (12 feet) from the manufacturer. In addition, both Standard End Fasteners (SEF) and 15 Series 5/16”-18 Standard Anchor Fastener Assembly (SAFA) connectors were ordered. Using a standard band saw, the sections were “cut long” to the specified lengths with a tolerance of $\frac{1}{8}$”. A disk sander was then used to round off rough edges and shape the sections to visually perfect dimensions using a standard meter stick with one millimeter markings as a reference.

Once all sections were cut to proper length, a tap ($\frac{5}{16}'' - 18$) was used to ready the ends of the necessary sections acting as columns for the SEF connectors. No drilling was needed as the section profiles come preformed for tapping of the proper size connector. In addition, a mill was used to cut the inlays for the SAFA connectors where needed. The frame was then constructed using the respective connectors and can be seen in Figure 4.11.

Enclosure: With the frame now assembled, plywood sheets (3/8”, 4’×8’ - standard dimensions) were cut sequentially using a table saw and skill saw (i.e. electric hand saw). Two panel types were made, the first to cover the 8020 frame, and the second to construct the *measurement cone*.

For the panels designed to cover the 8020 frame, $\frac{3}{8}$” holes were drilled at the four corners of each panel which align with the 8020 frame. Then the 8020 frame was drilled and tapped at $\frac{1}{4}'' - 20$, into which 2” long threaded rod sections were inserted. The plywood panels were then secured in place by aligning the threaded rod and drilled holes. A washer and nut were then used on each threaded rod to secure the panels to the 8020 frame. See Figure 4.12 for the CUBE 2.0 8020 frame with the enclosure panels attached.

Measurement Cone: For the panels designed to form the measurement cone, the joints were wood glued and 2” finishing nails were pneumatically driven to hold in place the connections. Each edge of these panels was bevelled such that the final board dimensions fit together with maximum surface area for gluing and nailing, resulting in maximum strength.

To further strengthen the measurement cone, six $\frac{1}{8}'' \times 1'' \times 8''$ steel bars were both glued (Gorilla Glue) and screwed through pre-drilled holes into the plywood sheets. These bars were bent in such a manner as to fit the contour of the measurement cone closely, reinforcing the primary joint of the cone. Similarly, $2'' \times 2''$ steel angle sections approximately $2''$ wide where also glued and screwed via one “leg” onto the measurement cone surface. The remaining “leg” contains a $\frac{3}{8}''$ whole, though which $\frac{1}{4}''$ threaded rod equipped with a nut and washer is run. These threaded rods are then screwed into T-Nuts inserted into the 8020 frame. These threaded rods then are tightened allowing the measurement cone to be pulled

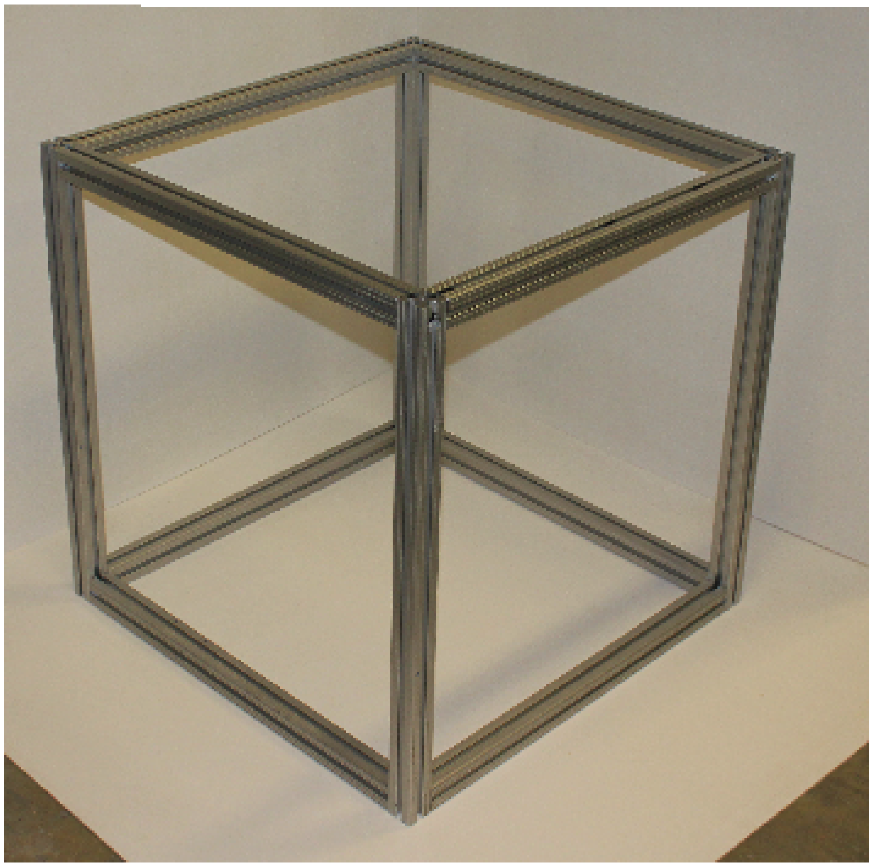


Figure 4.11: The completed 8020 frame of the CUBE 2.0 system.

tight against additional wooden blocks glued and nailed to its surface and the 8020 frame. In this manner, the measurement cone is securely attached to the 8020 frame.

With both the enclosure and measurement cone panels constructed, they were joined together to form the total basic structure of the CUBE 2.0 system, which can be seen in Figure 4.13. Note, the unmentioned box (unpainted) protruding from the top of the CUBE 2.0 is an enclosure for the MacBook Pro, LabJack T7-Pro, and other electronic components of the system.

Optical Fiber Array

With the enclosure panels cut to size and their connectors to the frame (i.e. $\frac{1}{4}'' - 20$ threaded rods), installed, it is time to turn to the optical fiber array. The optical fiber array is comprised of three components: acrylic hemisphere to hold the "sampler" optical fiber end, optical fibers, and a plywood sheet which holds the "measurement" ends of the optical fibers inside the measurement cone.

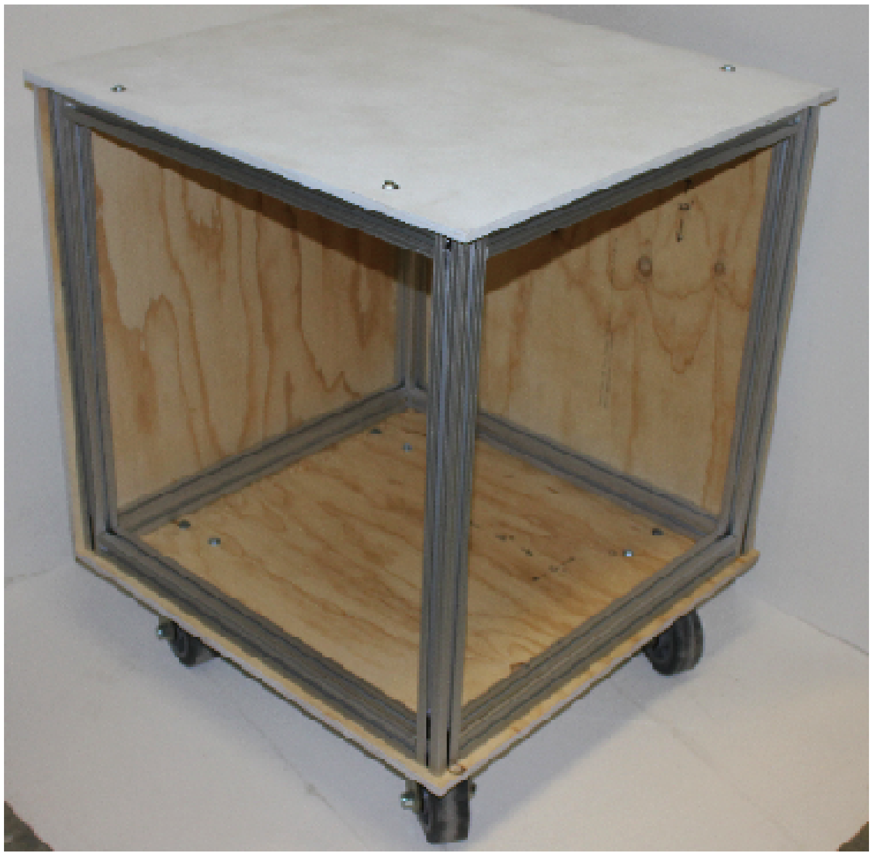


Figure 4.12: The complete enclosure of the CUBE 2.0 system.

Construction began with a standard 24 inch diameter clear acrylic hemisphere. The hemisphere was then placed on a custom made turn table which allowed for the marking of the Klems Basis solid angle patches using a Sharpie permanent marker. This resulted in a highly accurate demarcation of the areas in which the optical fiber ends should be placed. Holes ($\frac{1}{4}''$) were then carefully drilled through the acrylic hemisphere at the centroid of each of the Klems Basis Patches. These holes will hold the "sampling" end of the optical fibers.

With the holes for the optical fiber ends drilled, white paint was then sprayed on the inside of the hemisphere. While matte black paint immediately followed and is the needed color for the exiting luminance distribution, L_v, measurement space, white painted preserved the Sharpie markings when viewed from outside of the hemisphere. Next, the marked, drilled, and painted acrylic hemisphere was mounted using a cyanoacrylate based adhesive (i.e. super glue) to the corresponding CUBE 2.0 enclosure plywood sheet.

Using a custom made cutting device, the optical fibers were cut to length and hot-melt glued into the holes in the hemisphere. Note, small wood blocks, with holes through which the optical fibers passed, were used to more effectively mount the optical fibers to the

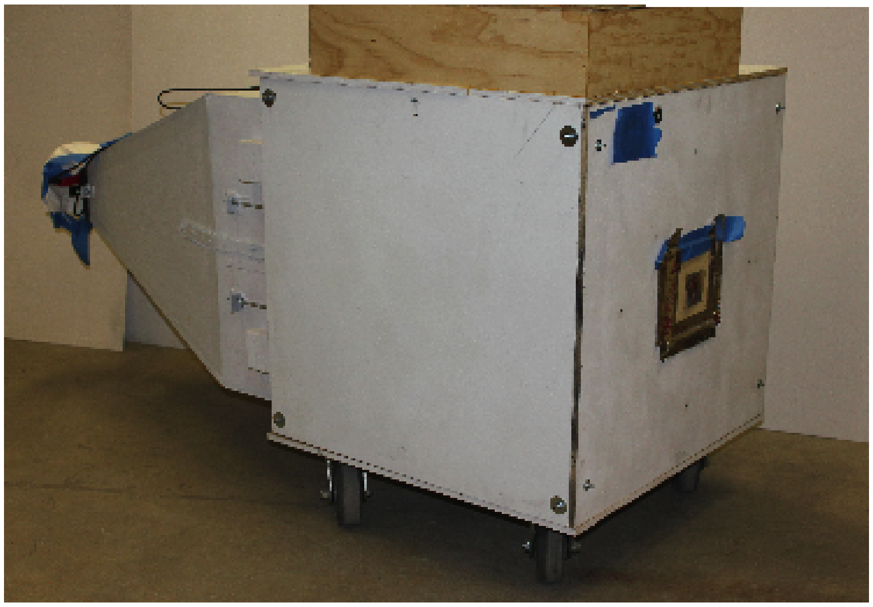


Figure 4.13: The complete enclosure of the CUBE 2.0 system with the *measurement cone* installed.

hemisphere surface. The wood blocks offered two main advantages, more surface area for which glue could be applied, and a friction fit for the optical fiber "sampling" end, allowing for depth adjustment of the optical fiber end once inside the hemisphere for sampling.

In total there are 145 optical fibers, one for each of the Klems Basis Patches. The length of each optical fiber is the same per theta ring of the Klems Basis, but varies from ring to ring, hence there are nine lengths used in total. There is, however, no significance to these lengths other than they be sufficient to reach the measurement cone mounted plywood sheet, and not impose angles of bending which would affect the total internal reflection of the optical fiber as calculated above.

This plywood sheet then forms the enclosing cap of the measurement cone and holds the ends of the optical fibers which are measured by the RPiCM. It is marked with the Klems Basis Patches projected onto a plane, and drilled with $\frac{1}{4}''$ holes for each complementary optical fiber end with respect to the hemisphere. The "measurement" ends of the optical fibers are then inserted into this plywood sheet, enclosing the measurement cone and completing the sampling process of exiting luminance measurement. The total optical fiber array system can be seen in Figure 4.14.

Raspberry Pi Camera Module Bracket

With the CUBE 2.0 frame, enclosure, measurement cone, and the optical fiber array complete, the RPiCM needs to be placed within the measurement cone to capture the "measurement" ends of the optical fibers. Alignment of the RPiCM is critical to ensure the ends

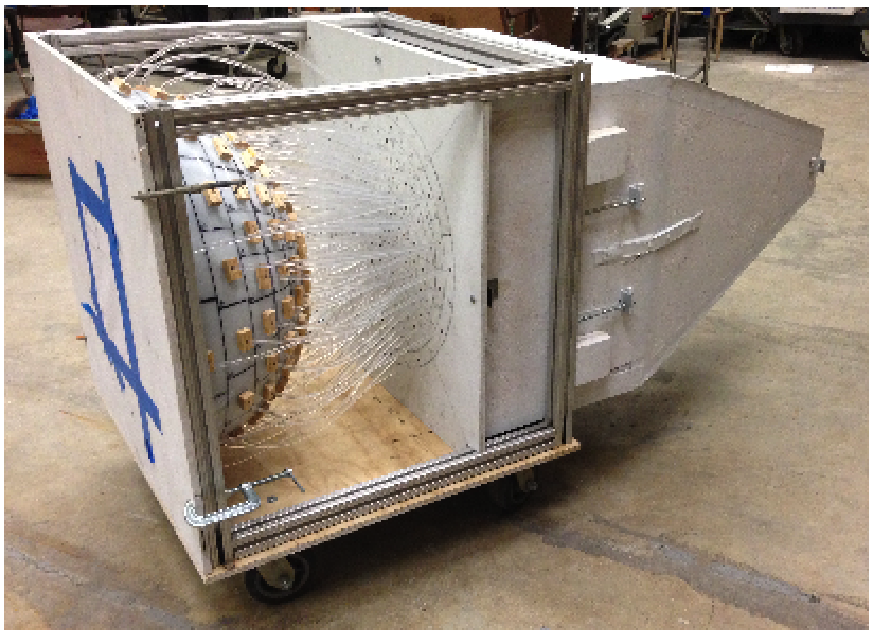


Figure 4.14: The total optical fiber sampling system from exiting luminance measurement hemisphere holding the "sampling" optical fiber end, through optical fiber "measurement" end terminating at the *measurement cone.*

of the optical fibers are captured in a consistent manner. As such, a custom camera holder was 3D printed (a.k.a. additive manufactured) from acrylonitrile butadiene styrene (ABS) plastic using a MakerBot Replicator 2X [97]. This ABS frame was then mounted onto a plywood sheet using threaded rods, springs, nuts, and washers. The tripod type design of the ABS camera frame allows for extremely fine adjustment of the RPiCM alignment with respect to the measurement cone, by turning the threaded rods in a controlled manner.

Ethernet Local Area Network (LAN)

Given the power constraints of the CUBE 2.0 system, a USB powered Ethernet router was selected. Technically a *switch* is what is actually used here, specifically the Dualcomm USB Powered 5-Port 10/100Base-T Ethernet Switch [152]. It can connect 5 devices, hence is perfect for the CUBE 2.0 with three devices. The switch was plugged into the USB power splitter and Ethernet cables were plugged into the RPiCM, LabJack T7-Pro, and via a Thunderbolt to Gigabit Ethernet Adapter, the MacBook Pro.

Electrical Power Network

Five components in the CUBE 2.0 require power: MacBook Pro, Canon 6D, LabJack T7-Pro, RPiCM, and the Ethernet switch. Two of these, the MacBook Pro and the Canon

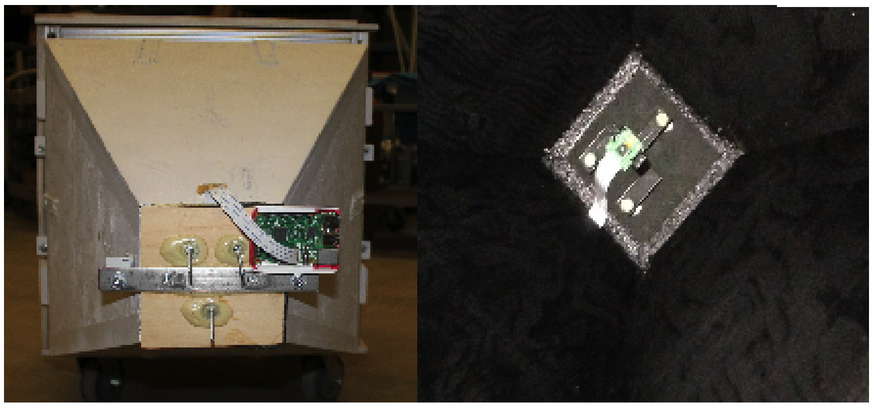


Figure 4.15: Left: RPiCM from outside of CUBE 2.0. Right: RPiCM inside the measurement cone.

6D have self contained battery power. Further, the LabJack T7-Pro, RPiCM, and Ethernet router can be powered off of a standard USB drive. This lead to the design pictured in Figure 4.16, where a USB splitter is used to distribute power.

Given the battery life of the MacBook Pro and Canon 6D, experience tells the system can operate for approximately 3-4 hours in this arraignment. Initially, during the CUBE 2.0 calibration period, this duration of testing was deemed appropriate as runs of this length were more than adequate to collect calibration data sets. Hence the electrical power network was used for real gathering of data.

With longer, day long runs the ultimate goal, however, a Goal Zero Yeti 400 battery [53] was purchased. The Yeti 400 battery is connected to the system via an extension cord connected to a splitter. The particular splitter has both USB and 120 V 3 prong outlets (Type B), allowing for direct connection of the USB splitter, MacBook Pro and Canon 6D power cords. With this configuration, the CUBE 2.0 system can now operate for periods of up to 48 hours on battery power. Further, if grid power is available it can operate indefinitely.

CFS Specimen Holder

Given the CUBE 2.0 system is designed to test complex fenestration systems (CFS), the specimens must be held in proper position with respect to the CFS test aperture. Originally, a $\frac{3}{8}''$ thick plywood sheet was precisely cut to fit into a square hole made in the front enclosure panel. That is the same panel with the acrylic hemisphere adhered to the back. A one inch circular hole was then drilled through using a hole saw, after which it was attached to the CUBE 2.0 enclosure via aluminum (i.e. light proof) tape. CFS specimens where then taped over the drilled hole.

This design failed, however, because of the relatively thick $\frac{3}{8}''$ plywood sheet. At large angles of incidence and existence (i.e. the outer two theta rings, Klems Basis Patches 118-145) the angle of view was interrupted. As such, a $\frac{1}{32}''$ thick brass plate was used instead

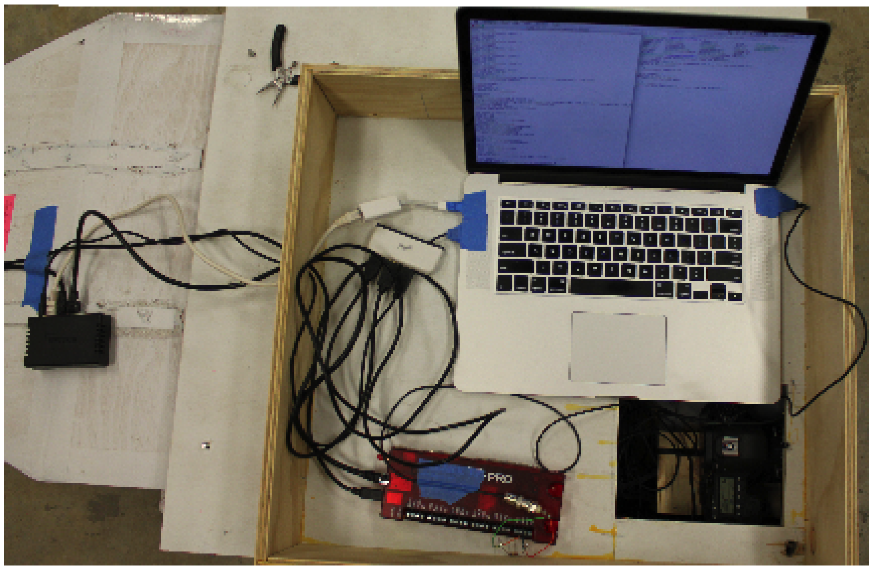


Figure 4.16: All devices requiring power in the CUBE 2.0: MacBook Pro, Canon 6D, LabJack, Ethernet Router, and the RPiCM is just off frame left.

of the plywood sheet. This brass plate too was drilled with a one inch diameter hole via a hole saw. It also was connected to the CUBE 2.0 enclosure via aluminum tape. The brass CFS specimen holder indeed works and is used in the CUBE 2.0. CFS samples are adhered to the front of the CUBE 2.0 with painters tape, or in heavier examples, they are connected via clamps to the 8020 frame.

Canon 6D, Sigma Fisheye Lens, and LiCor LI-210

As proposed above, the CUBE 2.0 system will measure the input illuminance distribution using a Canon 6D equipped with a Sigma 8mm F3.5 EX DG equisolid-angle projection fisheye lens. In addition, a LI-210 illuminance meter will be used to anchor the input measurement to absolute SI units of candela per meter squared $[\frac{cd}{m^2}]$.

As such, the Canon 6D must be secured to the CUBE 2.0 structure in a fixed manner to ensure consistent alignment of the lens and the CFS test aperture. A further requirement of the Canon 6D and LI-210 mounting is they not only be as close together as possible, but further, they be as close as possible to the CFS test aperture. This stems from the requirement they all (i.e. Canon 6D, LI-210, CFS test aperture) are attempting to measure the same photometric quantity which is the input illuminance distribution, E_v, onto the CFS undertest.

During the construction and subsequent calibration of the CUBE 2.0 system, two Canon 6D and LI-210 mounting schemes were realized. The first mounting scheme consisted of 3" and 1" diameter holes being cut in the front enclosure panel for the fisheye lens and LI-210 sensor respectively. The LI-210 was then friction fit into the 1" hole using paper shims. For the Canon 6D, a small plywood platform was constructed on the inside of the CUBE 2.0 structure, to which it was bolted using the standard tripod connector, a $\frac{1}{4}'' - 20$ bolt, nut, and washer.

The final result was an organized system, enclosed from the elements, which could be deployed to various testing locations. However, it proved difficult to ensure camera position remained fixed within the CUBE 2.0 system. This was evident as even the slightest "bump" to the front of the CUBE 2.0 enclosure would cause the camera to change alignment. While these changes were small, they were significant for a calibrated light measuring instrument. Further, accessing the Canon 6D to adjust settings by hand, or clean the lens was exceptionally challenging.

As a result of the inability of the first mounting scheme to hold the Canon 6D in position, and its poor camera accessibility, a second scheme was implemented. This second scheme, mounts the Canon 6D on top of the CUBE 2.0 enclosure plywood board, while keeping the LI-210 in its original position. The new mounting allows for a more secure connection with the tripod mount and the top of the CUBE 2.0, hence alignment is easier to maintain. In addition, accessing the camera via the control buttons and LCD screen, as well as service the lens, are eminently more practical.

The new Canon 6D mounting scheme subsequently required manufacturing a new plywood panel for the top enclosure section of the CUBE 2.0. This new enclosure panel is also securely attached to the CUBE 2.0 8020 frame via Allen keyed bolts directly into the top of the 8020 column sections. This allows for the most consistent alignment possible, further adding to the benefits of the second Canon 6D mounting scheme.

In addition, a new enclosure was constructed to house the Canon 6D along with the MacBook Pro, LabJack T7-Pro, Ethernet Router, and USB connector. Given the Canon 6D is now on top of the CUBE 2.0, the enclosure was fitted with a slot to allow the fisheye lens to remain in place even while the enclosure is opened and closed. Further, an additional secondary enclosure was made for the RPiCM to protect it from exposure as well.

In addition, the LabJack T7-Pro was placed on the CUBE 2.0 and the LI-210 was hooked up via a BNC connector. Due to the fact the LI-210 is a current producing sensor, a 604 ohm precision resistor was used to transform this current into a voltage. This voltage is then read via an analog-to-digital input channel. Using a simple linear model for the LI-210, the calibration constant is then used within the cyber components to get meaningful vertical illuminance readings which can be used for calibration of the Canon 6D fisheye generated HDRI to absolute SI units of $[\frac{cd}{m^2}]$.

CUBE 2.0 Transportation and Support Frame

The entire CUBE 2.0 system weighs approximately 80 pounds. While this is quite manageable, it is an awkward 80 pounds, with the total CUBE 2.0 dimensions coming in at $28'' \times 28''$ (plus enclosure for height) and 65" long. This makes transportation and handling inconvenient. As such, the CUBE 2.0 has the ability to have casters attached to the bottom.

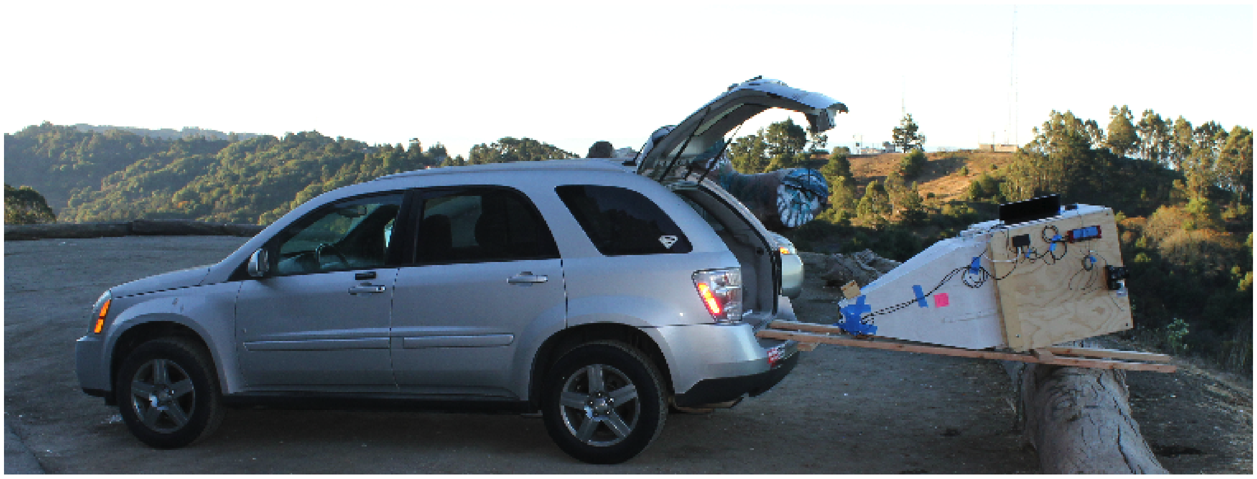


Figure 4.17: CUBE 2.0 deployed from a SUV at Tilden Park in Berkeley, California.

These wheel units can be easily connected and/or removed, depending on the application at hand, via drilled and tapped holes into the primary 8020 frame.

In addition to casters, the CUBE 2.0 system has carriage bolts attached to the bottom panel with only the heads protruding. These steel "dishes" of sorts, make low friction contact with the supporting surface. Meaning, the CUBE 2.0 can be easily slid along the surface it is resting for surfaces with low coefficients of friction between themselves and steel (e.g. concrete). In this orientation, the CUBE 2.0 can be relatively easily placed in a sport utility vehicle (SUV) and transported to various testing sites. Once at a testing site, a $2'' \times 4''$ frame can be deployed. With one end attached to the back of the SUV, and the other free standing, this frame supports the CUBE 2.0 and allows for testing at any motor vehicle accessible location. Figure 4.17 shows the CUBE 2.0 deployed in Berkeley, California at Tilden Park. Here the $2'' \times 4''$ frame is attached to a Eucalyptus tree log.

For more stationary tests, a permanent frame has also been constructed, and is housed on the roof of Cory Hall at the University of California, Berkeley. This frame can be rotated to face different orientations, and can store the CUBE 2.0 system for extended periods of time, including through inclement weather as it is equipped with a water proof cover.

CFS Actuator

It is believed most CFSs tested on the CUBE 2.0 containing augmentable components will be totally self contained. That is, they will interface to controllers using digital or analog signals and not need physical utilities from the CUBE 2.0 system beyond a physical mounting point. With this in mind, yet wanting to demonstrate an ability to "close-the-loop" with respect to a HWiL simulation, a basic "open or closed" CFS actuator system was constructed.

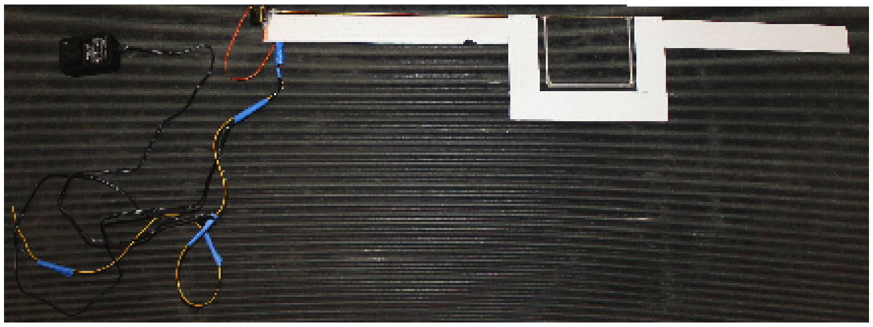


Figure 4.18: CFS actuator used to demonstrate CUBE 2.0's ability to conduct HWiL simulations with "closing-the-loop" capabilities.

This CFS actuator system, see Figure 4.18, contains three components: i) frame, ii) servor motor, and iii) CFS shade mount. These components were made from wood slates and aluminium tube sections of $\sim \frac{1}{16}''$ diameter. Using a PWM signal from the LabJack T7-Pro, the servo motor can thus control a CFS shade mounted on the device to either be "closed" or "open." This allows a control law to be calculated on the MacBook Pro which can be easily output and realized in the physical world.

While this system is admittedly simple, the purpose is to demonstrate the CUBE 2.0 capability, not in the actual analysis it provides.

4.4.2 Cyber Components

With the physical construction completed, the software components, referred to here as the "cyber components" were built. The process involved in the writing of these programs is overviewed below, while the actual code can be viewed in Appendix D.

main()

main() is a Python 2.7 program running on the MacBook Pro. main() controls all aspects of the simulation. Its general structure can be grouped into three stages: i) start-up, ii) looping, and iii) shut-down. Considered as a graph, these stages take the form shown in Figure 4.19. For a more detailed graph representation of main(), complete with line numbers, see Appendix D, section D.1.

Basic Structure: The writing of main() was thus begun by outlining the code as to perform in three distinct states. The program begins with several variable declarations and object instantiations. Next, it enters a while-loop in which it remains for the duration of the simulation. Finally, once the simulation termination criteria are reached, it exits the while-loop, cleans up connections, closes files, and terminates.

With this basic structure complete, "dummy functions" containing only sleep and print statements where written for the various tasks which are needed for main() to complete.

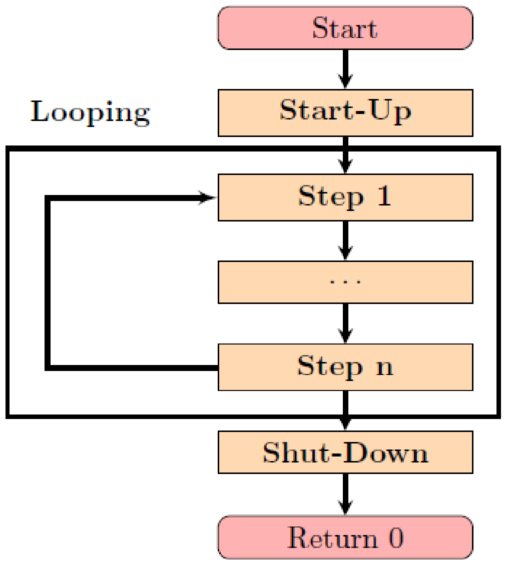


Figure 4.19: Simple outline of main()

These "dummy functions" were then inserted into the code outline in the corresponding start-up, looping, and shut-down sections of the outline. These tasks can be thought of as directly relating to the system specifications which were stated above concerning the total hardware-in-the-loop modeling system.

At this time, the main() program development was paused, and development of the RPiCM() program, running on the RPiCM, commenced. When RPiCM() was at a similar stage of development, complete with "dummy functions," concurrent development of main() and RPiCM() continued forward.

Initial Networking with RPiCM(): Seeing main() and RPiCM() must communicate over the Ethernet LAN using TCP/IP, both programs were given "localhost" IP address configurations with appropriate Port Numbers for use with the Python Sockets module. Still with both main() and RPiCM() containing nothing but "dummy functions," the two programs were developed further to include socket declarations which were used in both *client* and *server* capacities. The RPiCM() can be thought of as a "server" which is queried by main() acting as a "client" to take photographs which will eventually become the exiting luminance distribution measurement, L_v. The exact ordering of these interactions was straight forward to implement as the semantics were predetermined via the Ptolemy II model discussed earlier.

With the sockets up and running, both main() and RPiCM() were able to interact locally on the same machine, executing "dummy functions" simulating the total system functionality. To increase confidence main() and RPiCM() would also behave in this manner when actually deployed in the CUBE 2.0 system, a bench-top LAN was setup on which the same local execution behavior was replicated on a MacBook Pro (IP: 10.0.0.12) and a Raspberry Pi 2 Model B (IP: 10.0.0.13).

With the Ethernet networking capabilities established, focus was shifted toward replacing the "dummy functions" with meaningful code. These tasks are outlined in the paragraphs below.

Connecting to the LabJack T7-Pro: Given the manufacturer supplied Python module, LabJackPython, and the associated configuration program for the physical LabJack T7-Pro device, this connection was straight forward. First, configure the LabJack T7-Pro by installing the drivers and assigning a static IP address (IP: 10.0.0.14) using the provided desktop program, Kipling 3. Then, inside Python code, instantiate and configure an object, then call methods to get voltage values into the Python code. Using the sensor model proposed above, vertical illuminance, ill_{LC}, can be calculated as,

$$ill_{LC} = \mathcal{C} \cdot (V_{LC} - V_{dv}), \tag{4.36}$$

where $\mathcal{C}$ is the manufacturer provided calibration constant, V_{LC} is the voltage across a 604 ohm precision resistor used to convert the current to a voltage, and V_{dv} is the dark voltage associated with a measurement of zero illuminance. $\mathcal{C}$ is hardcoded in as it is provided by the manufacturer, V_{dv} is calculated as part of the start up process, and V_{LC} is the actual voltage level returned from a query to the LabJack T7-Pro. ill_{LC} can thus be calculated whenever the V_{LC} is queried, which occurs once per simulation step in a small batch and can be seen in the main.py code on lines 269-293, or in Block 4 of the code graph in Figure D.1.

Executing Input Illuminance Measurement: Each input illuminance measurement, of which there is one per simulation time step, consists of three steps: i) capture and download $LDRI^{11}$ using the Canon 6D with fisheye lens, ii) process that $LDRI^{11}$ set into a single $HDRI_{input}$, and iii) process that $HDRI_{input}$ into a Klems Basis input vector, $E_v \in \mathbb{R}^{145\times1}$.

Due to the fact that main() is written in Python and doesn't need to read in the large sized $LDRI^{11}$ and $HDRI_{input}$ photograph files themselves, it was decided to write individual programs in C to execute each of these steps. These programs are then called sequentially as a SubProcess call by main() of a Shell script. C was chosen primarily because of the Canon API [27], but also due to its speed and the author being familiar with this language.

Hence, to take an input illuminance measurement, main() calls the shell script, 6D.sh. 6D then sequentially calls fisheye6D to capture and download $LDRI^{11}$, hdrgen to process $LDRI^{11}$ to a single $HDRI_{input}$, and finally, fish2klems to calibrate and transform the $HDRI_{input}$ into a Klems Basis input illuminance vector, E_v. Each of these programs is discussed below in their respective sections. Examining main.py these executions can be found in the code on lines 295-311, or in Block 5 of the code graph in Figure D.1.

Executing Exiting Luminance Measurement: In order to execute the input and exiting measurements in parallel, the exiting luminance measurement occurs in two steps. First, the RPiCM() is notified to take a LDRI measurement of the inside of the measurement cone, which also involves a confirmation message in reply. The corresponding code for this interaction with RPiCM() can be seen in main.py on lines 242-267, or in Block 3 of the code graph in Figure D.1. Then, after the input illuminance measurement has been executed, the $LDRI_{RPi}^{j}$ set is downloaded and processed locally on the MacBook Pro.

While the first step of notification involves simply sending a string of characters via TCP/IP sockets and processing a reply, the second step is more complicated. Initially main() must check with RPiCM() and determine that the $LDRI_{RPi}^{j}$ measurement is indeed completed. Then, $LDRI_{RPi}^{j}$ must be downloaded and finally it must be processed into a Klems Basis exiting vector, L_v.

Checking $LDRI_{RPi}^{j}$ measurement is complete is straight forward using TCP/IP sockets, and is completed similarly to notifying RPiCM() to take a measurement. Transferring $LDRI_{RPi}^{j}$ from the Raspberry Pi to the MacBook Pro is accomplished by a call to a shell script, rpi.sh which uses another shell script, ssh2RPiCM.sh, running through the Secure File Transfer Protocol (SFTP), to enter the Raspberry Pi and securely download $LDRI_{RPi}^{j}$. With $LDRI_{RPi}^{j}$ now located on the MacBook Pro, a final shell script, proc.sh, is called which subsequently calls a C program, Cprog, to process $LDRI_{RPi}^{j}$ into a Klems Basis exiting vector, L_v. All of these executions can be seen in main.py on lines 313-388, or in Block 6 of the code graph in Figure D.1.

Calling Radiance Model and Closing-the-Loop: Given the Klems Basis exiting vector has been formulated, L_v, it is now time to call the Radiance component, or the mathematical model of the hardware-in-the-loop model and simulation which the CUBE 2.0 is executing. As chosen in the modeling process above, a three-phase inspired execution will be done for this Radiance model. This involves first reading in the precomputed view matrix, V_{cfs}, corresponding to the particular model which is being simulated. Then, multiplying L_v with V_{cfs} and parsing the results. This is accomplished with a call to the function "illumMeasure." If multiple models are desired, "illumMeasure" can be called multiple times with varying V_{cfs} and like-wise appropriate file names to save the results[4]. These results can also be parsed and input into control laws at this stage if an augmentable CFS is being tested. With output of the control law, these values can then be converted from the cyber work to the physical world via standardized outputs such as PWM or I^{2}C using the LabJack T7-Pro.

In addition, during validation of the CUBE 2.0 system (discussed in chapter 5), CFSs with known BTDFs will be tested. As such, the results of the CUBE 2.0's measurement system can be compared to the theoretical transmission of the measured input, E_v, multiplied by the BTDF. More formally, for a CFS of known BTDF, denoted T, a comparison can be made between, $i_m = V_{cfs} \cdot L_v$, and $i_{m,s} = V_{cfs} \cdot T \cdot E_v$. More succinctly, the exiting luminance distributions can also be compared L_v and $L_{v,s} = T \cdot E_v$. These comparisons, while not exact due to $L(\theta,\phi)_{fish} \approx L(\theta,\phi)_{CFS}$ and not $L(\theta,\phi)_{fish} = L(\theta,\phi)_{CFS}$, are at least a partial confirmation of the accuracy which one can expect given the CUBE 2.0 system. This comparison is thus also produced here in main() via a function call to "illumSimulate." It is

[4]This is possible if multiple design alternatives are being considered and each set of results are desired for the given CFS excitement conditions.

important to note, the BTDF must be known for the CFS under test, and be loaded. Both three-phase simulation executions, for those with L_v and E_v, the parsing of the results, and any possible calculation and output of control laws is handled in main.py on lines 390-427, or in Block 7 of the code graph in Figure D.1.

The final results therefore of a typical simulation are: measured input illuminance, E_v^t, measured exiting luminance, L_v^t, and some simulated luminous metrics, $i_m^t = V_{cfs} \cdot L_v^t$, $\forall\, t \in \{1, \cdots, t_{end}\}$ where t_{end} is the number of time steps associated with a certain simulation. Further, for CFS with known BTDFs used in validating the system, two additional outputs will be generated for each time step. The total outputs for a known BTDF CFS will be: measured input illuminance, E_v^t, measured exiting luminance, L_v^t, simulated luminous metric $i_m^t = V_{cfs} \cdot L_v^t$, then further, simulated exiting luminance, $L_{v,s}^t = T \cdot E_v^t$, and its corresponding simulated luminous metric, $i_{m,s}^t = V_{cfs} \cdot L_{v,s}^t$.

The Next Time Step: With the simulation step now complete, the CUBE 2.0 system as controlled by main() will wait until the start of the next time step. This time step interval is somewhat arbitrary, however, typical steps in building performance simulation are 15, 5, and 1 minute intervals [61]. In the majority of cases, the above steps are always completed before the next time step is ready to proceed. However, in low light situations (e.g. dawn and dusk) the EV varying $LDRI^{11}$ captured by the Canon 6D can take longer to acquire than the time step allows. In these circumstances, the next time step simply starts as soon as the above steps are complete. In main.py this code can be found on lines 216-240, or in Block 2 of the code graph in Figure D.1. The waiting code is found near the top, as before the simulation begins there is typically wait time between starting the program and the first time step. This allows for deploying the CUBE 2.0 for example the evening before a simulation and not needing to start the system by hand at times early in the morning.

Simulation Termination: If the simulation has indeed executed the required number of time steps and is ready to terminate, this simply involves terminating the socket connections, closing the open files, and exiting via a "sys.exit(0)" command. In main.py this code can be found on lines 429-432, or in Block 8 of the code graph in Figure D.1. Various configurations for termination have been implemented at different design stages. The current implementation simply involves the user cancelling the simulation via a "ctrl-c" command in the terminal at the end of the simulation. Given the timing variables which are maintained, other termination schemes can be easily implemented.

RPiCM()

Writing of RPiCM() commenced once the basic structure of main(), complete with "dummy functions" executing sleep and print statements, had been completed. Using the same basic structure of "start-up," "looping," and "shut-down" RPiCM() was too filled out with "dummy functions" for its needed tasks.

With the basic structure complete, development of RPiCM() needed to be coordinated with main() to ensure network communication would occur smoothly. Once the appropriate Socket connects allowing RPiCM() to act as a "server" were made, as stated above in the section 4.4.2 concerning main(), development continued with the addition of actual functionality.

Initially, declaration of several variables, instantiation of objects, and the definition of functions was completed. This can be seen in the code of RPiCM.py on lines 1-58, or in Block 0 of the code graph in Figure D.3.

Next, the actual camera configuration was completed. This involves changing various settings in order to get consistent measurements within the measurement cone of the optical fiber "measurement " ends. Note, a sleep call is also needed to ensure the camera unit has in fact changed its configuration before measurements can commence. These implementations can be seen in the code of RPiCM.py on lines 59-81, or in Block 1 of the code graph in Figure D.3.

The server functionality, that is waiting to hear from the MacBook Pro for notification to gather $LDRI^{j}_{RPi}$, can be seen in the code of RPiCM.py on lines 85-118, or in Block 2 of the code graph in Figure D.3.

PiCamera Module: The most prominent task for RPiCM() is taking the LDRI set, $LDRI^{j}_{RPi}$, of the optical fiber "measurement" ends inside the *measurement cone*. This is accomplished through the open source PiCamera module [115] and can be viewed in the code of RPiCM.py on lines 120-134 via the function call "HDRI", or in Block 3 of the code graph in Figure D.3.

Originally, it was proposed this LDRI set be processed locally into the exiting luminance distribution. Hence, development of a program written in C to process these photos was developed and compiled to run on the RPiCM when called by RPiCM() as a Subprocess. This program is called Cprog and is discussed below. After testing of network speeds it was determined sending the entire $LDRI^{j}_{RPi}$ to the MacBook Pro is far faster than processing it locally. As such, the processing program Cprog was recompiled to run on the MacBook Pro, thus it was nearly totally reused.

Knowing the RPiCM() doesn't need to process $LDRI^{j}_{RPi}$, the only job remaining is to wait and send $LDRI^{j}_{RPi}$ to the MacBook Pro when it requests it. This is accomplished in a "wait state" type architecture in RPiCM() and can be viewed in the code RPiCM.py on lines 136-167, or in Block 4 of the code graph in Figure D.3. After this, RPiCM() returns to a server state waiting to take another measurement. However, it should be noted, the MacBook Pro is not done with the RPiCM, rather, it uses an SFTP script to log into the RPiCM and download $LDRI^{j}_{RPi}$.

Summary: The basic functionality of RPiCM() can thus be summarized as a server, which upon query, takes a series of LDRI images, $LDRI^{j}_{RPi}$. When queried a second time then replies $LDRI^{j}_{RPi}$ is ready for transfer and returns to waiting to hear again to take an LDRI set. The main() uses a SFTP script to transfer the total LDRI set from the Raspberry Pi, hence the actual program RPiCM() running on the Raspberry Pi is not concerned with handling the LDRI set beyond its creation.

6D

A Shell script which is called by main() to execute the measurement of the input illuminance distribution using the Canon 6D with fisheye lens. The code first makes and changes to a directory for the current round. Then, it executes the C program fisheye6D to take and download 11 LDRI EV varying photographs, $LDRI^{11}$. Next, hdrgen is executed to process $LDRI^{11}$ into $HDRI_{input}$. Finally, fish2klems is executed which transforms $HDRI_{input}$ into the calibrated Klems Basis input illuminance vector, E_v.

fisheye6D

A custom made C program which uses the Canon API [27] to control the Canon 6D via USB. When executed, this program first declares the needed global variables and functions, seen on lines 1-242 in the code fisheye6D.c and in the graphical representation in Block 0 in Figure D.4. It then declares additional variables, initializes the EDSDK, and searches for and connects to the attached Canon 6D camera. In the code fisheye6D.c this is lines 244-294 and in the graphical representation is Block 1 in Figure D.4. Now, fisheye6D starts the camera session then proceeds to set the EC value of the Canon 6D to -5 and take a corresponding photograph, denoted $LDRI_{EC=-5}$. This is repeated for EC values from -4 to +5 at 1 stop increments, resulting in 11 total LDRI photographs, denoted $LDRI^{11}$. The camera session is then terminated. In the code fisheye6D.c this is lines 296-349 and in the graphical representation is Block 2 in Figure D.4. Using the Canon API, $LDRI^{11}$ is captured in minimal time, far faster than could be done by hand. Given "by hand" is considered acceptable in daylighting analysis, this is believed to be superior to standard practice. Starting a new camera session, $LDRI_{11}$ is downloaded to the execution directory, corresponding to the current time step of the simulation, on the MacBook Pro and deleted from the SD card in the Canon 6D. Ending the camera session, the program then terminates, all of which can be seen in the code fisheye6D.c on lines 350-384 and in the graphical representation in Block 3 in Figure D.4.

hdrgen

A freely available program which runs on UNIX which combines an appropriate set of LDRI photographs, $LDRI^{11}$, together to form an HDRI, $HDRI_{input}$. This combination is accomplished using a derived camera response function and is referred to as radiometric self calibration [40]. This program was not developed or extended for use by this thesis and is used purely as provided [14]. This program is the implementation of the function which shares its name, "hdrgen," described notationally in Equation 4.6 and graphically in Figure D.5. With respect to implementation details, the final resulting $HDRI_{input}$ will be saved in Radiance .hdr format with XYZE encoding [154].

fish2klems

A C program which transforms the fisheye HDRI, $HDRI_{input}$, into a calibrated Klems Basis input vector, E_v. It accomplishes this by reading in the newly formed $HDRI_{input}$ produced by hdrgen, along with the corresponding vertical illuminance measurement, ill_{LC}.

First, headers are imported, global variables are declared, and the needed functions are defined. This can be found in the source code fish2klems.c on lines 1-235 or in the graphical representation in Figure D.6 in Block 0.

Next, needed local variables in the main function are declared as can be seen in the source code fish2klems.c on lines 236-255 or in the graphical representation in Figure D.6 in Block 1.

Now the actual HDRI, $HDRI_{input}$, is read in from the file produced by hdrgen. This is accomplished in part with usage of prewritten functions available with the Radiance program. The parsing of the actual pixel values in CIE-Y luminance is done byte by byte via Equation 4.8 defined above. The pixel values of $HDRI_{input}$ as defined above in Equation 4.7 are now loaded into a double array with dimensions $x = 720$ and $y = 480$. At this point, the vertical illuminance value, ill_{LC}, which was captured to calibrate the $HDRI_{input}$ is converted from a string of chars input via a command line argument to a double. These steps are located in the source code fish2klems on lines 257-297 or in the graphical representation in Figure D.6 in Block 2.

With the pixel values $\mathcal{P}_h^{x \times y}$ and the vertical illuminance value ill_{LC} now loaded into memory, the $HDRI_{input}$ can be both corrected for vignetting and scaled to absolute SI units of $[\frac{cd}{m^2}]$. First, each pixel in $\mathcal{P}_h^{x \times y}$ is corrected for vignetting as specified in Equation 4.9. This is implemented as a simple element by element matrix multiple, where the construction of the matrix is quite involved and is the product of a Matlab program headerMaker6D. Next, each pixel is scaled by K_I for SI unit anchoring, where K_I is calculated via Equation 4.13 using ill_{LC} and ill_{HDRI} as calculated using Equation 4.12. Again, the Matlab program headerMaker6D is critical for making these calculations simple scaler multiplication and addition in this implementation. Due to function use, both vignetting and SI unit anchoring can be found in the source code fish2klems on lines 299-303 or in the graphical representation in Figure D.6 in Block 3.

Finally, the pixel values $\mathcal{P}_h^{x \times y}$ of the full hemispherical luminance input, $L(\theta, \phi)_{fish}$, can be binned and cosine corrected to form the Klems Basis input illuminance vector, E_v. As with vignetting and SI unit anchoring this is accomplished via scaler multiplication and addition with matrices created by headerMaker6D. Please note, for regular operation illuminance is calculated in this implementation of the Equation 4.23. This results in a vector 145 elements long which is saved as the input illuminate vector, E_v, which is saved. For calibration, however, input *luminance*, L_v^{in}, is calculated and is the output of fish2klemsC discussed below. Binning of pixels $\mathcal{P}_h^{x \times y}$ is implemented in the source code fish2klems on lines 305-341 or in the graphical representation in Figure D.6 in Block 4. Termination follows.

rpi

A Shell script called by main() which changes directories to the current round's directory, then calls Secure File Transfer Protocol (SFTP) using the script ssh2RPiCM to download the LDRI set, $LDRI_{RPi}^{j}$, generated by RPiCM() currently residing on the RPiCM. These simple tasks comprising five lines can be seen in the source code rpi very clearly.

ssh2RPiCM

A Shell script called by rpi through the Secure File Transfer Protocol (SFTP) to download the LDRI set, $LDRI_{RPi}^{j}$, generated by RPiCM() residing on the Raspberry Pi. As with rpi, the source code of ssh2RPiCM is very brief, including a simple directory change, "get" command to download $LDRI_{RPi}^{j}$, and termination.

proc

A Shell script called by main(), proc, first changes into the current round's directory. Then, it calls a C program, Cprog, to transform $LDRI_{RPi}^{j}$ gathered by RPiCM() on the RPiCM to the Klems Basis exiting vector, L_v. It then terminates. The source code of proc.

Cprog

A C program called by proc which processes the LDRI set, $LDRI_{RPi}^{j}$, downloaded from the Raspberry Pi to form an HDRI measurement. This HDRI measurement, similar to that generated by hdrgen, is taken inside the measurement cone and hence corresponds to the Klems Basis exiting vector, L_v.

The program begins by importing headers, declaring global variables, and defining the needed functions. This takes place in the source code of Cprog on lines 1-299 or in the graphical representation shown in Figure D.7 in Block 0.

Next, further needed variables are declared, after which the entire set of $LDRI_{RPi,e}$s produced by the RPiCM() is imported into memory, $LDRI_{RPi}^{j}$. In the actual implementation $j = 3$ is determined in the calibration section, chapter 5, to be sufficient over $j = 6$. Further, recall the RPiCM is equipped with a Bayer Pattern, hence each $LDRI_{RPi,e}, ee = \{1, 2, 3\}$ has dimensions $x = 648$ and $y = 486$. This implementation is found in the source code of Cprog on lines 300-324 or in the graphical representation shown in Figure D.7 in Block 1.

At this point, the center pixel location of the first optical fiber is located via a call to the function, OF as defined in Equation 4.27. With respect to implementation, this is simply a matrix with dimensions of 2×145 were $i_{of}^{n} = (1, n)$ and $j_{of}^{n} = (2, n)$. Construction this matrix, however, is non-trivial and is the topic of its own section in chapter 5, section 5.1.2. Further, it is the product of the Matlab program headerMakerRPiCM discussed below. With (i_{of}^{1}, j_{of}^{1}) known, Cprog then gathers the sets $\mathcal{OF}_1^1$, $\mathcal{OF}_1^2$, and $\mathcal{OF}_1^3$ as defined by Equation 4.28. Next, progressing from $\mathcal{OF}_1^1$ to $\mathcal{OF}_1^3$, the set without saturation in any channel is located by constructing the sets sat_1^1, sat_1^2, and sat_1^3, defined by Equation 4.29 and stopping when the set is in fact the empty set. With the empty set located, assuming it exists, here denoted sat_1^e, its corresponding set $\mathcal{OF}_1^e$ is converted to $\mathcal{OFY}_1^e$ as defined in Equation 4.31. Further, a metric defined in chapter 5 is calculated from $\mathcal{OFY}_1^e$ creating $\mathcal{OFM}_1^e$ as defined in Equation 4.32. Finally, implementing Equation 4.35, $\mathcal{OFM}_1^e$ is converted to a calibrated luminance measurement. As with the function OF, Equation 4.35 uses Equations 4.33 and 4.34 which are a product of the calibration process and further the Matlab program headerMakerRPiCM, discussed below. With L_v^1 now obtained from $LDRI_{RPi}^3$, the remaining 144 optical fiber measurements are gathered in the same manner. The final L_v is then saved before the program terminates. In the source code of Cprog these steps can be found on lines 325-341 with the usage of function calls or in the graphical representation shown in Figure D.7 in Block 2.

As with fish2klems, there exits two versions of Cprog: one for CUBE 2.0 system use and one for calibration purposes. The one used for calibration purposes is called CprogC and is discussed below.

fish2klemsC

As discussed at length in the calibration process of the CUBE 2.0 in chapter 5, the input luminance as measured by the Canon 6D is used to determine the functions 4.33 and 4.34 which are used in Equation 4.35 to convert the bit valued $\mathcal{OFM}_n^e$ to a measure of luminance.

As such, the program fish2klems has been modified to calculate the input luminance as opposed to the input illuminance required for regular operation. This simply involves removing the cosine term from Equation 4.23, resulting in Equation 5.5. Meaning the code is virtually identical, except in that calculation which can be found in fish2klemsC on lines 305-341 or in the graphical representation in Figure D.8 in Block 4.

CprogC

In addition to the input measurement requiring an update to fish2klems, the exiting luminance measurement also needs updating. This is because the actual metric used to calculate a single number for the optical fiber measurement, described by Equation 4.32, is unknown at the point of calibration. Hence, Cprog is extended to CprogC which outputs the entire set $\mathcal{OF}_n^e,\ e = 1, \cdots, 6,\ n = 1, \cdots, 145$.

Implementation wise, this simply involves writing to a file much more data than a single luminance measure for each optical fiber, but rather the entire set of BPAs for each optical fiber, for each LDRI. In the source code of CprogC these steps can be found on line 264 with the usage of a function call or in the graphical representation shown in Figure D.9 in Block 2.

headerMaker6D

In order to simplify the coding requirements in C for the program fish2klems and to speed up program execution for processing picture measurements, several matrices are precomputed in the form of header files. Examples of the precomputed variables include θ_e, ϕ_e, and r_e for all of the pixels. Klems Basis patch labels for each pixel are also precomputed.

These matrices are initially formed using a Matlab script, then processed into their final form using a Python script. While this technique is fast and works, there exist more efficient techniques which the author would now implement if recoding this functionality.

headerMakerRPiCM

In addition to the C program fish2klems having matrices encoded as headers precomputed, Cprog also has the need for precomputed headers. The most prominent example is in the encoding of the function OF to determine the position of the optical fiber ends. This formation is the topic of a special section in chapter 5 on the calibration of the CUBE 2.0 system. Again, the precomputed values are first derived using a Matlab script, then processed into their final form using a Python script.

Chapter 5

System Calibration and Operational Validation

In chapter 4, a detailed description of the design and construction process for the hardware-in-the-loop (HWiL) model of building daylighting systems, the *CUBE 2.0*, was presented. In this chapter, the calibration procedure for CUBE 2.0 will be overviewed. The theory behind the calibration method will be discussed, along with detailed accounts of the actual procedure used, as well as the calibration data sets. Also, testing of the system will be discussed which ensured the calibrated system operated as expected. Next, in chapter 6, the fully operational CUBE 2.0 is used with real *complex fenestration systems* (CFS) for daylong simulations, with results presented and analyzed.

5.1 System Calibration

The CUBE 2.0 system is an engineering research device with many different parameters which must be calibrated in order for the results to be meaningful. These calibration processes involve measuring the vignetting correction function, $v(\theta)$, confirming the values of solid angles used for pixels, $\omega(i,j)$, and the slopes and intercepts used for anchoring bits to SI units, expressed through the functions $\mathcal{S}$ and $\mathcal{I}$, amongst many others. Details of these various calibration processes involving both the input illuminance and exiting luminance measurements are presented below.

5.1.1 Input Illuminance Distribution Measurement

The calibration process begins by considering the input illuminance measurement. In the process outlined above in the CUBE 2.0 modeling and simulation section, several different parameters were defined in an abstract manner which need to be determined empirically in order for the CUBE 2.0 to work. These empirical values are determined below.

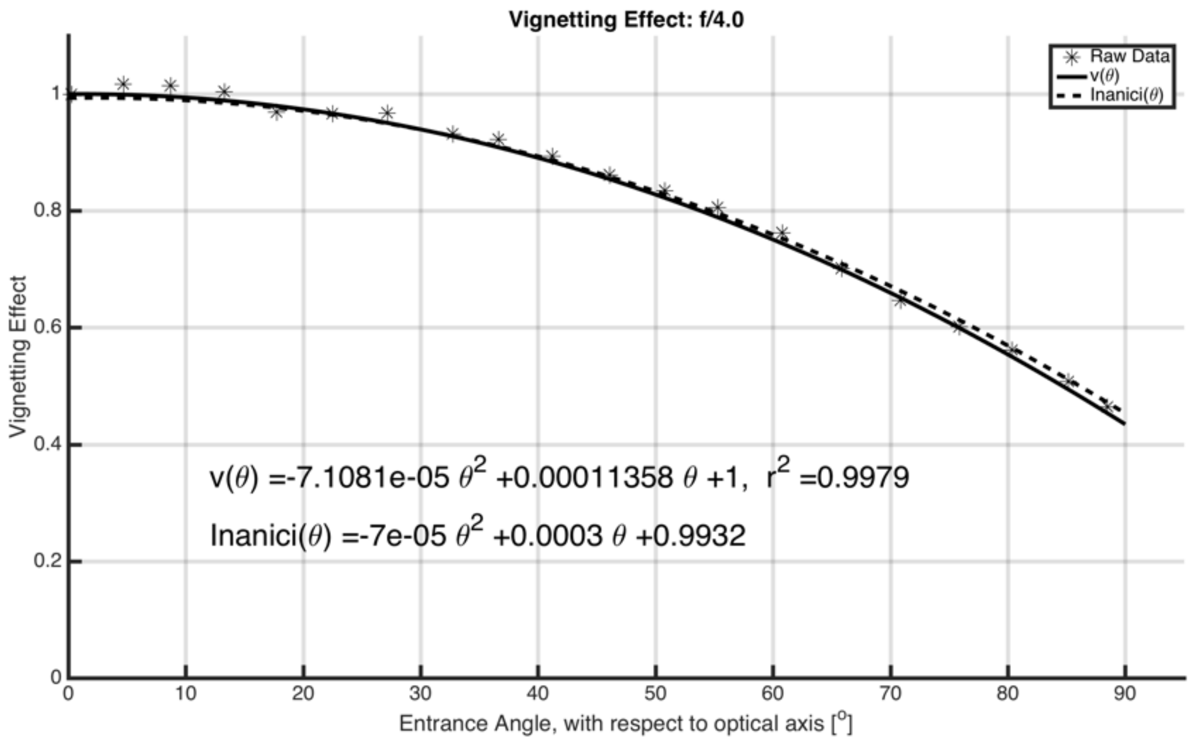


Figure 5.1: Vignetting correction for the Sigma 8mm F3.5 EX DG Fisheye lens.

Vignetting Effect Measurement

While the vignetting behavior of the Sigma 8mm f3.5 EX DF fisheye lens has been documented in the literature before [64], it was measured again as part of this thesis work. The results as a function of incident angle θ along side the existing measurement are presented in Figure 5.1. Here this correction was measured using an aperture of f/4.0, that which is used exclusively in this thesis work.

A circular fisheye lens projects the entire hemispherical view onto a subset of the sensing chip. This causes many pixels to report no meaningful information, hence they are "masked" and set to zero. The projection behavior of the lens is governed by a *projection function* discussed in the input illuminance measure section above. While the vignetting function is defined for incident angle, θ, when it comes to actual implementation, it will be done on a pixel level. Hence, each pixel has an effective incident angle, denoted θ_e, and calculated by Equation 4.15. Each pixel is corrected for vignetting, or masked if it's outside the hemispherical view, a process for which the scaler corrections are shown graphically for each pixel location in Figure 5.2.

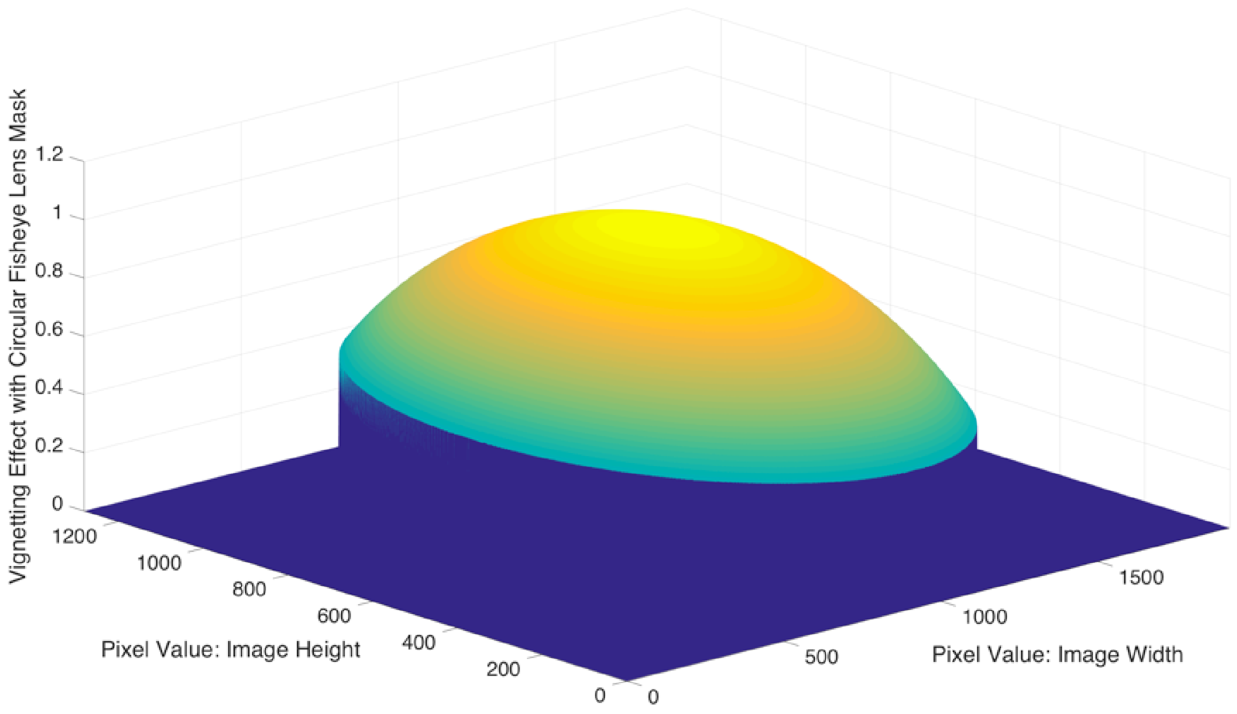


Figure 5.2: Vignetting correction for the Sigma 8mm F3.5 EX DG Fisheye lens, expressed as a matrix which is element wise multiplied with the $HDRI_{input}$. Note, the dimensions of the figure show a similar matrix used for a 1920×1280 pixel image, not the one used in the actual running of the CUBE 2.0 system, a 720×480 pixel image.

Measurement of the Fisheye Lens *Projection Formula:* r_{90}, f_{pixel}

The projection formula for the utilized solid-angle projection fisheye lens (Sigma 8mm F3.5 EX DG) is image formation abstract. This means the formula will work for all possible pixel dimensions, however, the parameters of the function, r and f, will have different values depending on the pixel sizes used.

In this work, the image formed by the Canon 6D has dimensions $x = 720$ and $y = 480$ pixels. As such, with a known θ in degrees and r in pixels, the focal length of the fisheye lens in units of pixels can be calculated. This is accomplished by aligning the Canon 6D on a jig and orienting a well defined target exactly at $\theta = 90°$, determining r, then using Equation 4.14 to calculate f_{pixel}. The setup utilized in in this exercise can be seen in Figure 5.3.

With this setup, an image is captured and analyzed to determine r_{90} of the target at 90°. First, the pixel associated with the target is located, (i, j), from which effective coordinates of the target are calculated, $(i_e = i - i_c, j_e = j - j_c)$, where (i_c, j_c) is the center of the fisheye image in pixels. These effective pixel coordinates are then used to calculate the effective radius by, $r_{90} = r_e = \sqrt{i_e^2 + j_e^2}$. Finally, using the projection formula shown in Equation

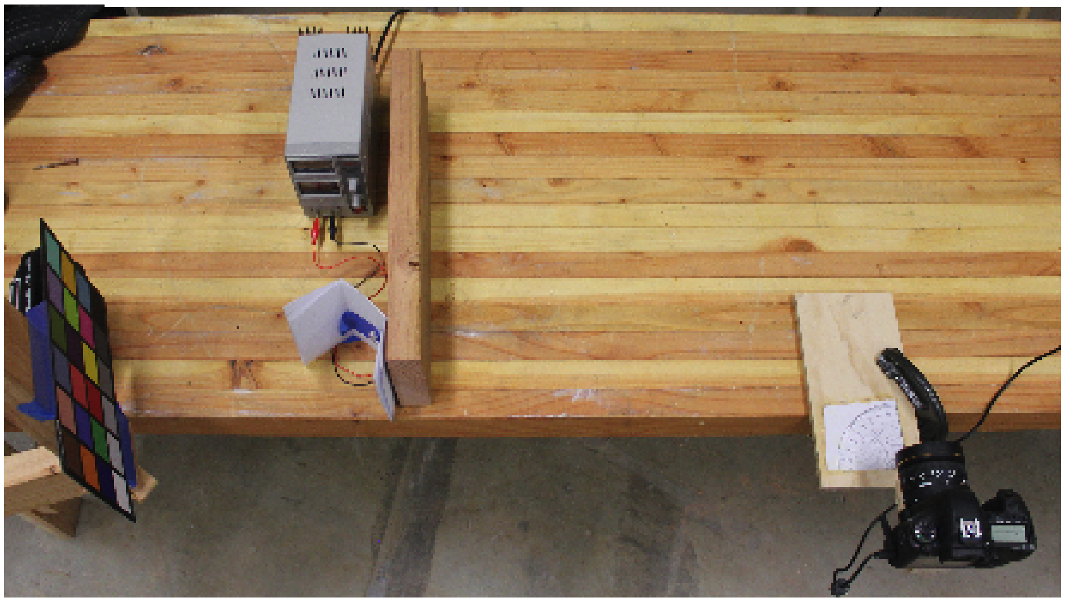


Figure 5.3: Measuring f_{pixel} of the Sigma 8mm F3.5 EX DG Fisheye lens.

4.14, the focal length in pixels is solved for by,

$$f_{pixel} = \frac{r_{90}}{2tan(\frac{90}{2})}. \tag{5.1}$$

The actual implementation of this process was accomplished using the vignetting measurement data and can be found in the program headerMaker6D.m.

f_{pixel} can now be used in Equation 4.15 to assign effective theta values, θ_e, to the pixels in an image. An example of this is shown in Figure 5.2 where the θ_e values of the pixels are needed in order to correct for vignetting.

Pixel Solid Angle Value: $\omega(i, j)$

As a check, the total summation of the solid angle over all pixels should total 2π $[sr]$ as this is the number of steradians in a hemisphere. When checked for the base image used in this work, $x = 720$, $y = 480$, the agreement,

$$2\pi = \sum_{pixels} \omega(i, j), \tag{5.2}$$

was accurate to 14 digits. Implementation details of this process can be explored by examining the code in headerMaker6D.m.

5.1.2 Exiting Luminance Distribution Measurement

The calibration process continues by considering the exiting luminance measurement. In the process outlined above in the CUBE 2.0 modeling and simulation section, several different parameters were defined in an abstract manner which need to determined empirically in order for the CUBE 2.0 to work. These empirical values are determined below.

LDRI Set Capture

The actual LDRI shutter speeds are unknown at design time, hence calibration data define their values within the CUBE 2.0 system. Reasoning is presented in the design section advising nominal shutter speeds of 10^6, 10^5, 10^4, 10^3, 10^2, 10^1 be used. As a final implementation, the actual shutter speeds as reported by the Python PiCamera Module are respectively 760 568, 99 980, 9 995, 987, 81, and 12 [μs]. The reason for the inability to set exact shutter speeds is unknown, as these shutter speeds are the reported values after the respective nominal shutter speeds were set. While not exactly what is sought, the shutter speeds can be set consistently and are roughly one order of magnitude different from each other, hence are deemed acceptable for the CUBE 2.0 system.

Color Space Transformation

The construction of the mapping, $\mathbb{M}$, is known as spectral characterization and a least squares regression methodology is used [153]. For training data, the RPiCM was subjected to a monochrome beam of light ranging from 380 to 700 [nm] in increments of 10 [nm]. This was accomplished using a Bentham/IVT PVE300 spectral response system [18] shining directly onto the CMOS chip as opposed to the target method discussed in the document, "Graphic Technology and Photography" [66], as this system is typically used for solar cell characterization and was ready for use. This saved the effort of setting up an optical bench with the needed equipment and the same monochrome light was achieved, thus the authors believe this is a valid alternative.

Due to a low energy level of the monochrome light source, a shutter speed of 450 ms was used, as this is the integration time of the built-in silicon photodiode detector used by the system. Correct shutter speed is crucial, as too short an exposure will result in no meaningful signal (i.e. noise), and too long an exposure will saturate the 10-bit channels. It should be noted that the shutter speed of the RPiCM is limited by the "frame rate" parameter, and thus "frame rate" must be adjusted downward from the default (i.e. 30 fps to 2 fps) to use a 450 ms exposure. This value was chosen by examining the highest level of energy monochrome wavelength and ensuring it did not saturate the 10-bit sensor output. If the highest energy wavelength monochrome light did not saturate the sensor, then none of the monochrome wavelengths will do so.

The normalized results of the RPiCM response are presented along with the CIE colour matching functions in Figure 5.4. Notice the reasonable qualitative agreement between the two sets of curves. This overall agreement is expected, because the RPiCM produces

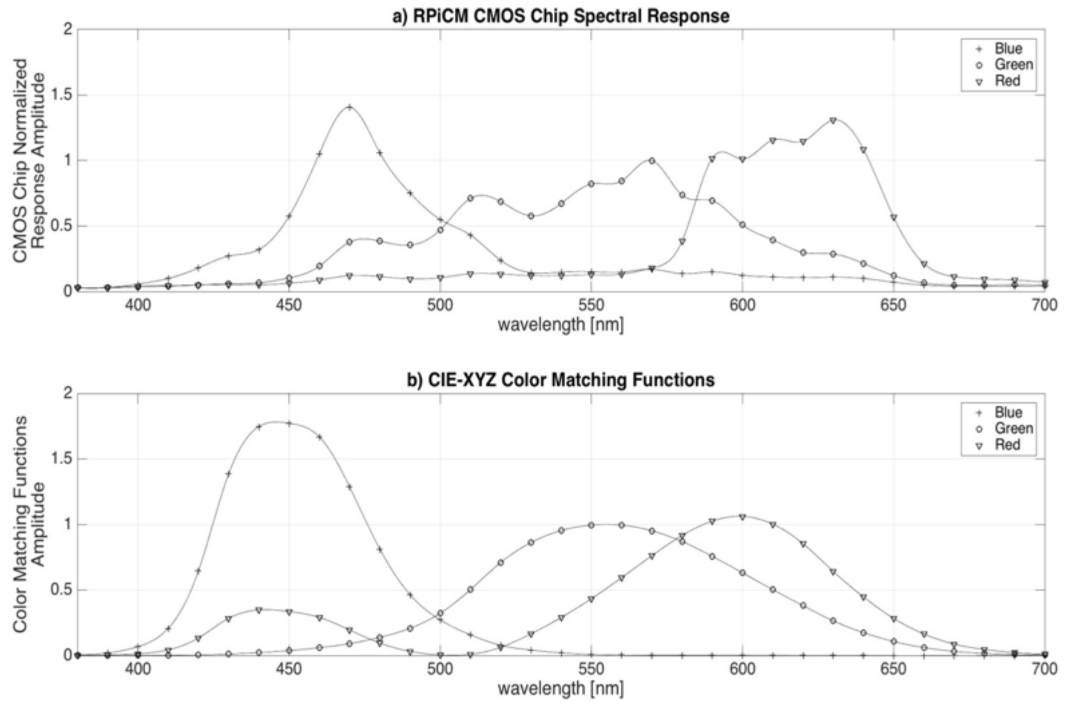


Figure 5.4: The RPiCM spectral response data (a) and the CIE-XYZ standard (b) expressed at 10 [nm] resolution.

photographs similar to how the scene appears to the human eye, which is the purpose of the CIE colour matching functions. Therefore, the response is deemed acceptable and used as is for the luminance measurements in this study, i.e. no additional filters were placed on the CMOS chip. This is favorable because it eliminates the filter addition at the semi-conductor manufacturing level, which requires greater effort and that leads to higher costs.

The above analysis was completed for a single set of four pixels comprising a unique BPA. Due to the high consistency in the silicon semiconductor industry, it is assumed that each BPA will respond spectrally in an identical manner. Thus, the above results are used for all the pixels in the camera image. Note that this response is independent of the amount of light striking the sensor, rather, it is the result of the spectral content of the light. During the analysis, all spectral responses of the light were normalized for spectral light energy to that of the used monochrome beam of light.

The specific RPiCM used in this work has the BPA data, RGB_λ as a 3×33 matrix, with each column containing the RAW red, green, and blue values produced by the CMOS chip under investigation when excited by monochrome light of 380 to 700 [nm] wavelength (λ) light at 10 nm intervals. Further, XYZ_λ is a 3×33 matrix, with each column containing the X, Y, and Z values as defined by the CIE colour matching functions at the same wavelengths.

These matrices are related, with the form of $\mathbb{M}$ assumed here, and this relation can be expressed formally as,

$$XYZ_{\lambda} = \mathbb{M}RGB_{\lambda}. \tag{5.3}$$

Note, RGB_{λ} and XYZ_{λ}, are the training data, and further, using a linear least squares regression technique (note in the following superscripts T and -1 denote transpose and inverse, respectively), $\mathbb{M}$ can be expressed as,

$$\mathbb{M} = \left[XYZ_{\lambda}RGB_{\lambda}\right] \cdot \left[RGB_{\lambda}RGB_{\lambda}^{T}\right]^{-1}. \tag{5.4}$$

The linear mapping $\mathbb{M}$ thus allows for the transformation of individual BPA measurements, pixels in a $LDRI_{RPi}$ of arbitrary field of view from the device-specific colour space, RGB_{RPi}, to the standardized colour space XYZ expressed in a wavelength discretized form as XYZ_{λ}. With respect to the formalisms presented in the design section, this $\mathbb{M}$ is the one used to transform the set $\mathcal{OF}_{n}^{e}$ defined by Equation 4.28 into the set $\mathcal{OFY}_{n}^{e}$ defined by Equation 4.31.

Individual Optical Fiber HDRI Formation

Empirical investigation has shown this subset to be 12 by 12 BPAs in size, meaning 144 BPAs must be processed per optical fiber. In the actual development of the CUBE 2.0 system, this term was left unspecified, but for clarity it is stated explicitly here. An example of this 144 BPA set can be seen in Figure 5.5. Hence, over the entire field of view 144×145 =20 800 BPAs need to analyzed, in contrast to the 1 259 712 BPAs in the total $LDRI_{RPi}$. This is a reduction in processing of over 98%[1].

While the size of the BPA array used to capture the end of the optical fiber was specified in the modeling section, it was only solved for explicitly once the CUBE 2.0 system was constructed and calibration began. A visual depiction of an arbitrary pixel set $\mathcal{OF}_{n}^{e}$ can be seen in Figure 5.5.

Calibration Parameter Calculation

How to Calibrate CUBE 2.0: In a previous study [104], this same CMOS chip was used to make full field of view HDRI measurements. To make this final transformation, the chip was exposed to a well known luminance source, an open port of an integrating sphere, and had many pictures taken at various luminance levels. The process was started from a zero luminance exposure and gradually increased in steps to over 25 000 $[\frac{cd}{m^2}]$. At the various steps exposure sets were generated, $LDRI_{RPi}^{4}$, where the nominal shutter speeds were $t = 10^4, t = 10^3, t = 10^2, t = 10$ $[\mu s]$. Roughly speaking, from 0 to 120 $[\frac{cd}{m^2}]$ the $t = 10^4$ $[\mu s]$ exposure was active, from 120 to 1800 $[\frac{cd}{m^2}]$ the $t = 10^2$ $[\mu s]$ exposure was active, from 1800 to 17 000 $[\frac{cd}{m^2}]$ the $t = 10^2$ $[\mu s]$ exposure was active and finally from 17 000 to the maximum available of 25 000 $[\frac{cd}{m^2}]$ the $t = 10$ $[\mu s]$ exposure was active. For each exposure,

[1]Due to simpler coding and little overhead, the subset is actually 13 by 13 BPAs in size.

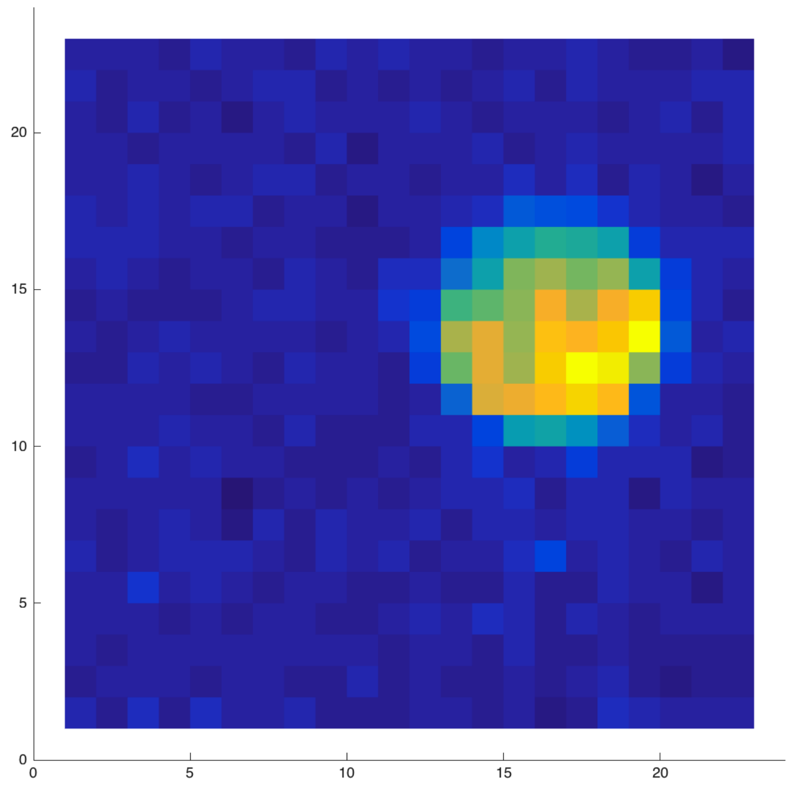


Figure 5.5: The plot of an optical fiber end as captured by the RPiCM. Note, this is actually the Bayer Pattern data, reduce the dimensions by a factor of two to get the actual $LDRI_{RPi,e}$ set.

the set of excitation values was matched up with the color space transformed BPA values. These values were then regressed using simple least squares regression to determine both the slopes and intercepts of the transformations from the arbitrary bit values to those of luminance $[\frac{cd}{m^2}]$. Examining Figure 5.6, the data from that process is shown.

With the above process in mind, mimicking this for the CUBE 2.0 calibration will require some known source of excitation. This known source of excitation will be inserted into the CUBE 2.0 through the CFS test aperture to excite the optical fiber "sampling ends," which in turn excite the "measurement ends," which finally are measured by the RPiCM to create the set $LDRI^6_{RPi}$. While an integrating sphere, like that used in the previous study, produces a controlled and adjustable luminance excitation like that required, they are expensive instruments[2] and none was available for use.

An alternative idea involves using the sun itself as a calibration source, with motivation

[2]The study [104] used an integrating sphere at the National Metrology Centre in Singapore for a price of $800 USD for three hours.

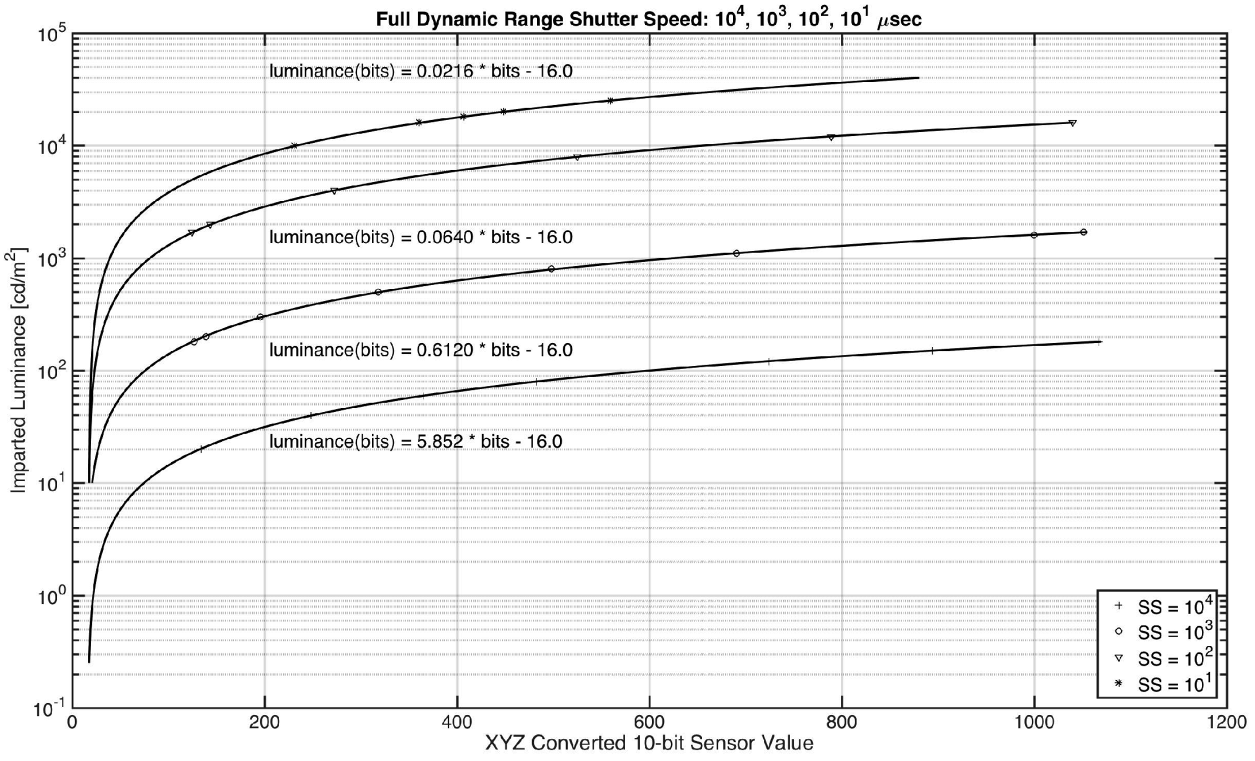


Figure 5.6: Transformation from arbitrary bit values to SI units of luminance $[\frac{cd}{m^2}]$

including it is the real excitation, and the input value is already being measured via the Canon 6D and fisheye lens[3]. Hence, by exposing the CUBE 2.0 to sufficiently large luminance excitations with no CFS specimen covering the CFS test aperture, a set of known luminance and associated $\mathcal{OFM}_n^e$ responses can be generated. This set, if sufficiently large and diverse to cover enough of the exposures $LDRI_{RPi,e} \in LDRI_{RPi}^6$, can then be used to calculate the slopes and intercepts (i.e. define the functions $\mathcal{S}$ (Equation 4.33) and $\mathcal{C}$ (Equation 4.34) needed to transform the arbitrary bit values to SI units of luminance $[\frac{cd}{m^2}]$.

As discussed above in chapter 4 with respect to the CUBE 2.0 design analysis, the Canon 6D is not aligned perfectly with the CFS test aperture. Thus, one could question the input measurement generated by the Canon 6D, and wonder if it is the true input which is exciting the ends of each respective optical fibers. To address this valid concern, one must look to the input sensitivity analysis which was conducted concerning this question. It was learned, both analytically through the use of Radiance models and experimentally through actual photographic measurements, for light sources which are far away from the CUBE 2.0, the Canon 6D and the CFS test aperture are in fact being excited by virtually identical luminance distributions (i.e. $L_{fish}(\theta, \phi) = L_{CFS}(\theta, \phi)$). This analysis is overview in the

[3]Technically the value calculated above in Equation 4.23 is illuminance, but luminance can also be calculated by simply removing the $cos()$ term, see Equation 5.5 below.

CUBE 2.0 design section, with the full data sets and analysis presented in Appendix F.

Sun Calibration Process Formalized: Within the context of the given formalisms developed so far, the proposed calibration of the exiting luminance measurement is now expressed. Beginning with a modification of Equation 4.23, the input luminance to CUBE 2.0 can be expressed as,

$$L_v^{in,n} = \frac{\sum_{K_n} HDRI_{input}(i,j)\omega(i,j)}{\sum_{K_n} \omega(i,j)}. \tag{5.5}$$

Here, $L_v^{in,n}$ is equivalent to E_v^n, however, without the cosine term, meaning it corresponds to the luminance which passes through the CFS test aperture and strikes the corresponding optical fiber "sampling end" within the acrylic hemisphere. Notice, the $L_v^{in,n}$ value has calibrated SI units of $[\frac{cd}{m^2}]$, meaning the exiting luminance exciting the respective optical fiber is now known at a given time, t. Further, this luminance input to the optical fibers is produced for each optical fiber, $n = \{1, \cdots, 145\}$. This altered measurement of the input *luminance* instead of *illuminance* is the product of the cyber component fish2klemsC and not fish2klems which is used in regular operation.

In parallel, that is at the same time, the RPiCM is used, as stated above, to take a measurement within the measurement cone of the optical fiber ends. This measurement, as shown above, will produce an arbitrary bit value, $\mathcal{OFM}_n^e$, in one of six $LDRI_{RPi,e}$ exposures, which ever is capturing the proper dynamic range in which the excitation is a part, for all optical fibers, $n = \{1, \cdots, 145\}$. It should be noted, however, the metric used to calculate $\mathcal{OFM}_n^e$ is unknown before the calibration process is executed. As such, an alternative program to Cprog for processing the $LDRI_{RPi}^6$ is used for calibration, CprogC. The output of this program is in fact the full set of BPAs associated with optical fiber n,

$$\mathcal{OFYC}_n \;=\; \{\mathcal{OFY}_n^1, \mathcal{OFY}_n^2, \ldots, \mathcal{OFY}_n^6\}. \tag{5.6}$$

From these BPA values, various metrics will be explored to determine which is appropriate. To continue with the explanation of the spirit of the calibration process, however, it is assumed a representative metric, like that defined in Equation 4.32, can indeed be calculated from the properly exposed $\mathcal{OFY}_n^e$ within the set $\mathcal{OFYC}_n$, for all 145 optical fiber ends.

As such, these two measurements, $calPt_n^t = (L_v^{in,n}, \mathcal{OFM}_n^e)$, are produced for each optical fiber, $n = \{1, \cdots, 145\}$, at each round, t. Thus, if a sufficiently large and representative set of $calPt_n^t$s is collected, they can aggregated into their respective exposures, e, and used to estimate the corresponding *slope* and *intercept* using a linear least squares framework, respectively defined here as the functions $\mathcal{S}$ and $\mathcal{I}$. Technically speaking, because the RPiCM is well modeled by a linear function, only two $calPt_n^t$ data points are needed per exposure, e. However, in real engineering systems, redundancy is desired, hence dozens of points from multiple data collection runs are sought.

With an understanding of dynamic ranges of the RPiCM, one will know that lower $L_v^{in,n}$ values will correspond to $e = 1$ exposures on the RPiCM, with respective increases in $L_v^{in,n}$

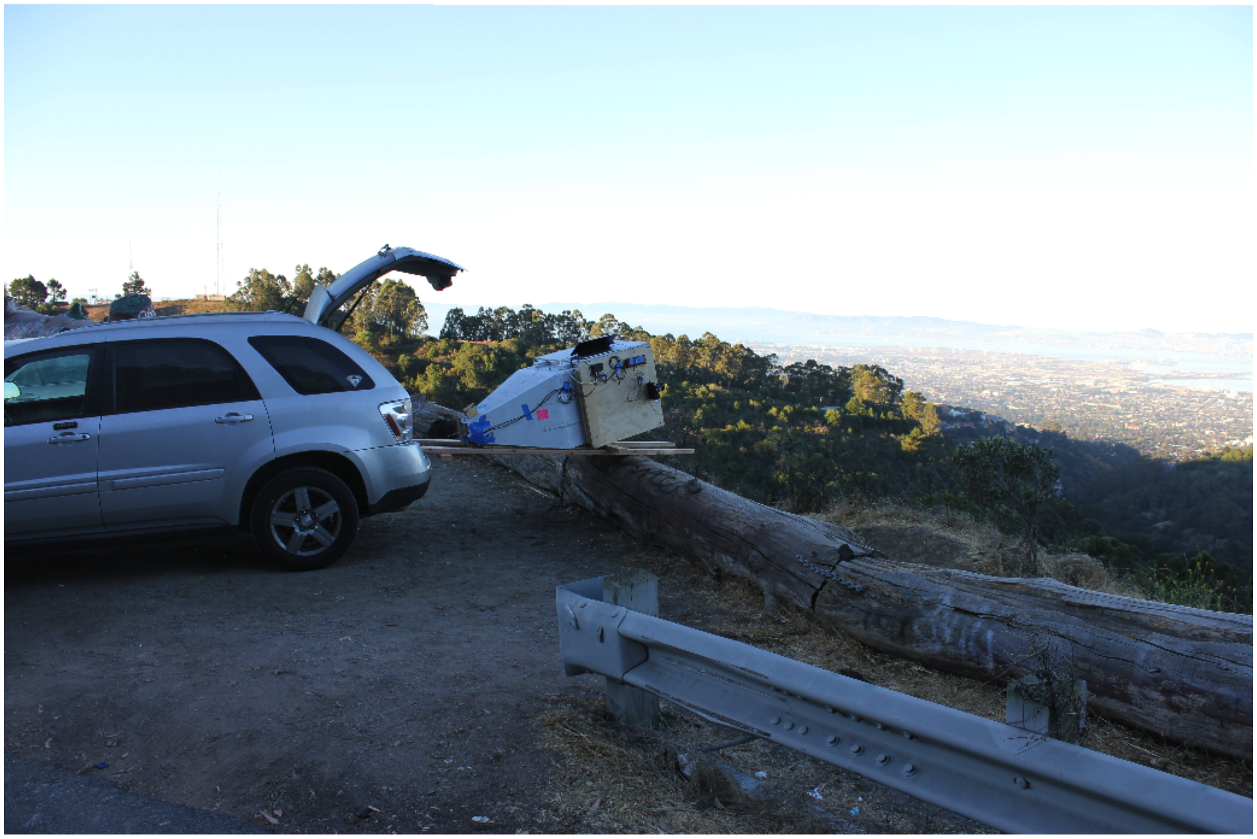


Figure 5.7: The CUBE 2.0 system deployed at Tilden Park for calibration data collection.

corresponding to larget and larger e values. Thus, the set of $calPt_n^t$ should be collected over a large span of input luminances from low to high.

Data Calibration Collection Deployment: With the above calibration technique in mind and knowing the sky vault must shine on the optical fibers in order to produce meaningful $calPt_n^t$ data points, conditions were sought where the sky vault (i.e. an infinite distance source) could be used to excite the CUBE 2.0, yet the remaining hemispherical view has a relatively low luminance value and can be considered to be of negligible excitement in its geometrically non-correct excitement of the optical fibers with views of the sky (i.e. it is not infinity far away as the sky vault is). Given this thesis work was conducted primarily in Berkeley, California, the surrounding region was considered. In this search, the famed Tilden Park was identified to have several potential locations, with one finally decided upon. This location, a pull off of an access road, has large views of the sky, with mostly forested hills (i.e. dark green, thus low luminance) making up the remaining hemispherical view. Figure 5.7 has a picture of the CUBE 2.0 system deployed at that location for one of several calibration data gathering deployments.

During the calibration data collection deployments, the CUBE 2.0 was set up with as much as possible of its input Klems Basis Patches viewing the sky vault. Meaning the most possible optical fibers could be calibrated at one time. This involved tilting the CUBE 2.0 back from plane. Further, all the patches need to be excited, hence the CUBE 2.0 was set on its side allowing for a complete half of the inputs to be excited by the sky vault. Thus, in two collection periods, the entire input hemisphere with respect to the Klems Basis patches can be excited. See Figure 5.8 for a view of the input measurement in picture form (i.e. not "binned" into the 145 Klems Basis Patches).

Data collection rounds ideally have a distribution of low to high luminance inputs per Klems Basis Patch, corresponding to data points which span the respective $LDRI_{RPi,1}, \cdots,$ $LDRI_{RPi,6}$, as justified above in the formalization section. This allows for more data points to be used in the estimation of $\mathcal{S}$ and $\mathcal{I}$. The most optimal time to collect these data points is either at sunrise or sunset, where the sky vault luminance changes most dramatically in the shortest period of time. For this calibration, sunrise times were used in order to ensure the fewest interaction with other park patrons. Few people are in Tilden Park at the calibration location before 8:00 am, however, many are located there to view the sunset. Further, no direct solar component can be in the calibration data, hence this being a west facing location, sunrise was the preferred choice.

The actual calibration deployments consisted of eight sunrises in the year 2016: 29 September, 6, 11, and 20 October, and 1, 2, 3, and 5 November. Table **??** below shows the number of rounds per deployment, as well as the specific optical fibers which were excited for those rounds. For 29 September this corresponds to $\mathcal{T}_1$ and $\mathcal{K}_1$, respectively. It should be noted that half the input hemisphere was excited for each deployment, however, the numbering of Klems Basis Patches is circular around the zenith patch, hence the patch numbers appear to be random without viewing in context of the labeling system.

With the calibration data sessions well defined with respect to number of rounds as well as the specific Klems Basis Patches which are excited in those rounds, attention can now be put to formally defining the calibration data. Examining the definition of $calPt_n^t$, one sees that each t and n are defined as $t \in \mathcal{T}_r$ and $n \in \mathcal{K}_r$ for $r = \{1, \cdots, 8\}$. Meaning the total number of calibration points used is,

$$\sum_{r=1}^{8} |\mathcal{T}_r| \cdot |\mathcal{S}_r| = 202,577. \tag{5.7}$$

Further, when actually examining the data, rarely does luminance exiting the optical fiber ends saturate an RPiCM generated $LDRI_{RPi,3}$, with shutter speed $10^4[\mu s]$. Hence, $j = 6$ is reduced to $j = 3$, making the new set of LDRIs which are processed $LDRI_{RPi}^3$ as opposed to the original definition above, $LDRI_{RPi}^6$.

With all the data now collected, attention is turned to aggregating the data via optical fiber and $LDRI_{RPi,i}$ exposure. With the reduction of j to 3, each optical fiber will now have 3 slopes and 3 intercepts which will need to be estimated. Notationally, $calPt_n^t|_e$ refers to the $LDRI_{RPi,e}$ from which the encapsulated $\mathcal{OFM}_n^e$ has been calculated (i.e. the superscript e). In plain English, this is the $LDRI_{RPi}^e$ which is properly exposed for optical fiber n during

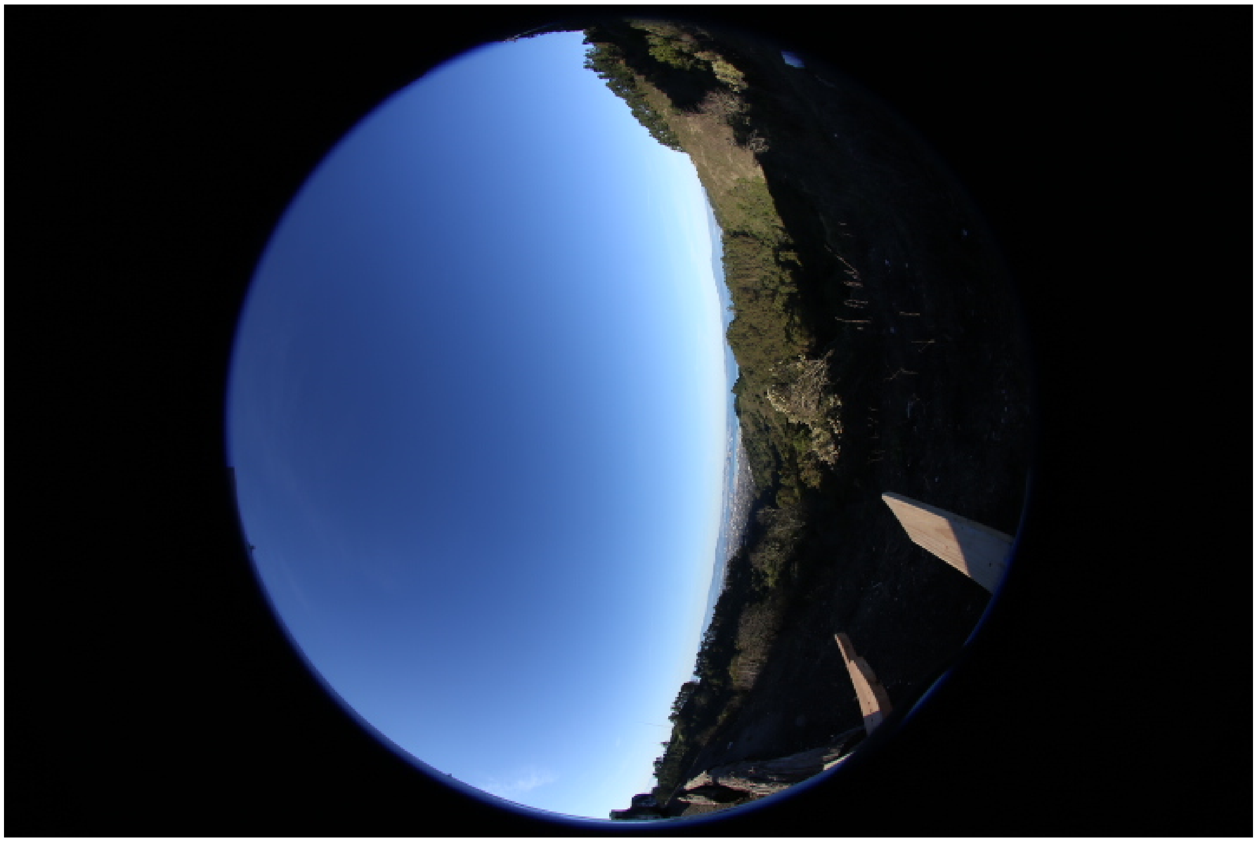


Figure 5.8: The input measurement as taken during a calibration deployment at Tilden Park. Notice the large highly uniform sky luminance and corresponding low luminance from the forested hills.

round t. Thus, for each $n = \{1, \cdots, 145\}$, these slopes and intercepts will be estimated from the three respective sets,

$$D1^n = \{calPt_n^t \ |(t,n) \in (\mathcal{T}_1, \mathcal{K}_1) \cup \cdots \cup (\mathcal{T}_8, \mathcal{K}_8) \ \& \ calPt_n^t|_e == 1 \ \}, \tag{5.8}$$

$$D2^n = \{calPt_n^t \ |(t,n) \in (\mathcal{T}_1, \mathcal{K}_1) \cup \cdots \cup (\mathcal{T}_8, \mathcal{K}_8) \ \& \ calPt_n^t|_e == 2 \ \}, \tag{5.9}$$

and

$$D3^n = \{calPt_n^t \ |(t,n) \in (\mathcal{T}_1, \mathcal{K}_1) \cup \cdots \cup (\mathcal{T}_8, \mathcal{K}_8) \ \& \ calPt_n^t|_e == 3 \ \}. \tag{5.10}$$

Considered in English, these respective sets can be thought of as follows. For each respective optical fiber n, the set $D1^n$ contains all the $calPt_n^t$ from each of the eight calibration data sets that are properly exposed with a shutter speed of 10^6 $[\mu s]$. Similarly, $D2^n$ and $D3^n$ contain the respective $calPt_n^t$ which are properly exposed with a shutter speed of 10^5 $[\mu s]$ and 10^4 $[\mu s]$ for optical fiber n.

Table 5.1: Summary of the calibration rounds for CUBE 2.0.

Round	**Date**	**Total Rounds in Deployment**	**Klems Patches Excited**
1	29 Sep.	$\mathcal{T}_1 = [1, \cdots, 240] \subset \mathbb{N}$	$\mathcal{K}_1 = \{1:48, 56:71, 81:94, 105:118, 126:133, 141:145\}; \lvert\mathcal{K}_1\rvert = 105$
2	6 Oct.	$\mathcal{T}_2 = [1, \cdots, 320] \subset \mathbb{N}$	$\mathcal{K}_2 = \{1:12, 15:37, 45:58, 70:82, 94:106, 119:125, 135, 137:139\}; \lvert\mathcal{K}_2\rvert = 86$
3	11 Oct.	$\mathcal{T}_3 = [1, \cdots, 296] \subset \mathbb{N}$	$\mathcal{K}_3 = \{1:28, 30:33, 40:53, 63:76, 88:100, 112:121, 131:136, 144:145\}; \lvert\mathcal{K}_3\rvert = 91$
4	20 Oct.	$\mathcal{T}_4 = [1, \cdots, 301] \subset \mathbb{N}$	$\mathcal{K}_4 = \{1, 3:9, 14:22, 31:41, 52:64, 76:85, 101:112, 116, 123:129, 139:142\}; \lvert\mathcal{K}_4\rvert = 75$
5	1 Nov.	$\mathcal{T}_5 = [1, \cdots, 345] \subset \mathbb{N}$	$\mathcal{K}_5 = \{1:9, 12:24, 31:42, 52:65, 77:89, 101:112, 123:130\}; \lvert\mathcal{K}_5\rvert = 81$
6	2 Nov.	$\mathcal{T}_6 = [1, \cdots, 303] \subset \mathbb{N}$	$\mathcal{K}_6 = \{1:9, 12:23, 30:42, 52:65, 76:85, 101:112, 123:129, 139:142\}; \lvert\mathcal{K}_6\rvert = 81$
7	3 Nov.	$\mathcal{T}_7 = [1, \cdots, 317] \subset \mathbb{N}$	$\mathcal{K}_7 = \{1:16, 20:26, 28:32, 40:53, 63:71, 73:76, 88:100, 112:121, 131:136, 144:145\}; \lvert\mathcal{K}_7\rvert = 86$
8	5 Nov.	$\mathcal{T}_8 = [1, \cdots, 271] \subset \mathbb{N}$	$\mathcal{K}_8 = \{1:2, 6:10, 18:26, 36:46, 58:70, 82:93, 107:117, 127:133, 141:145\}; \lvert\mathcal{K}_8\rvert = 76$

This estimation is completed via a function, denoted LSQ, whose domain is a set, Di^n, of $calPt_n^t$ values and whose range is a 2-tupel of reals, which are the slope and intercept which, in the sense of linear least squares, best fits the data, where,

$$LSQ : De^n \to [\mathbb{R}, \mathbb{R}]. \tag{5.11}$$

Hence, the functions $\mathcal{S}$ and $\mathcal{I}$ are defined as,

$$\mathcal{S}(n,1) = LSQ(D1^n)_1 \quad , \quad \mathcal{I}(n,1) = LSQ(D1^n)_2 \tag{5.12}$$

$$\mathcal{S}(n,2) = LSQ(D2^n)_1 \quad , \quad \mathcal{I}(n,2) = LSQ(D2^n)_2, \tag{5.13}$$

and

$$\mathcal{S}(n,3) = LSQ(D3^n)_1 \quad , \quad \mathcal{I}(n,3) = LSQ(D3^n)_2, \tag{5.14}$$

for $LDRI_{RPi,1}$, $LDRI_{RPi,2}$, and, $LDRI_{RPi,3}$, which is repeated for all the optical fiber ends.

Optical Fiber End Metric: With the total calibration process now outlined, attention can be focused on what metric will be used to process the set $\mathcal{OFY}_n^e$ into a single number $\mathcal{OFM}_n^e$ as specified in Equation 4.32. While many different metrics were explored in the data processing stage, a rather simple metric was settled upon. For some properly exposed $\mathcal{OFY}_n^e$ for an arbitrary optical fiber n, its definition is,

$$\mathcal{OFM}_n^e = mean(max(\mathcal{OFY}_n^e, 10)), \tag{5.15}$$

where $max(\cdot, 10)$, returns the ten largest values of the input set and $mean(\cdot)$ is the simple arithmetic average. This same metric is applied to all properly exposed $\mathcal{OFY}_n^e$ (i.e. $|sat_n^e| = 0$, where sat_n^e is defined by Equation 4.29 and e is the lowest value possible) regardless of exposure or optical fiber. Meaning, this same metric works in all cases of optical fiber and for all exposures.

Calibration Results and Discussion: As a visual representation of this data and the fitting process, the graph of an arbitrary Klems Basis Patch, patch 13, is shown in Figure 5.9. As expected, a linear model[4] is a good approximation of the optical fiber and RPiCM system, hence the process is repeated for all Klems Basis Patches, $n = 1, \ldots, 145$. In total, for the first exposure, all but two Klems Basis Patches in $LDRI_{RPi,1}$ have an r^2 value greater than 0.89, with the vast majority having r^2 values greater than 0.95. For the two Klems Basis Patches below 0.89 (116, 126), they are still obviously visually linear when their data is plotted, with respective r^2 values of 0.79 and 0.82. However, their data has more spread, perhaps due to Klems Basis Patch rounding of clouds in the inputs, or geometrical based error in the excitement from non-sky vault viewing areas of the input. It is thought with more calibration data, these estimates would improve, yet they are used here as the vast majority of Klems Basis Patches are highly confident in their estimates. The plot of one of Klems Basis Patch 116 is shown in Figure 5.10 to illustrate this point.

Beyond the first exposure, $LDRI_{RPi,2}$ shows similarly good behaviour as captured by a linear model. Of 145 Basis Klems Patches, 131 show r^2 values greater than 0.89, with the remaining 14 having similar behavior to patch 116 and 126 for the first exposure. Further, it should be noted, for the last two theta rings (i.e. Klems Basis Patches 118-145), little luminance gets through due to the large incident angle. Hence, much of these lower r^2 values are simply due to the lack of calibration data at the larger incident angles. For example, Patch 134 shown in Figure 5.11 is clearly linear and similar behaviorally to patches 13 and 116, there is just less calibration data, hence the relative spread is larger and the estimate less confident.

As shown via Table 5.1 and Equation 5.7, the data for this calibration process is quite voluminous. The 200,000 plus data points have been synthesized from approximately 400 gigabytes of raw CUBE 2.0 data. As stated earlier, the outputs needed for calibration as somewhat different than when actually running an experiment, hence fish2klemsC and CprogC were developed to output the appropriate data. Once this data is produced, all 400 gigabytes, it must be processed to actually calculate the needed metrics and least squares regressions. As such, this job fell to a set of Shell, Python, and Matlab scripts. While technically rather straight forward, processing this much data became quite burdensome and is beyond the purview of this thesis as the actual formalisms of what the code is doing were just presented above.

To produce this data, the CUBE 2.0 code which processes the set $LDRI_{RPi}^3$ from the RPiCM, Cprog, needed to be edited to output the entire 144 BPA subset for each optical fiber end for each round, not just L_v as it would during normal operation. Further, the

[4]Technically this is an affine model due to the intercept term.

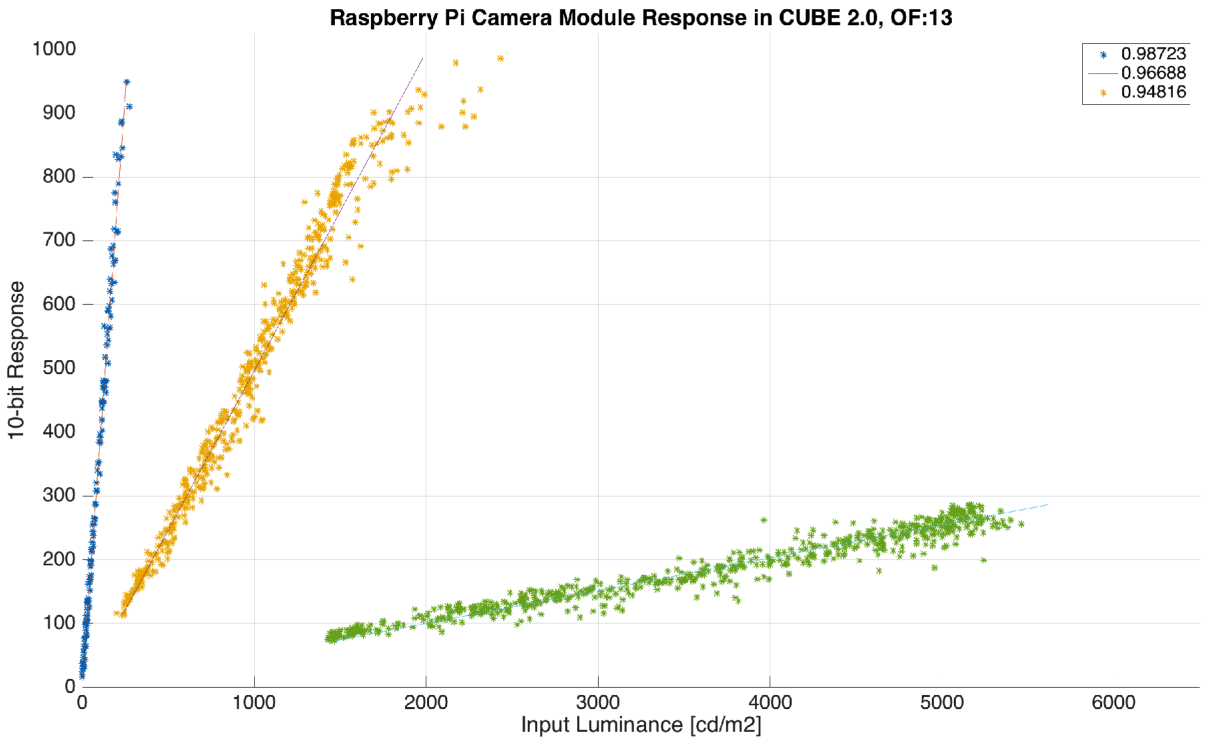


Figure 5.9: The $calPt_n^t$ data points for Klems Basis Patch 13. Notice the highly linear behavior of the optical fiber and RPiCM system.

program fisheye6D11 needed to be edited to output input luminance, $L_v^{in,n}$, and not input illuminance, E_v, as it would during normal operation. Thus, these functionalities has been commented out in the sources code of these programs, yet can still be seen if one is inclined.

Processing this 400 gigabytes of data, however, was the job of another set of Shell, Python, and Matlab scripts which became quite involved as the process was teased out. Given the actual data flow has been formally defined above, implementation details are limited in their presentation here. If one is interested in exploring this “sub-code based” used for calibration, the Matlab function “slopesNOWlimited()” is a good starting point.

As a final implementation step in the calibration process, the Matlab function, “headerMakerRPiCM.m” is executed. This function reads in the matrices which define $\mathcal{S}$ and $\mathcal{I}$ and create headerfiles for the C program Cprog. Cprog is then recompiled to include the new header files and also use these header files to output the exiting luminance distribution function, L_v, instead of the 144 BPA subset per optical fiber end. Further, the optical fiber end locations, discussed below, are also included via header file in the new compilation.

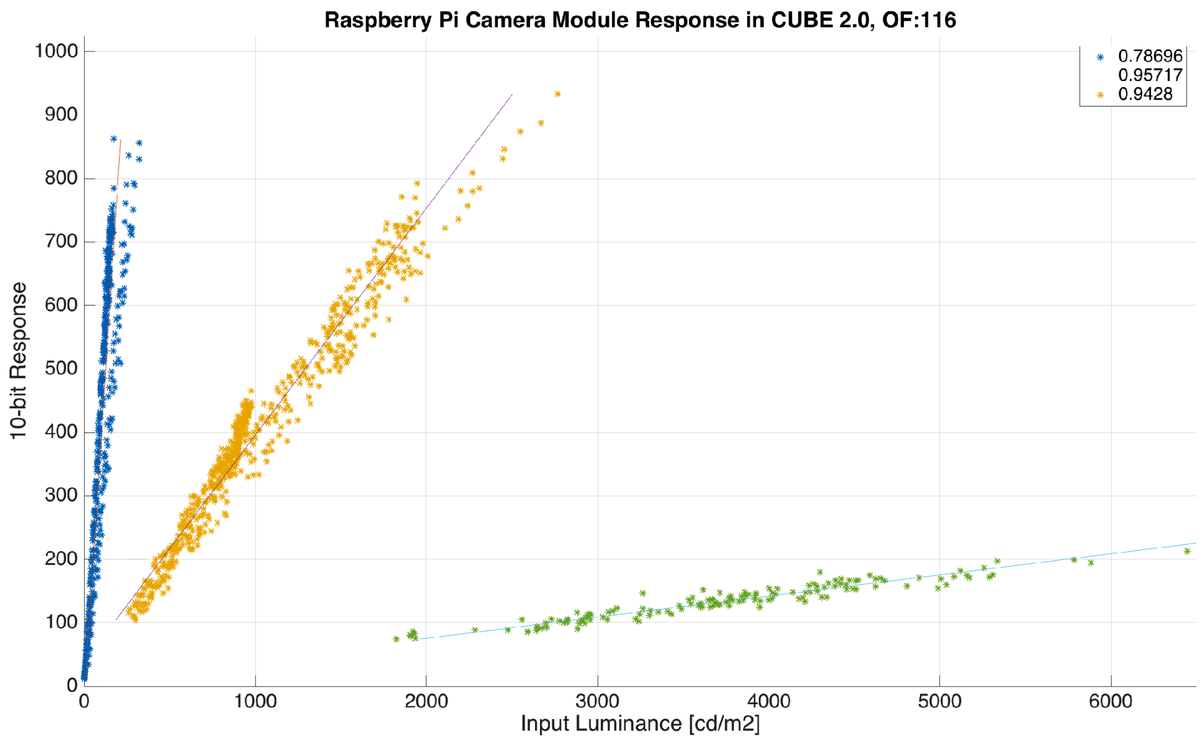


Figure 5.10: The $calPt_n^t$ data points for Klems Basis Patch 116, which shows noisy linear behaviour.

Aligning RPiCM within the Measurement Cone :: Physical Component

With the RPiCM mounted within the measurement cone, the camera must be aligned such that the photograph set $LDRI_{Rpi}^3$ can be used to make a measurement of the optical fiber ends. In relation to the formalisms presented above, this is the definition of the function,

$$OF : \{1, \cdots, 145\} \longrightarrow [i_{of}] \times [j_{of}]. \tag{5.16}$$

The alignment process begins by placing a 50 W halogen lamp directly inside the hemisphere measurement area through the CFS test aperture. This light source ensures each "sampling" end of the optical fibers is well excited, resulting in each "measurement" end inside the measurement cone being clearly visible to the RPiCM.

Then, the Python 2.7 program *alignment.py* is executed on the MacBook Pro. This script, with the aid of *alignShooter.py* running on the RPiCM, uses the Shell scripts *sshAlignment.sh* and *sshAlignmentGet.sh* to respectively SSH and SFTP into the RPiCM, taking and downloading a photographic measurement of the optical fiber ends. These optical fiber ends are known to be excited by the 50 W halogen lamp, hence will be visible to the RPiCM.

After alignment.py returns, the Matlab program *headerMakerRPiCM.m* is executed. This execution produces a Matlab plot of type "surf," on which superimposed yellow boxes

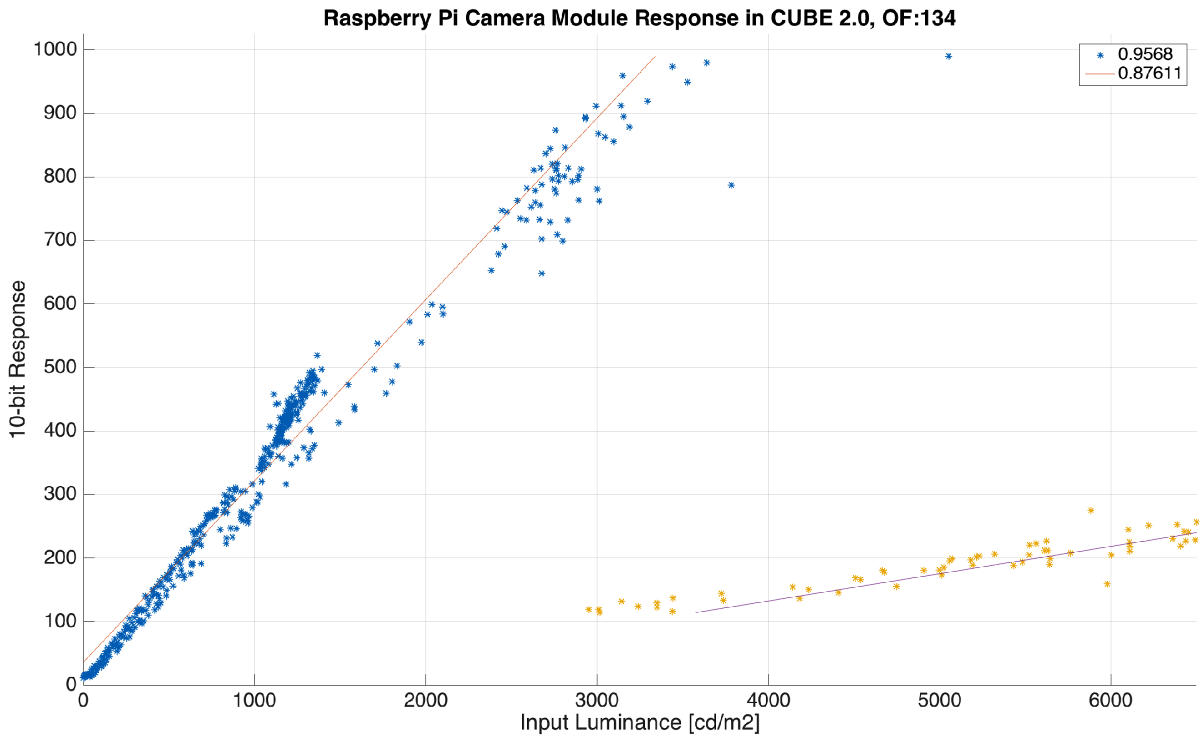


Figure 5.11: The $calPt_n^t$ data points for Klems Basis Patch 134. Notice the limited data for second exposure, resulting in a lower r^2 value due to spread.

appear along with the "measurement" optical fiber ends. The yellow boxes must surround the "measurement" optical fiber ends for the RPiCM to be considered "aligned," which is a necessity for the exiting luminance measurement, L_v, to be valid. If the yellow boxes are not aligned, the threaded rods on the RPiCM bracket described above can be rotated to adjust the alignment of the RPiCM such that they are. As such, the process of executing *alignment.py* and *headerMakerRPiCM.py* respectively is repeated, with appropriate threaded rod adjustments made in between, until the yellow boxes are aligned.

At this time, the RPiCM is considered "aligned" and ready for measurement within the CUBE 2.0 system[5]. With the alignment made, the program headerMakerRPiCM.m "finds" the respective end of each optical fiber by examining the pixels within the "yellow boxes" and performs a weighted average to determine the most bright location. Due to the 50 W halogen lamp, it can be assumed this bright location will be the end of the optical fiber. This

[5]To ensure the proper "yellow box" is aligned with each optical fiber "measurement" end, one must look at the definitions inside *headerMakerRPiCM.m*. It is noted this is tedious and less than ideal, however, given the legacy of development of the CUBE 2.0 system, it remains as redoing this process at this stage is not productive, especially considering this is a one time process and only needs to be repeated in the RPiCM is moved, a rare occurrence that involves taking the CUBE 2.0 apart.

weighted location is then saved as a header file which can be referred by Cprog to encode the function OF.

At this stage, the RPiCM is now attached to the CUBE 2.0 and is aligned within the measurement cone, which are can been in Figure 4.15.

5.2 System Verification

To verify the system was indeed producing the proper data (e.g E_v, L_v) for each round, several long run tests were performed in laboratory conditions. These tests included two 16 hours plus and one 48 hour continuous run, where the CUBE 2.0 was excited using halogen lamps. These data sets were meaningless with respect to the actual numerical values of the measurements, however, “stress testing” the CUBE 2.0 was achieved for long term deployments. This gave confidence the CUBE 2.0 could work when deployed either on location, or in the static location on the roof of Cory Hall at the University of California, Berkeley.

In addition, it was ensured the cyber components were in fact operating properly with respect to the physical hardware. For example, individual optical fiber ends were excited using a laser pointer (causing a much higher value compare to the other optical fibers, making it’s place obvious in the output measurements) to ensure the proper orientation of the optical fiber array within the measurement cone was in fact being processed. Similar tests were conducted for the input illuminance measurement using the Canon 6D and fisheye lens.

One factor which was discovered in the system is the inability of low light situations to have time steps on the order of one minute. Due to the $LDRI^{11}$ collection taking more than one minute. While ideas exit to use limited LDRI sets for these low light situations, it was determined to not implement them, as lighting conditions of this magnitude will have limited impact on the daylighting analysis. That is, no analysis for daylighting is associated with these steps because the condition can still be effectively classified as “night.”

Chapter 6

HWiL Daylighting Simulations and Results

In chapter 5, the calibration process for the *CUBE 2.0* hardware-in-the-loop model (HWiL) was presented. In this chapter, day long simulations with commercially available *complex fenestration systems* (CFS) are conducted using the CUBE 2.0 HWiL model. In addition, a novel CFS, a translucent concrete panel (TCP), is also tested on the CUBE 2.0 for a daylong simulation. Results of all these simulations are presented and comparisons to validations using measured BTDF for one of the commercially available CFS are also presented.

6.1 Simulation Results

With the CUBE 2.0 system fully designed, constructed, calibrated, and verified in its performance, experiments with real complex fenestration systems (CFS) can now be executed. In this thesis three CFSs will be presented, two commercial products and one novel CFS designed by other members of the author's research group.

For the first commercial product, Twitchell's Textilene 80 Black (Twitchell Product Number: T18DES036), the bidirectional transfer distribution function (BTDF) has been measured as part of the Complex Glazing and Shading Database (CGDB) [35]. This allows for a validation of the CUBE 2.0's performance as the input to the CFS under test (i.e. input illuminance distribution, denoted E_v) can be used with respect to the three-phase method to compute the same factor which is measured in the CUBE 2.0 (i.e. exiting luminance distribution, denoted L_v). In addition a second commercial product, Twitchell's Shade View Ebony (Twitchell Product Number: T18AMS001) is tested, however, it does not have a BTDF, hence its results are presented as an example output of a commercial product without validation.

The novel CFS measured is a translucent concrete panel (TCP) [3]. A TCP has never been tested on a goniophotometer before, thus no measured BTDF exists, hence as with the

Shade View Ebony, only the final results and no validation will be presented.

6.1.1 Twitchell Textilene®

A large subset of all CFSs is the class known as shades. A shade can be thought of as some fabric or plate type barrier which is placed in an opening of a building envelope to modify the luminous energy entering the building space through it. Shade type CFSs can be used alone, as in for screened in porches, or in combination with glazing units. Further, shades can be dynamic, using either automated deployment systems or human activated systems, or static, where the shades are in place all the time.

Regardless of a static or dynamic setup, the purpose of shades is to reduce the amount of luminous energy entering a space, yet maintain some connection to the outside environment. In addition, shades offer a level of daytime privacy which is often sought in densely populated areas. Shades are often quantified with respect to their transmission ratio, that is the total proportion of light leaving the back side with respect to the incident light on the front side, irrespective of direction. Further, due to the shade the light no longer travels through a building envelope opening in a specular manner, hence bidirectional transfer distribution functions (BTDF) must be used to quantify the modification of the luminous energy through the shade. This non-specular transmission through the enclosure opening is what classifies shades as CFSs.

Here, two examples of commercially available shades are analyzed using the CUBE 2.0, Textilene 80 Black and Shade View Ebony. Both are manufactured by the industry leading Twitchell Corporation [150], have nominal transmission ratios of 80 and 90 percent, and are members of the versatile Textilene® family. The first CFS, Textilene 80 Black, has been tested on a goniophotometer and characterized via a BTDF in the Complex Glazing and Shading Database (CGDB).

Simulation Setup

For both Twitchell's Textilene 80 Black, see Figure 6.1, and Shade View Ebony, see Figure 6.2, the same experiment was conducted. For a whole day, that is from Nautical Twilight start to end (i.e. approximately one hour pre-sunrise and post-sunset), at a time step equal to one minute, a simulation of the respective CFS is conducted. This involves measuring the input illuminance distribution (E_v), exiting luminance distribution (L_v), the associated luminous metrics as part of the Radiance simulation (i_m), the simulated exiting luminance distribution using the BTDF and E_v ($L_{v,s}$) and the simulated luminous metrics using $L_{v,s}$ ($i_{m,s}$).

To quantify the validation of the CUBE 2.0 a comparison can be made with respect to the exiting luminance distribution: $L_v^{t,n}$ and $L_{v,s}^{t,n}$ for all $t \in \{1, \ldots, t_{end}\}$ and $n \in \{1, \ldots, 145\}$. These measured and simulated exiting luminance distribution values, however, are only intermediate, with the final desired result being the actual luminous metrics calculated by

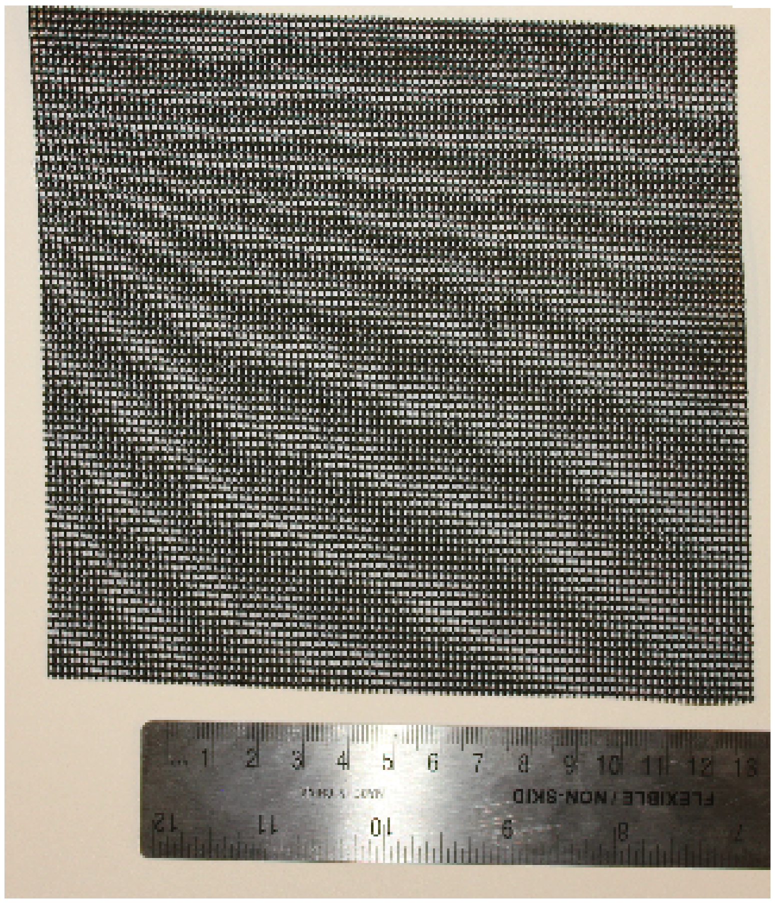


Figure 6.1: Twitchell's Textilene 80 Black shade material.

Radiance and represented by i_m and $i_{m,s}$. Hence, further comparisons are made concerning the outputs of the Radiance simulation results.

Results: Twitchell Textilene 80 Black®

The simulation involving the CFS Twitchell Textilene 80 Black was conducted on March 13, 2017 from Nautical sunrise (6:26 am) to sunset (8:11 pm). At one minute intervals this resulted in 757 total time steps. Recall, at low light conditions a step needs more than one minute to execute, hence 757 steps, instead of the calculated 825, comprise the simulation. Note, however, the vast majority of the simulation time horizon, and from a magnitude perspective, the meaningful simulation duration, all have time steps of exactly one minute.

First, in Figure 6.4 the exiting luminance distribution as measured by the CUBE 2.0 system, L_v, is presented. Next, in Figure 6.5 the simulated exiting luminance distribution, $L_{v,s}$, is presented. These two plots show the respective measured and simulated exiting

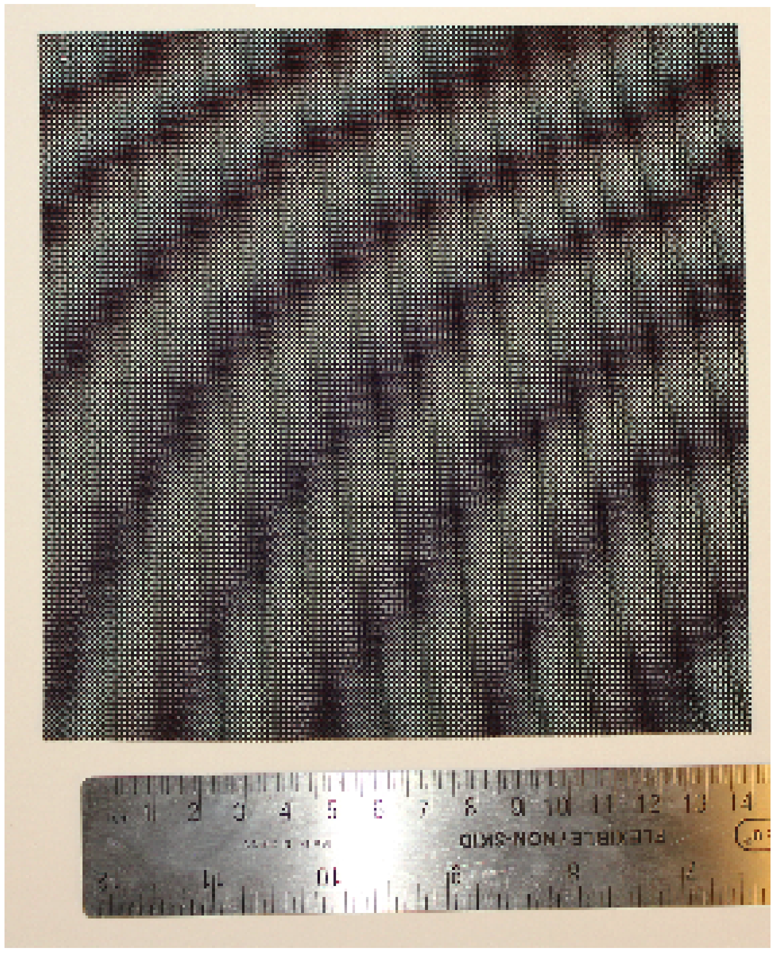


Figure 6.2: Twitchell's Shade View Ebony shade material.

Table 6.1: Outline of the simulation for Textilene 80 Black.

Round Number:	Time of Day:	Significance
1	6:26 am	Start
15	7:20 am	Sunrise
26	7:59 am	1 Min. Time Steps Begin
700	7:14 pm	Sunset
757	8:11 pm	Stop

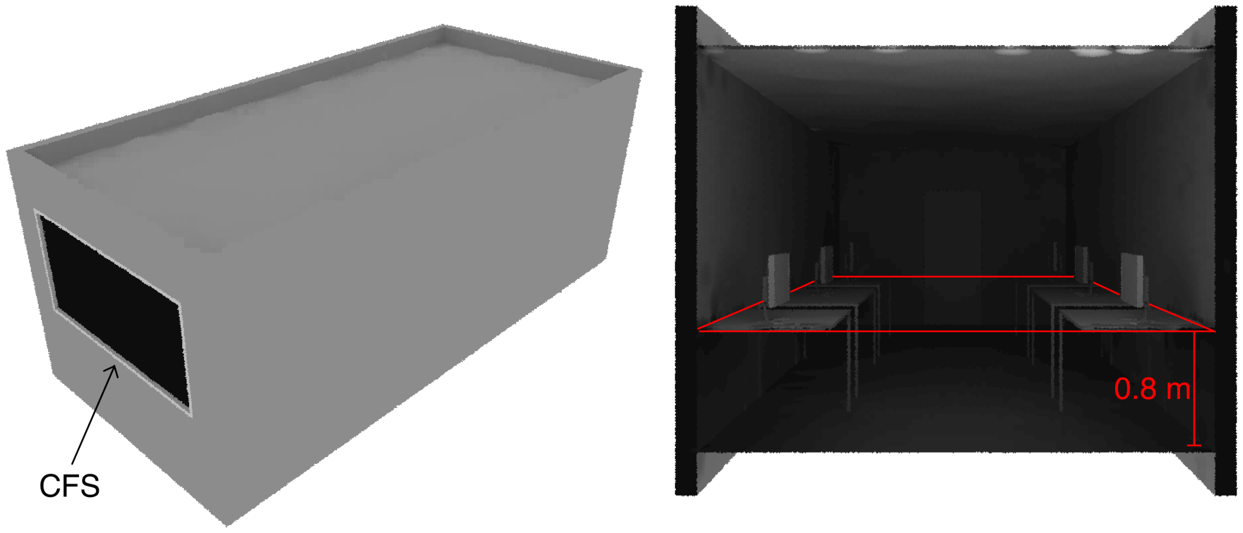


Figure 6.3: Right: An outside view rendering of the Reference Office; Left: An inside view rendering of the Reference Office. The red plane indicates the location of the working plane at which horizontal illuminance is measured.

luminance distribution data aligned in a vector and plotted for each time step. This results in 145 values per time step.

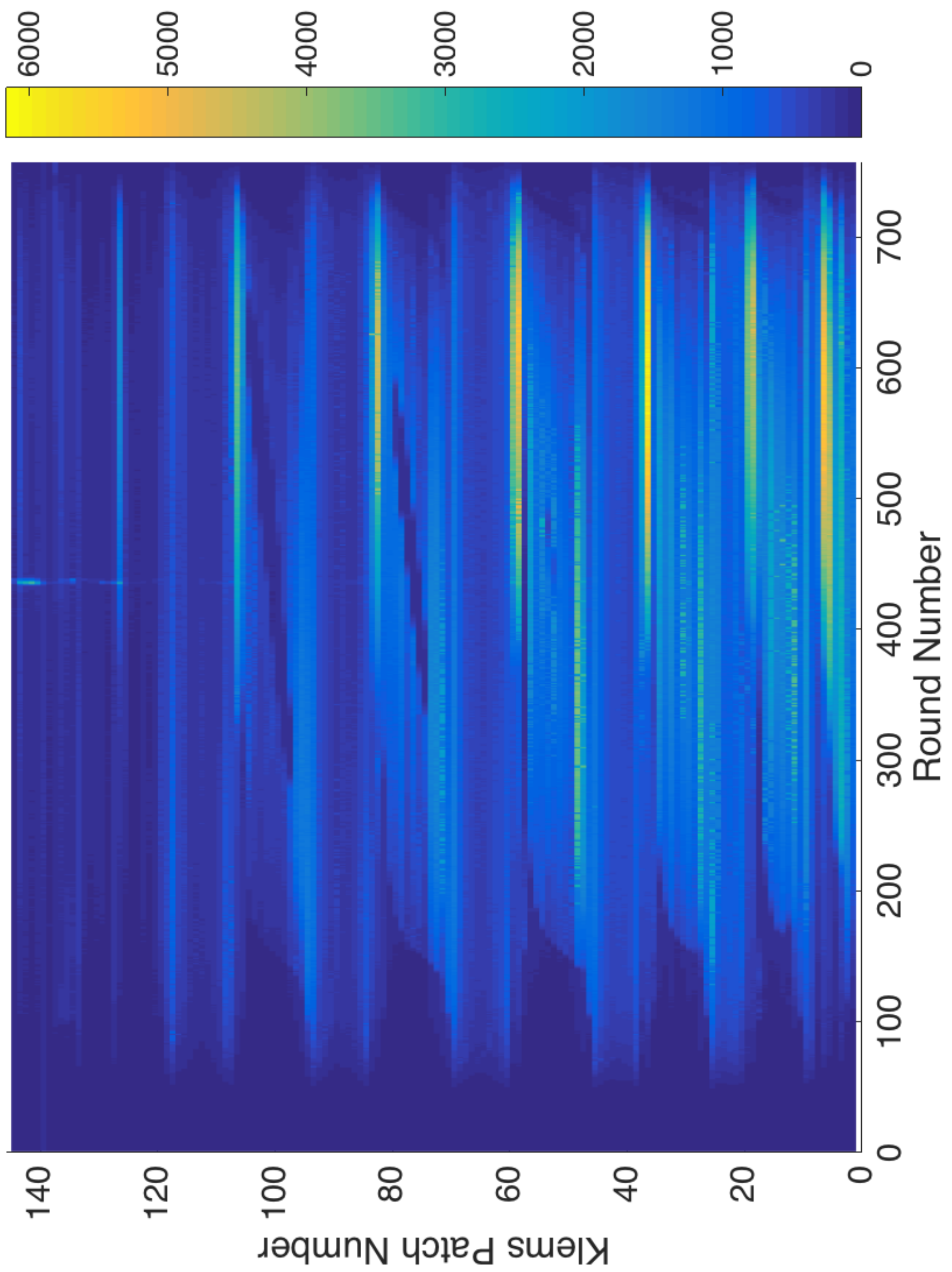


Figure 6.4: Measured exiting luminance distribution, L_v^t, for all $t \in \{1, \ldots, 757\}$, for Textilene 80 Black, in units of $[\frac{cd}{m^2}]$.

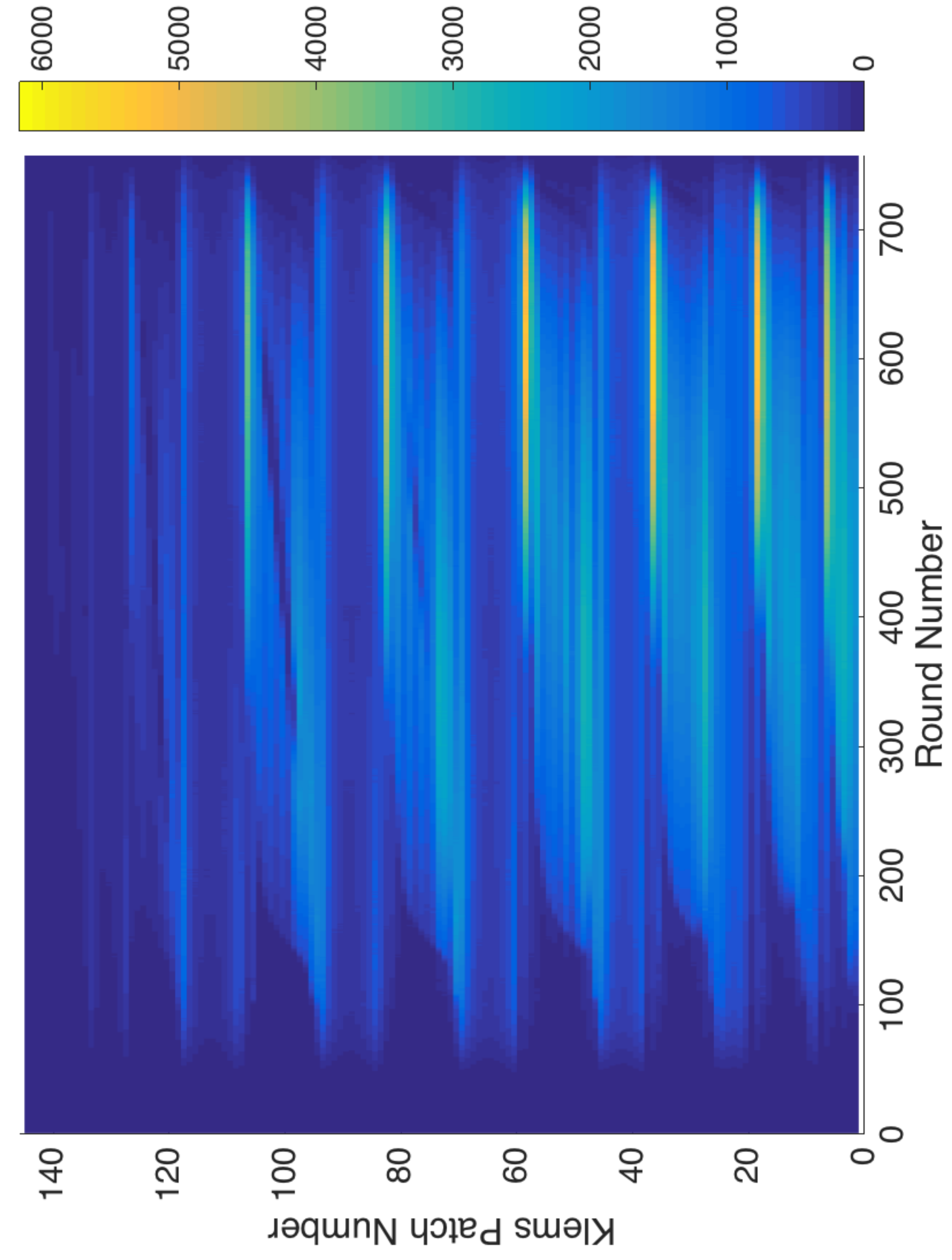


Figure 6.5: Simulated exiting luminance distribution, $L^t_{v,s}$, for all $t \in \{1, \ldots, 757\}$, for Textilene 80 Black, in units of $[\frac{cd}{m^2}]$.

Presented next, is a representation of the actual luminous metrics as calculated by Radiance. For this simulation Radiance calculated horizontal illuminance at the workplane height of 0.8 m within the Reference Office. This horizontal illuminance was calculated at a grid of points spaced every 10 centimeter, resulting in a set of 35 by 81 measurements. With point (1,1) at the left side of the CFS wall when looking directly at the CFS and point (81,35) in the back right corner, the grid is reshaped into a vector $35 \times 81 = 2835$ units long, making 2835 points per round. Figure 6.3 shows a rendering of both an outside and inside view of the reference office. The inside view is marked with a red boundary to indicate the location of the working plane at 0.8 meters. The results of this model and simulation, i_m and $i_{m,s}$, are shown in the respective plots in Figure 6.6 and 6.7.

As a further reference to confirm intuition, the calculated horizontal illuminance on the 0.8 m workplane of the Textilene 80 Black sample, for both i_m and $i_{m,s}$, are shown in plan view for the Reference Office at 4:57 pm (t=563) in Figure 6.8 and Figure 6.9.

Results: Twitchell Shade View Ebony®

The simulation involving the CFS Twitchell Shade View Ebony® was conducted on March 12, 2017 from Nautical sunrise (6:27 am) to sunset (8:10 pm). At one minute intervals this resulted in 773 total time steps. 773 steps, instead of the calculated 823, comprise the simulation[1]. Note, however, the vast majority of the simulation time horizon, and from a magnitude perspective, the meaningful simulation duration, all have time steps of exactly one minute duration.

Further, unlike the Textilene 80 Black CFS from Twitchell, the Shade View Ebony does not have a measured BTDF. As such, only the simulation results of the exiting luminance distribution, L_v, will be presented, along with the horizontal illuminance, i_m, as part of the hardware-in-the-loop simulation.

First, in Figure 6.10 the exiting luminance distribution as measured by the CUBE 2.0 system, L_v, is presented. Next, in Figure 6.11 the Radiance model output of i_m is presented.

Finally, to add to the intuition of the system, a plan view of the Reference Office showing calculated horizontal illuminance on the 0.8 m workplane for the Shade View Ebony sample, i_m, at time step t=343 (12:34 pm) is presented in Figure 6.12.

[1]Notice the day is shorter for Shade View Ebony than Textilene 80 Black, yet Shade View Ebony has more time steps. This can be explained in that the shutter speed chosen for the $LDRI^{11}$ is based on the camera meter reading, which could cause time steps to be executed faster or slower. With the time steps already over one minute, these differences will affect the number of total time steps in the simulation. At longer shutter speeds, the Canon 6D has limited options due to manufacturer limitations, for example jumping from 15 to 30 seconds. Hence, a small luminance difference can expand the input illuminance measurement by several seconds. Note as well, the built in luminance meter may be pointed at a dark area of the input hemisphere.

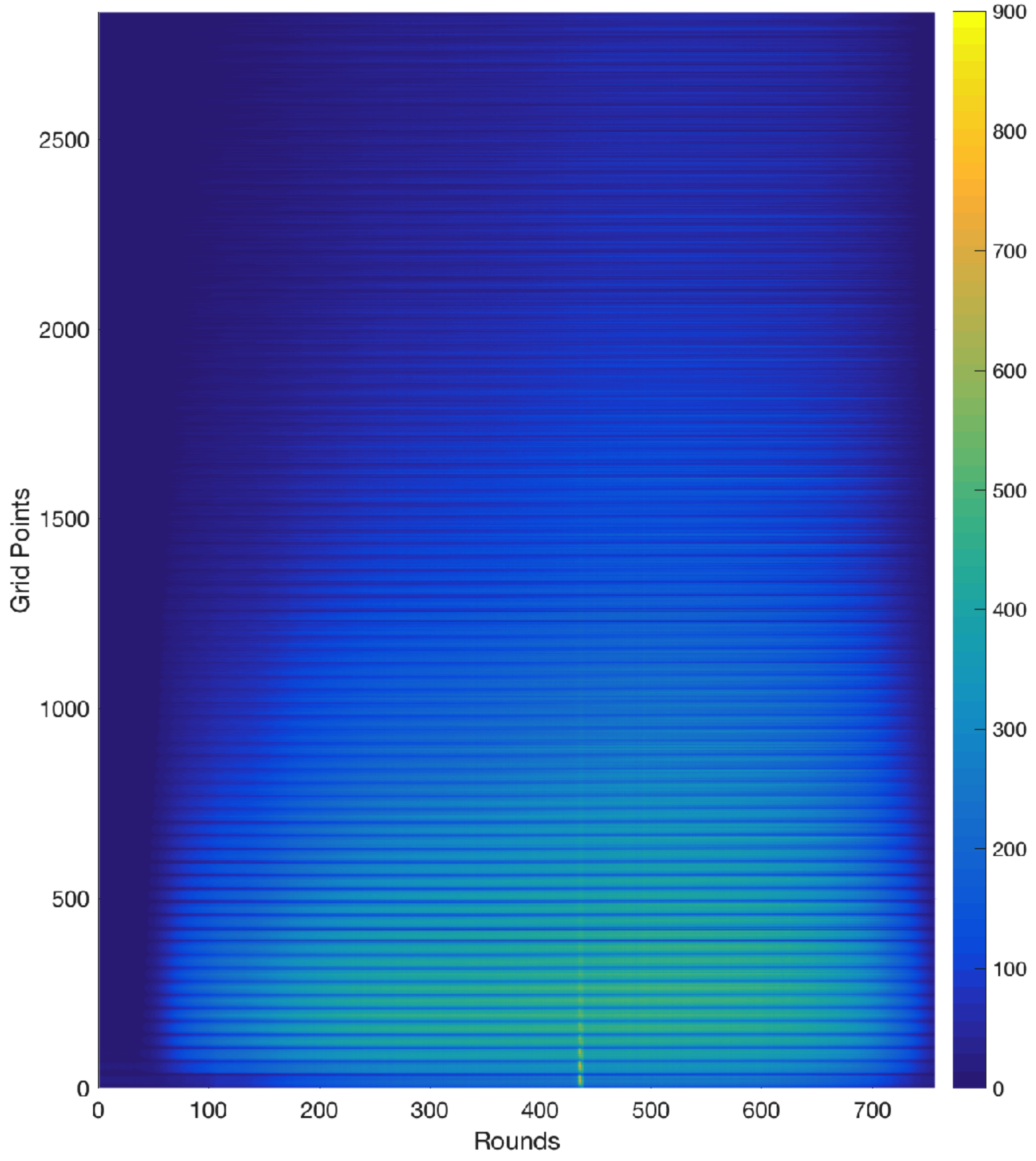


Figure 6.6: Simulated workplane (0.8 m) horizontal illuminance, i_m^t using L_v^t, for all $t \in \{1, \ldots, 757\}$, oriented as a vector, for Textilene 80 Black, in units of [lux].

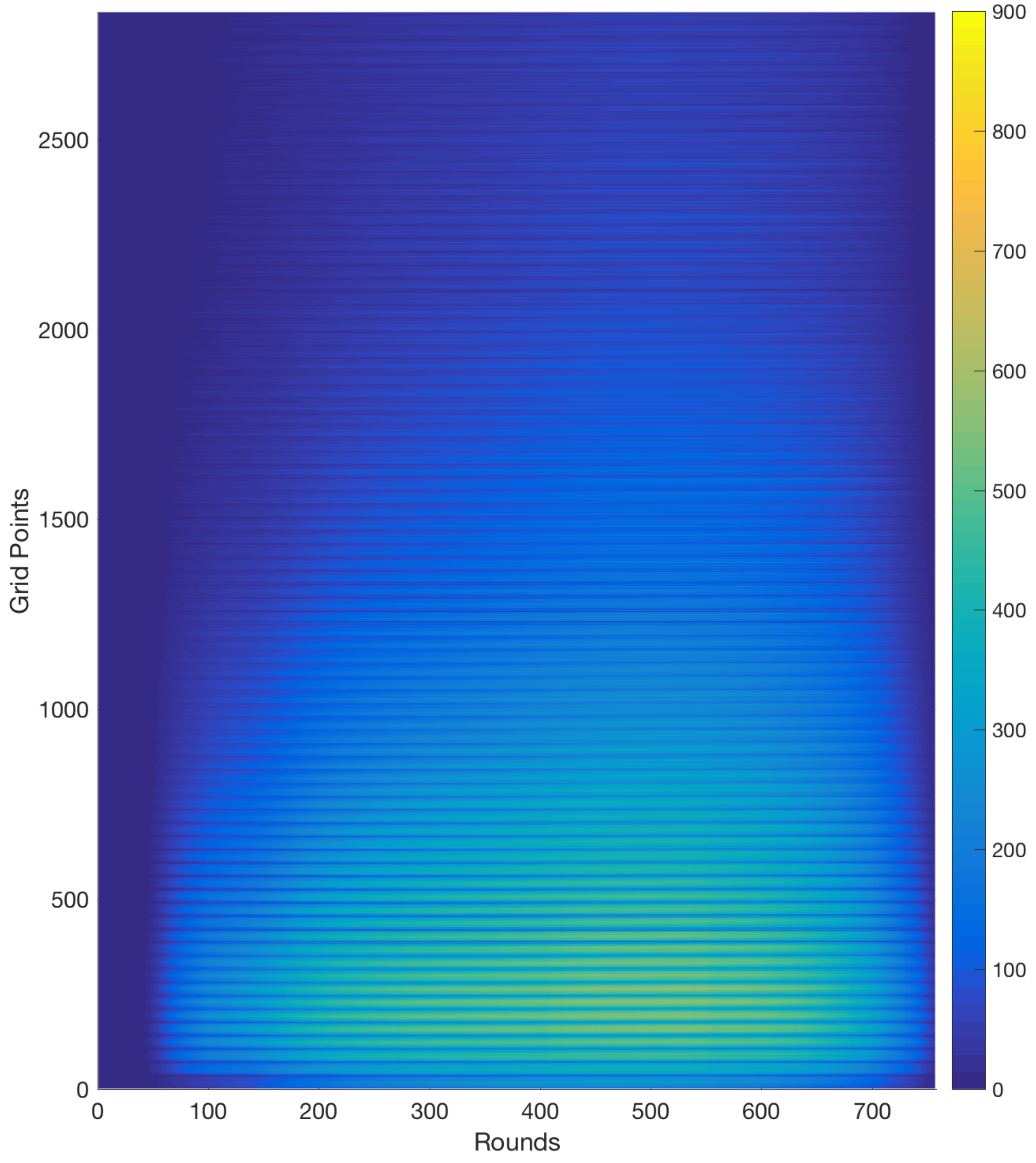


Figure 6.7: Simulated workplane (0.8 m) horizontal illuminance, $i^t_{m,s}$ using $L^t_{v,s}$, for all $t \in \{1, \dots, 757\}$, oriented as a vector, for Textilene 80 Black, in units of [lux].

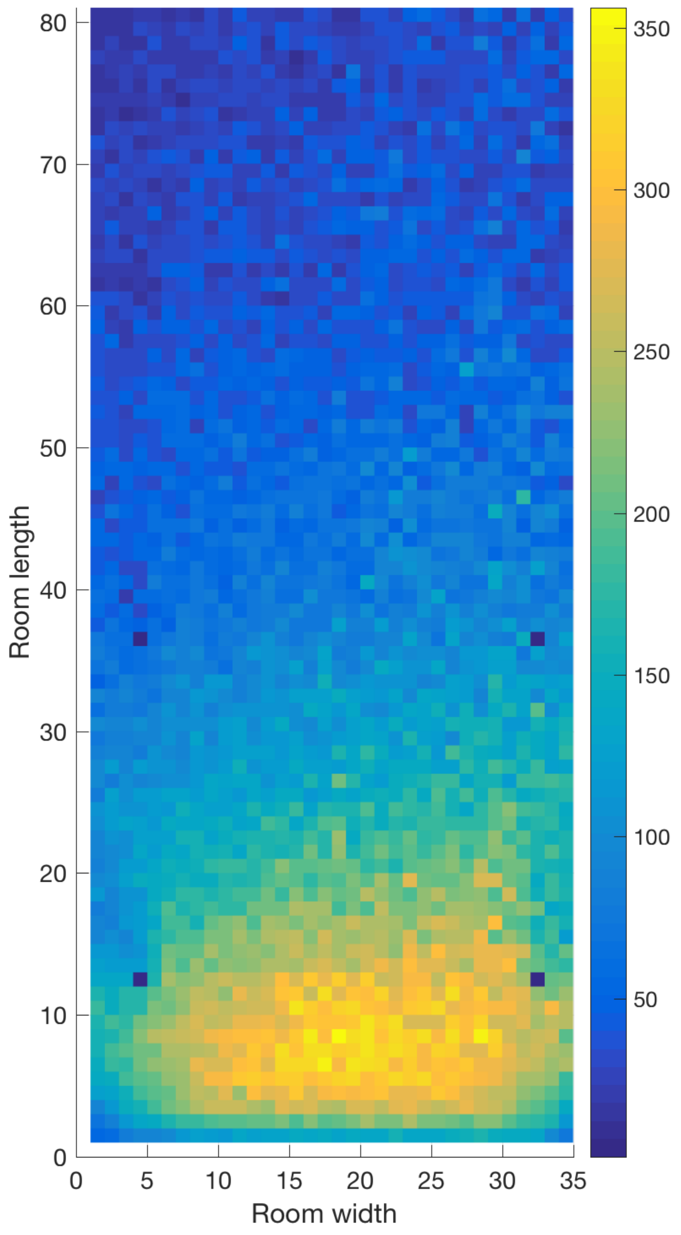


Figure 6.8: Horizontal illuminance at 0.8 m workplane, i_m, for Textilene 80 at 4:57 pm, for Textilene 80 Black, in units of [lux].

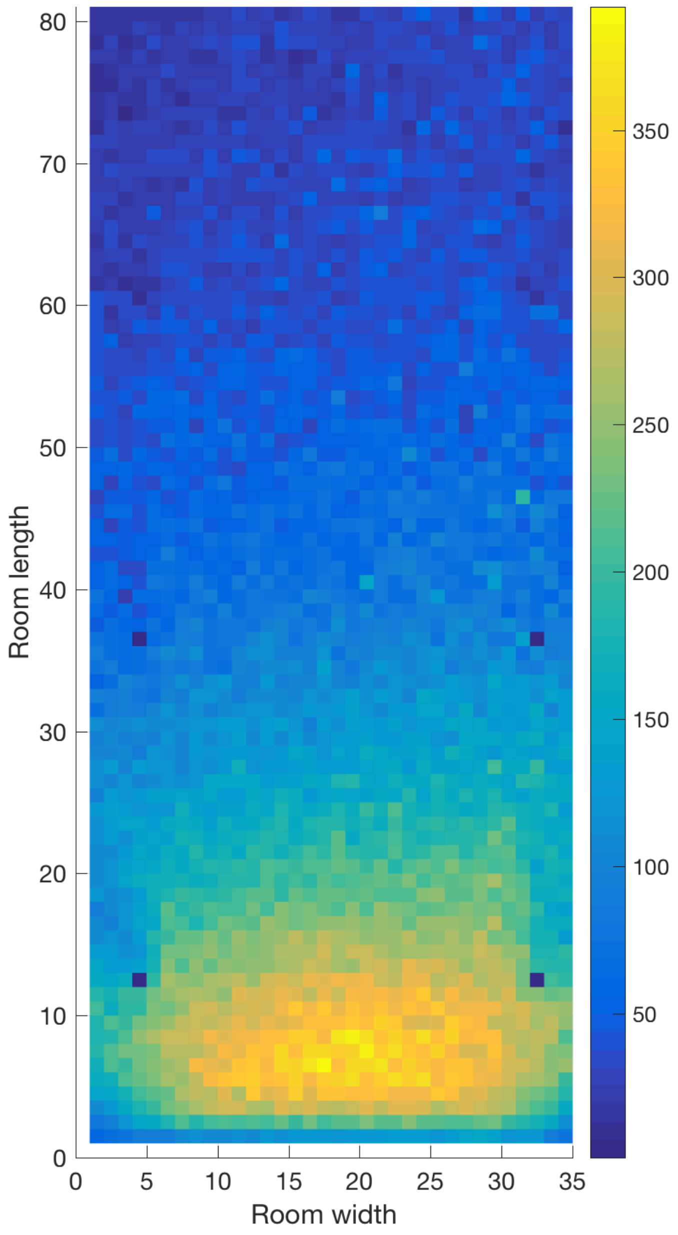


Figure 6.9: Horizontal illuminance at 0.8 m workplane, $i_{m,s}$, for Textilene 80 at 4:57 pm, for Textilene 80 Black, in units of [lux].

Table 6.2: Outline of the simulation for Shade View Ebony.

Round Number:	Time of Day:	Significance
1	6:25 am	Start
15	7:24 am	1 Min. Time Steps Begin
33	7:59 am	Sunrise
742	7:13 pm	Sunset
763	7:35 pm	1 Min. Time Steps End
773	8:05 pm	Stop

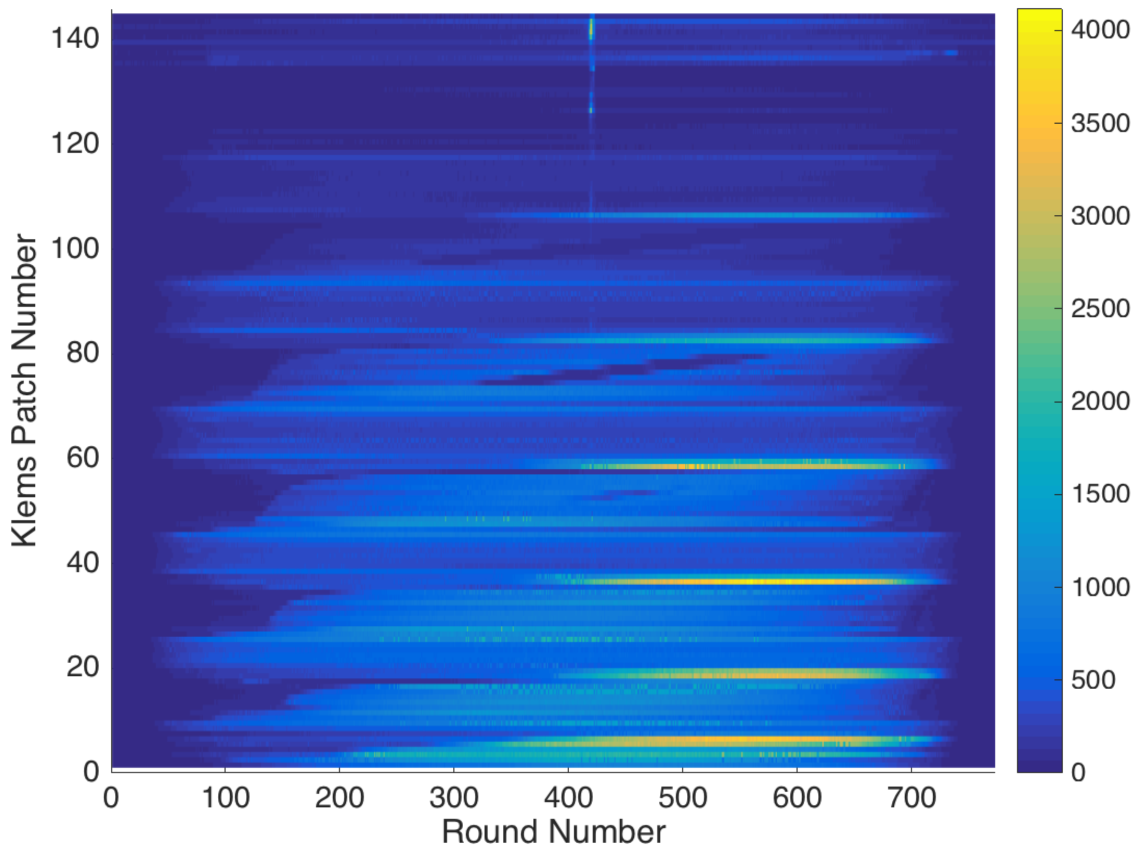


Figure 6.10: Exiting luminance distribution, L_v^t, for all $t \in \{1, \dots, 773\}$ for Shade View Ebony, in units of $[\frac{cd}{m^2}]$.

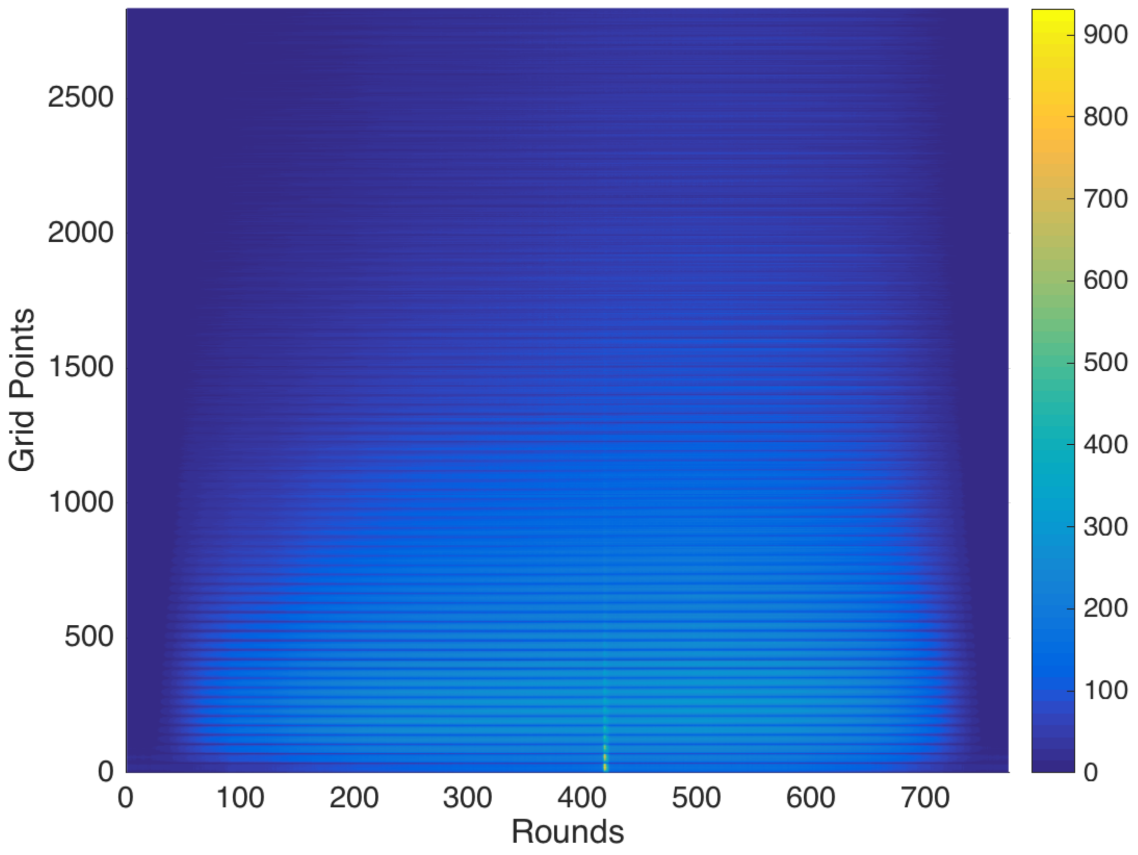


Figure 6.11: Horizontal illuminance at 0.8 m workplane height, i_m^t, for all $t \in \{1, \ldots, 773\}$, oriented as a vector, for Shade View Ebony, in units of [lux].

Results Discussion

Textilene 80 Black: To begin, one should notice the striking qualitative agreement between the measured and simulated exiting luminance distribution, Figures 6.4 and 6.5. The brightest spots of the input hemisphere are associated with the sky and indeed these bright spots do in fact exist in both L_v and $L_{v,s}$ in their proper locations. That is the skyward facing Klems Basis patches report higher values. Further, the general shape of the luminance distribution exiting the CFS is very similar with lobes and valleys appearing in L_v and $L_{v,s}$ as one would expect.

Quantitatively, however, to process through the large volume of results for the CUBE 2.0 system, various metrics must be used. These metrics will quantify the agreement between the actual measured values (i.e. L_v and i_m) and their corresponding simulated values (i.e. $L_{v,s}$ and $i_{m,s}$), allowing for a rigorous evaluation of the CUBE 2.0 system.

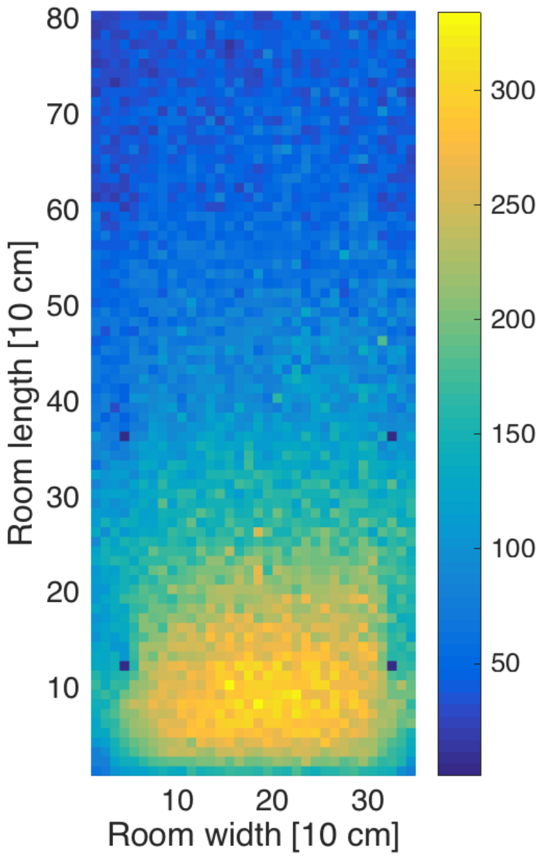


Figure 6.12: Horizontal illuminance at 0.8 m workplane height, i_m^t, for time step t=343, for Shade View Ebony, in units of [lux].

Beginning with an individual round, take for example time step t=563[2] as shown above, consider the exiting luminance distribution, L_v^{563} and $L_{v,s}^{563}$. Here, the individual Klems Basis patches are plotted with their respective luminance values in Figure 6.13. Notice the near exact qualitative agreement between "peaks" and "valleys" for nearly all the Klems Basis patches. If the alignment were prefect and the measurement of both the input illuminance distribution and BTDF of the sample were exact, these two lines would overlap perfectly. However, the alignment isn't perfect (i.e. $L(\theta,\phi)_{cfs} \neq L(\theta,\phi)_{fish}$) and there exists errors in both E_v and L_v, hence they don't agree.

One way of quantitatively characterizing this error is to plot the two distributions as pairs of points on a simple two dimensional plane, where the plotted points form the set,

$$L_{pts} = \{(L_v^1, L_{v,s}^1), \ldots, (L_v^{145}, L_{v,s}^{145})\}. \tag{6.1}$$

[2]t=563 is a typical round and chosen as a representation, no special reason exists for its presentation.

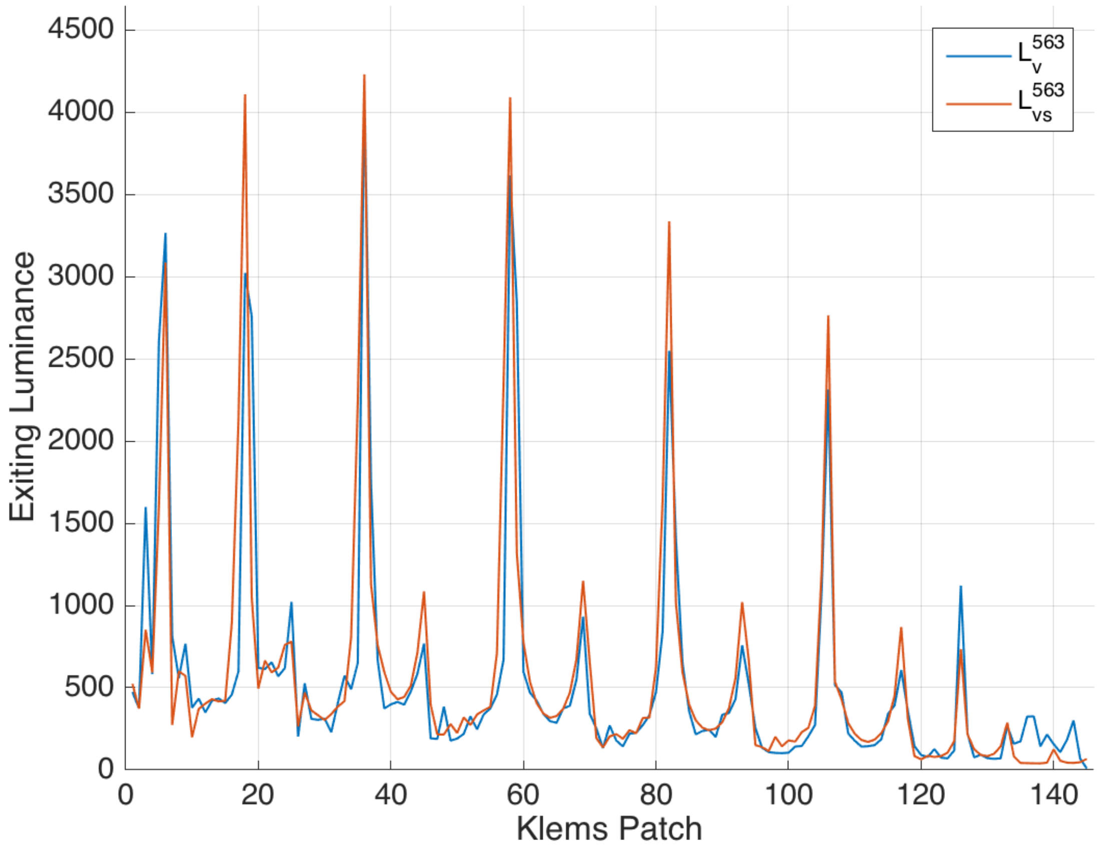


Figure 6.13: L_v^{563} and $L_{v,s}^{563}$ plotted together for comparison, in units of $[\frac{cd}{m^2}]$.

These points should fall on a straight line $x = y$, hence, by fitting a linear function in the sense of least squares and evaluating goodness of fit metrics, one can get a measure of the agreement between the two exiting luminance distributions [63].

This process is demonstrated using the time step t=563 (4:57 pm) and can be seen in Figure 6.14. If the measurements were in perfect agreement, they would all fall on a line with slope of one, shown in the diagram as the red line. Clearly this doesn't happen, however, the least squares best fit line, shown in yellow, is a close match. Notice the slope is 0.93 where as an ideal value would be 1.0.

While the lower luminance values are very "tight," the higher luminance values tend to be in less agreement. This spread of the data is captured by the r^2 metric, with a value of 0.76 where a perfect value would be 1.0. Notice the spread has values both above and below the unit slope line. It is believed this is due to the courser discretization of the CUBE 2.0 measurement system for the exiting luminance distribution (i.e. optical fibers and RPiCM)

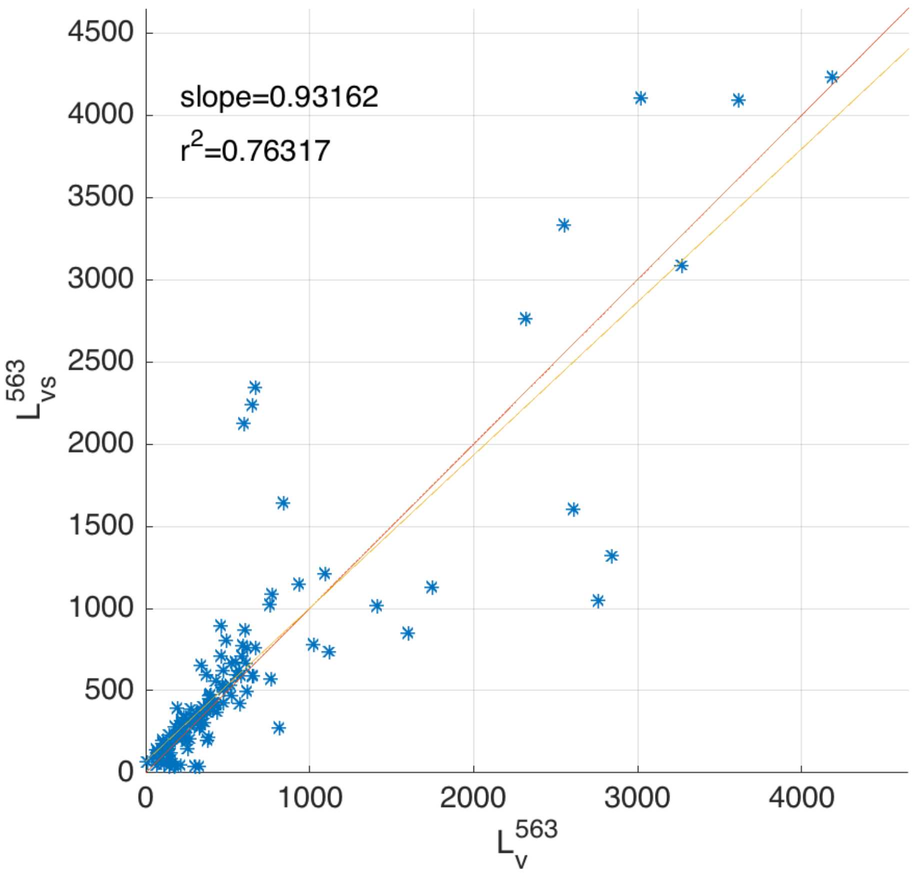


Figure 6.14: L_{pts} plotted with best fit line (yellow) and ideal fit line (red), in units of $[\frac{cd}{m^2}]$.

as compared to the input luminance distribution (Canon 6D and fisheye lens). For example, a large luminance value may not be sampled by a given optical fiber, yet it impacts the Canon 6D fisheye lens measurement resulting in a $L^n_{v,s}$ measurement higher than the corresponding L^n_v measurement (point above the red line). Alternatively, the optical fiber may sample a relatively higher luminance value and, given the CUBE 2.0 operational semantics, assumes this takes place over the entire input Klems Basis Patch. This will result in a value of L^n_v which is higher then the corresponding $L^n_{v,s}$ (point below the red line). Given both situations are born out in the data, it is believed this is a very possible explanation of the observed spread.

The agreement is within intuitive sense and on the order of magnitude one would expect for errors of luminance measurement. Thus, at least for one round, t=563, the CUBE 2.0 appears to be working correctly.

With one round evaluated in this manner, t=563, attention is now turned to the remain-

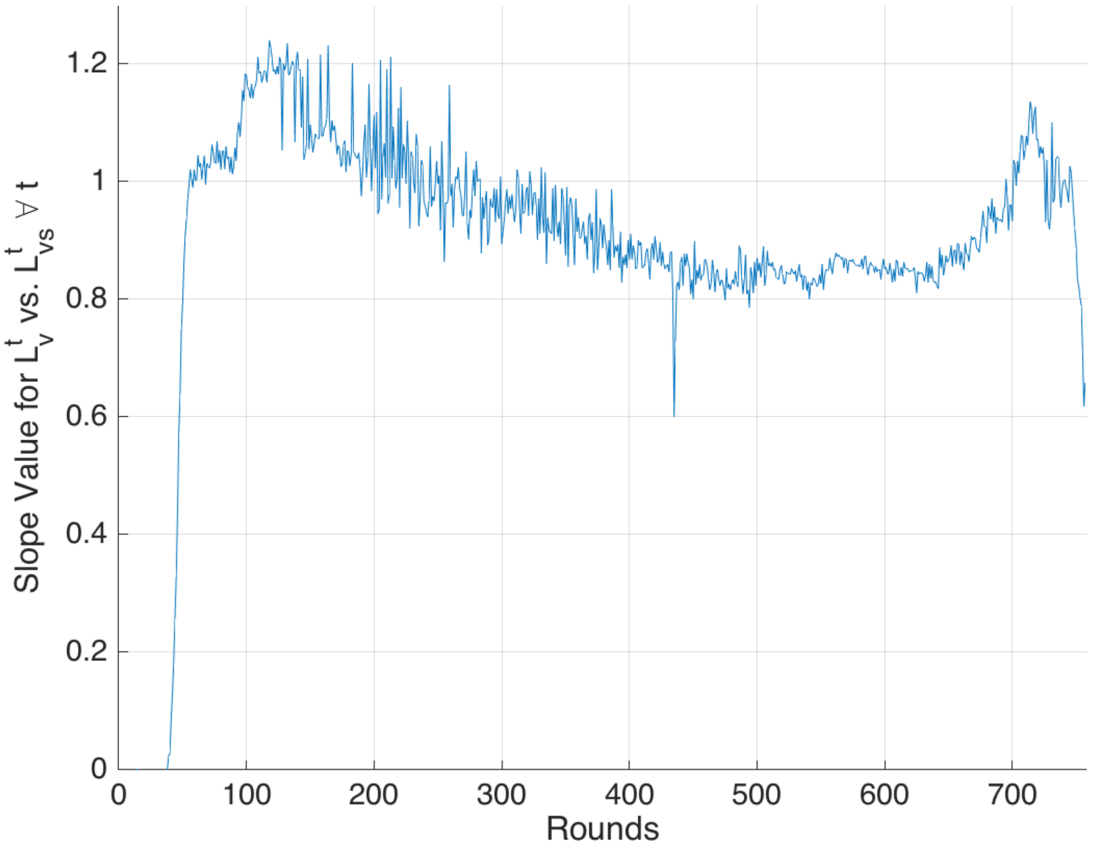


Figure 6.15: Slope values for all the least square fit lines for L_v and $L_{v,s}$, note ideal values would all be 1.0.

der of the 757 rounds. The same process of plotting L_{pts} is thus carried out for the remaining rounds with their goodness of fit metrics evaluated in total. Examining Figures 6.15 and 6.16, one can see the slopes and r^2 values for each of the 757 rounds of the Textilene 80 Black simulation.

Examining these metrics, there continues to be an agreement between the measured exiting luminance values, L_v, and the simulated values, $L_{v,s}$. The slopes are distributed both above and below 1.0, offering evidence the CUBE 2.0 is not biased in it's measurements. Further, a slope greater than 1.0 in general indicates $L^n_{v,s} > L^n_v$ and conversely a slope less than 1.0 in general indicates $L^n_{v,s} < L^n_v$. One should note the intercepts can affect this, however, in some cases, yet these were not observed here.

While the exiting luminance distribution numbers are important, as they are the input for the final simulation, the luminous metrics are the final goal (i.e. i_m, $i_{m,s}$), and are thus

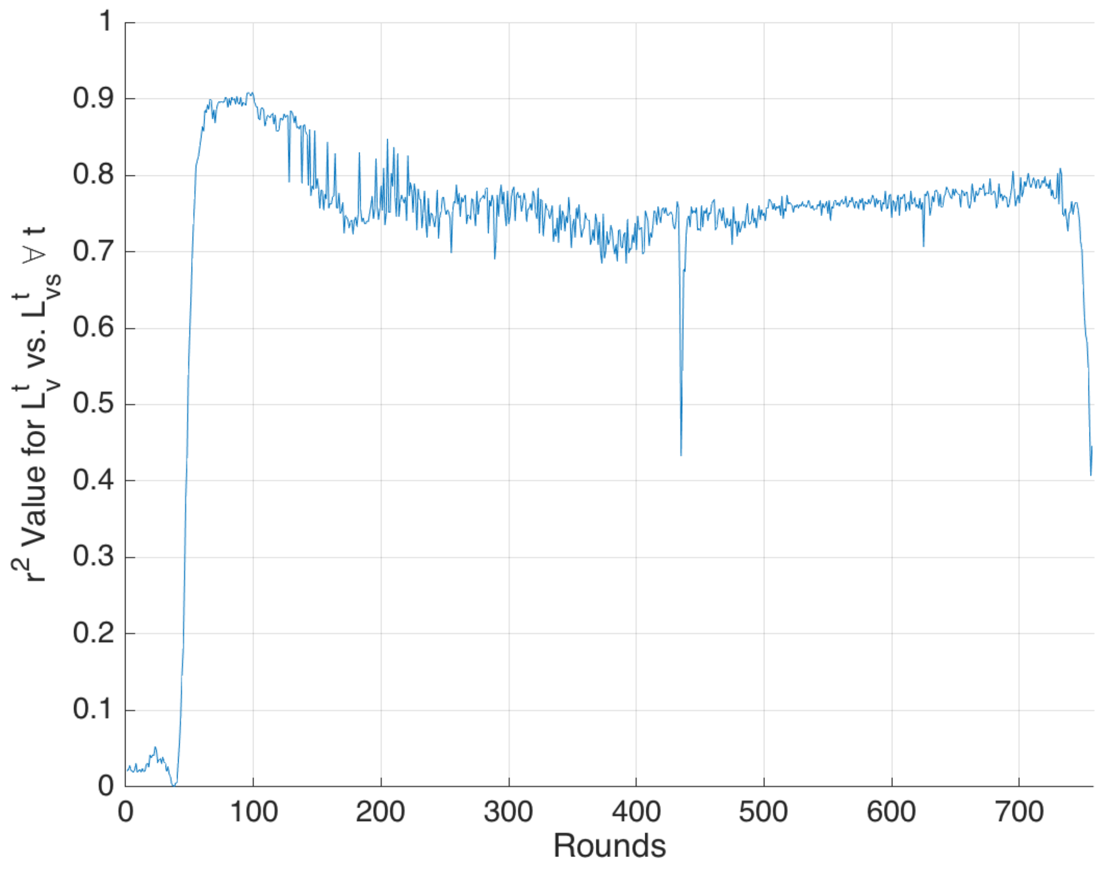


Figure 6.16: r^2 values for all the least square fit lines for L_v and $L_{v,s}$, note ideal values would all be 1.0.

discussed now. First, each grid of points (81,35) is transformed into a vector (1,2835). An example of this vector for time step t=563 is shown for both i_m^{563} and $i_{m,s}^{563}$ in Figure 6.17. While the number of points is large, and therefore the plot is a bit crowded, the qualitative agreement is again quite clear. It should be pointed out, the two pairs of zero horizontal illuminance are places where desk objects are located, hence have zero horizontal illuminance as expected.

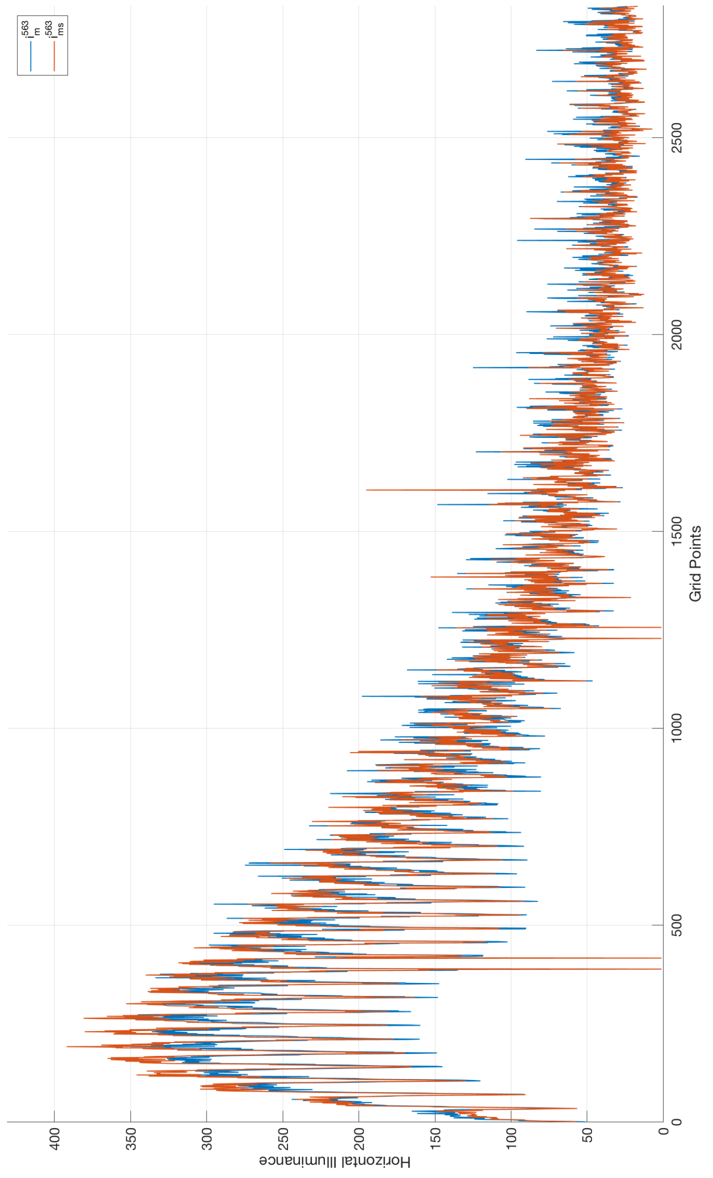


Figure 6.17: i_m^{563} and $i_{m,s}^{563}$ in vector form plotted together for comparison, in units of [lux].

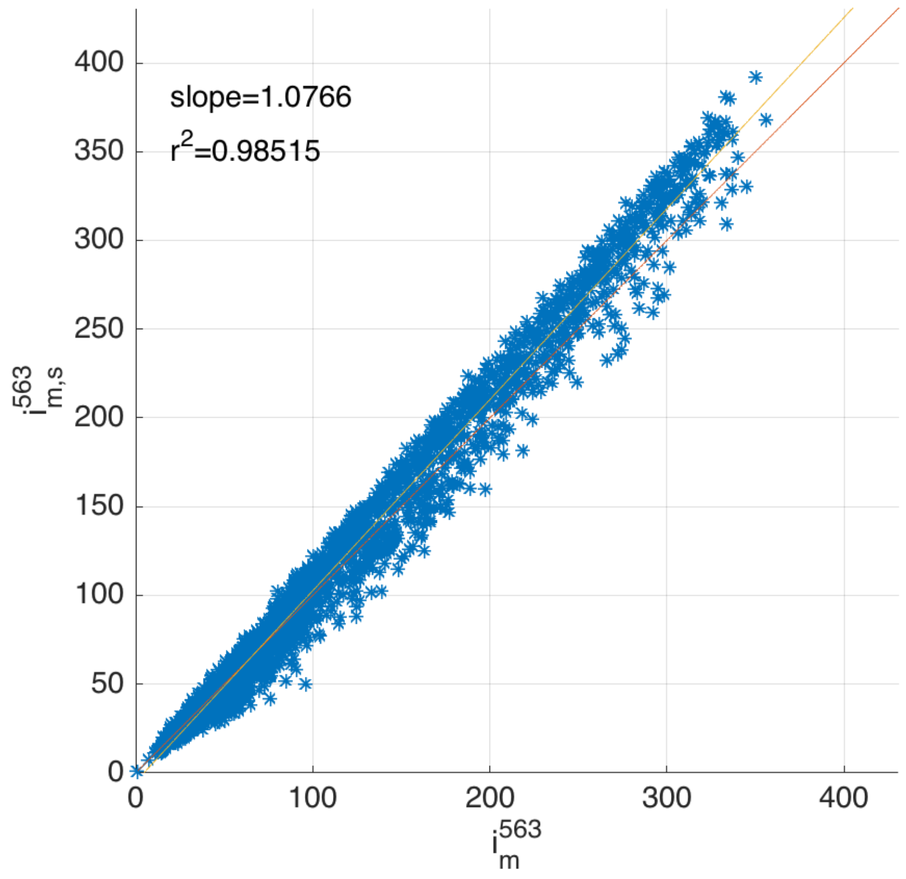


Figure 6.18: i_m^{563} and $i_{m,s}^{563}$ plotted as pairs of points for agreement comparison, in units of [lux].

Using the same notion of plotting pairs of points and fitting a line as with the exiting luminance distribution, the horizontal illuminance at the working plane of 0.8 m height are plotted. Recall, the horizontal illuminance is calculated by Radiance using the three-phase method. Beginning with time step t=563 in Figure 6.18, it can be observed there is a very nice agreement between the measured and simulated data. In contrast to the exiting luminance distribution measurements, here there is a clear bias of points reporting higher illuminance values in the simulation, $i_{m,s}$, as compared to the measured value, i_m. This means, the hardware-in-the-loop model which is executed by using the CUBE 2.0 system has over estimated the horizontal illuminance as compared to the simulated values. This over estimation however, is quite small as evidenced by the slope having value of 1.0766. In conjunction with this, the r^2 value is 0.98515, which is very close to the ideal value of 1.0. This means, there exists a very high correlation between the measured and simulated grid points.

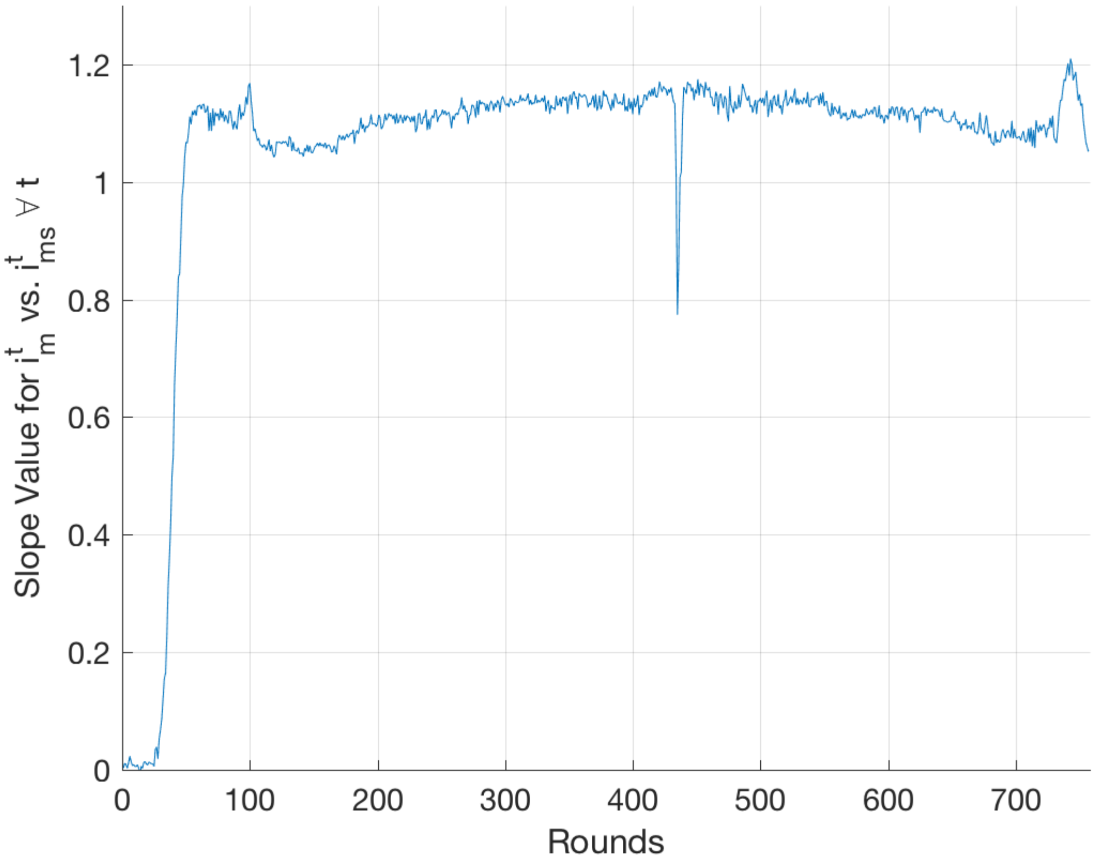


Figure 6.19: Slope values for all the least square fit lines for i_m and $i_{m,s}$, note ideal values would all be 1.0.

As shown above, this analysis method is expanded for all 757 rounds of the Textilene 80 Black CFS. Again the r^2 and slopes values are plotted in Figures 6.19 and 6.20 respectively. Notice the very close agreement of r^2 values and 1.0. This indicates there is a strong correlation between the measured, i_m, and simulated, $i_{m,s}$, horizontal illuminance values for the entire simulation duration of 757 rounds. With respect to the slopes, the agreement is also quite good, however, notice a persistence of the values to be above 1.0. This universal valuation of greater than 1.0 means the CUBE 2.0 system is under estimating the horizontal illuminance at the workplane height of 0.8 m as compared to the simulated value.

Possible explanations of this consistent under estimation of the horizontal illuminance value could be an incorrect BTDF which is allowing more luminous energy into the the model through the $L_{v,s}$ term than the actual measurement, L_v. Another possible cause is a consistent error in the vertical illuminance measurement coming from the LI-210 as

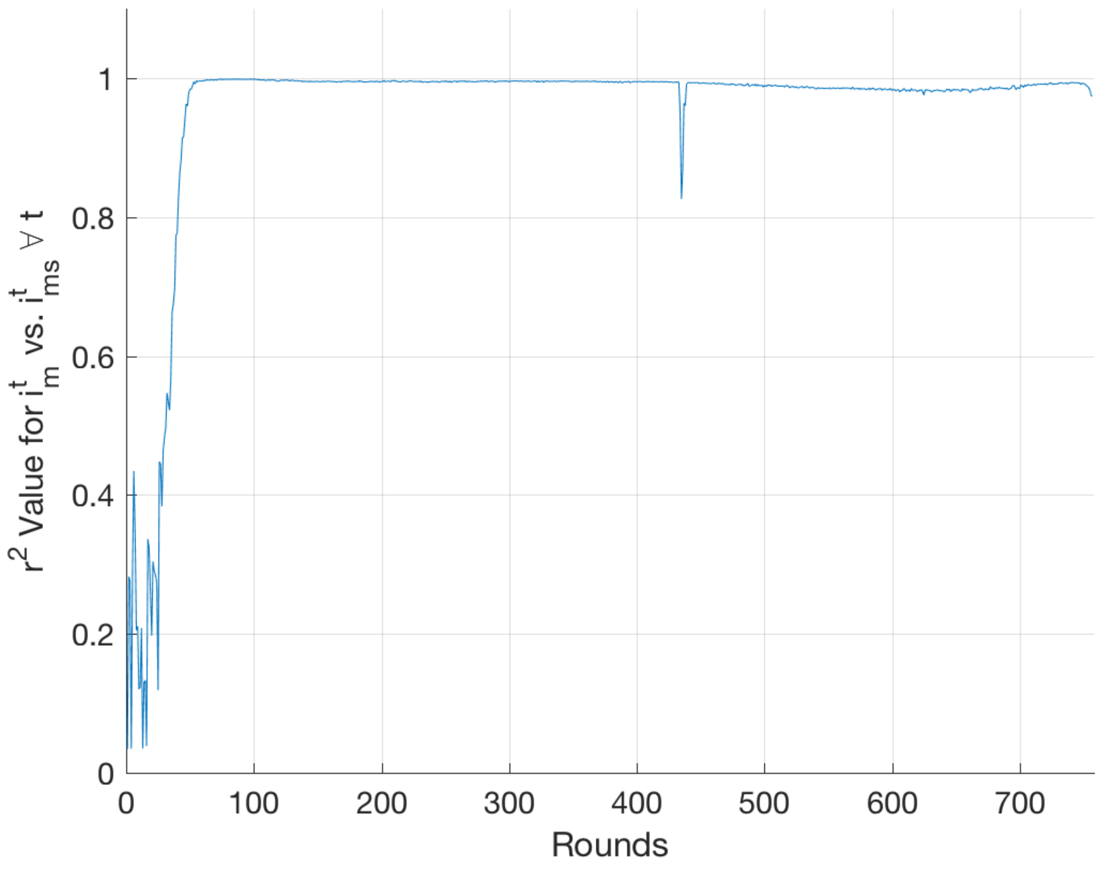


Figure 6.20: r^2 values for all the least square fit lines for i_m and $i_{m,s}$, note ideal values would all be 1.0.

compared to the calibration data sets. This would artificially inflate the E_v measurement which is used with the BTDF to create the $L_{v,s}$ measurement, which is the input to the Radiance horizontal illuminance measurements. Another possible explanation is there may exist stationary objects for this particular CUBE 2.0 orientation which happen to excite the Canon 6D and fisheye lens more than the CFS undertest and the optical fiber and RPiCM smapling system. While unlikely, this could be explored by painstakingly examining each Klems Basis input patch from the Canon 6D and examining the distributions with respect to the corresponding optical fiber locations within the Klems Basis Patches on the exiting side. Given the course discretization scheme of the CUBE 2.0 system, this is unlikely yet would require a large analysis for a small error which is already well within the accepted limits of the building daylight community.

To quantify this error more explicitly in terms of the horizontal illuminance values, the

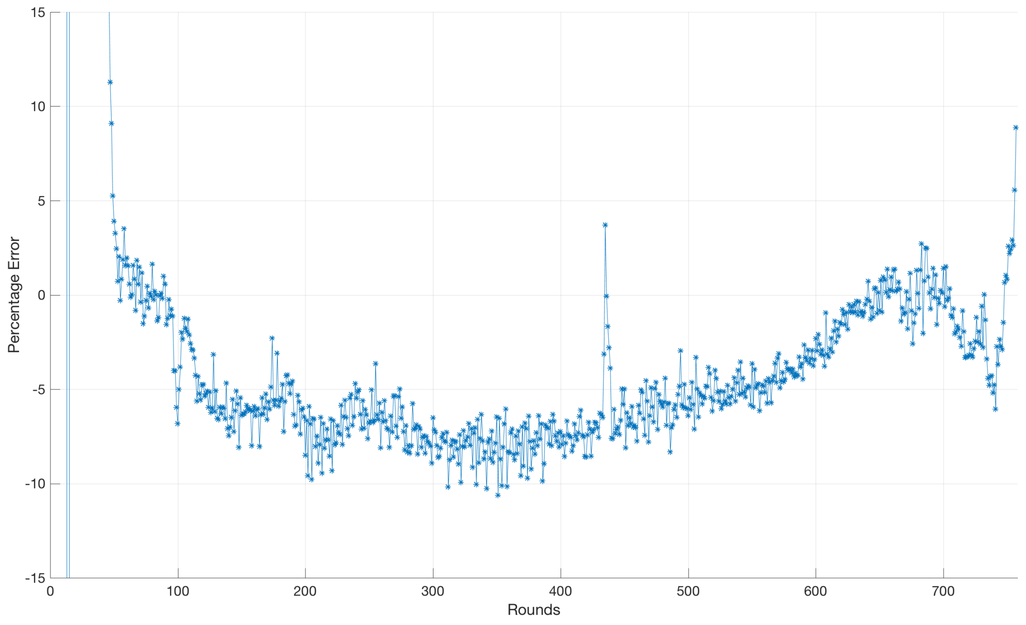


Figure 6.21: Mean percentage error per round of the horizontal illuminance at workplane height of 0.8 m.

mean percentage error over the entire grid of points is calculated. This percentage error is thus,

$$err^t = \frac{1}{35 \times 81} \sum_{g \in (81,35)} \left(\frac{i_m^{t,g} - i_{m,s}^{t,g}}{i_{m,s}^{t,g}} \right), \; \forall\, t \in \{1, \ldots, 757\}, \tag{6.2}$$

where for each round it is averaged into one number and plotted in Figure 6.21. Notice, the consistent negative percentage errors, meaning the CUBE 2.0 is under estimating the horizontal illuminance. The actual quantitative values, however, are between +3% and -10%. Given acceptable experimental measurements within the daylighting community for experimental work are typically ±20% [113, 46], this is exceptionally good.

This agreement means the CUBE 2.0 system is highly faithful in measuring the exiting luminance distribution coming out of a CFS under test and propagating this measured luminance energy into the Radiance model and simulating the building space for which the CFS daylighting system will be used.

As a point of question, one may inquire as to how the CUBE 2.0 improved in it's ability to simulate the behavior of the horizontal illuminance as compared to the exiting luminance distribution. To answer this, the simulation of the horizontal illuminance metrics can be thought of as a filter of sorts for the exiting luminance distribution as the light is bounced around the inside of the Reference Office. This filtering is what causes the improvement in the measured and simulated values of i_m versus $i_{m,s}$ with respect to L_v versus $L_{v,s}$ from above. The already close qualitative agreement of the exiting luminance distribution measurement and simulation are thus made closer by this filtering process as the higher and lower values are averaged together.

Final Discussion Points

Two final points are now mentioned to clarify artifacts in the data which have yet to be mentioned. The first of which is the consistent failure of the first roughly 50 time steps in the analysis discussed so far. This poor behaviour comes down to the fact that the CUBE 2.0 system is running per the simulation starting conditions of Nautical Twilight, yet there is very little light actually exciting it. Hence, the values are all quite small, and thus error and percentage errors are perceived to be quite large, yet the actual luminous energy associated with these time steps is very little. Hence, these beginning time steps, which can be though of a post-Nautical Twilight, pre-sunrise time steps, are included for completeness, yet considered to be a non-issue with respect to the CUBE 2.0 and its validation as these time steps would hardly be used and have very small impact.

The second point addresses the exiting luminance distribution. Upon careful inspection one will notice a large spike in the measured L_v as compared to $L_{v,s}$ occurring at approximately round 436 (2:50 pm) for Textilene 80 Black. This spike is quite strange, however, it is believed to be specular reflection directly from the solar disk shinning onto some mechanical component of the roof of Cory Hall, home of the Department of Electrical Engineering & Computer Sciences at UC Berkeley, where the tests were being conducted. This specular reflection is believed to have influenced the CFS test aperture, hence was measured by the optical fiber and RPiCM, yet didn't shine on the Canon 6D. As evidence of this, Figure 6.22 shows $LDRI_{EC=0}$ from the $LDRI^{11}$ for that time step. As can be seen, the shadow is directly in front of the CUBE 2.0, hence the solar disk is directly behind, making the geometry possible for such a specular reflection. Further, when examined on a sun chart, the sun is indeed directly behind the CUBE 2.0 at this time in the simulation. While this is not proof, and the explanation will never be known for sure, given a similar spike occurs at time step 420 (2:50 pm) for Shade View Ebony, presented further below, at the same time in daily back to back measurements, the author believes this is quite likely. In a silver lining to this effect in the results, it in fact confirms the analysis put forth in Appendix F which indicates direct solar radiation and strong specular reflections can affect the $L(\theta, \phi)_{cfs}$ and $L(\theta, \phi)_{fish}$ differently. Notice below in the TCP data this artifact is missing. This is consistent with the proposed explanation, as the TCP simulation was conducted at a later date and the solar disk geometry no longer caused this effect, yet the CUBE 2.0 was in the same location

Shade View Ebony: While no simulated data can be used as a comparison, intuition and relative performance to Textilene 80 Black can be used to gauge the CUBE 2.0 simulation. Beginning with the exiting luminance distribution, L_v, shown in Figure 6.10, one can clearly see a similar qualitative behavior to that of the Textilene 80 Black. This makes sense as they are both rectangular patterned woven fabric type shades. Further, both days

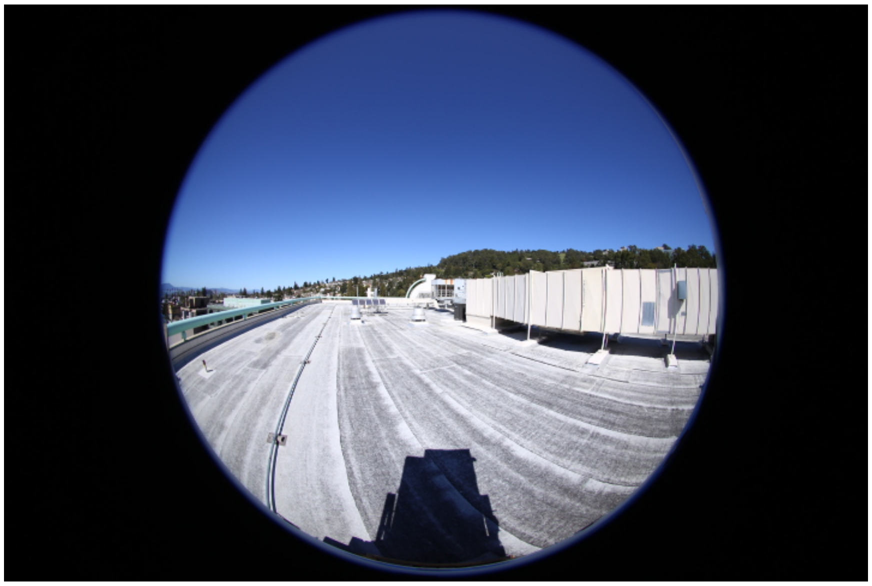


Figure 6.22: The properly exposed LDRI from the $LDRI^{11}$ showing the directly orthogonal CUBE 2.0 shadow from the CFS test aperture.

exhibited clear sky conditions, meaning their excitations were similar. Further, as can be seen in Figure 6.1 and 6.2, the openings are smaller in Shade View Ebony than the Textilene 80 Black. This is born out in the exiting luminance distribution as a reduced maximum luminance value down from about 6000 to 4000 [cd/m^2].

This overall reduced magnitude luminance is also shown in the horizontal illuminance plot oriented as vectors for each round which can be found in Figure 6.11. Finally, shown in Figure 6.12 for time step t=343 is the horizontal illuminance as calculated by the Radiance model.

6.1.2 Translucent Concrete Panels

This unique building daylighting system comprises optical fibers embedded into a concrete matrix, see Figure 6.23. These panels are then installed as part of a building's enclosure system allowing daylight to transfer through the building envelope in places where it would not otherwise enter the space. That is, TCPs are an alternative to what would traditionally be opaque building enclosure sections. With favorable properties such as structural loading capabilities and low thermal transfer, TCPs are a multifunctional building envelope system which can be used for greater energy efficiency within buildings, yet allow for favorable occupant experiences through the transmission of daylight into a space.

Much work has been done on TCPs, being the focus of two dissertations, both the Casquero-Modrega Dissertation [30] and the Ahuja Dissertation [4], and a pending patent

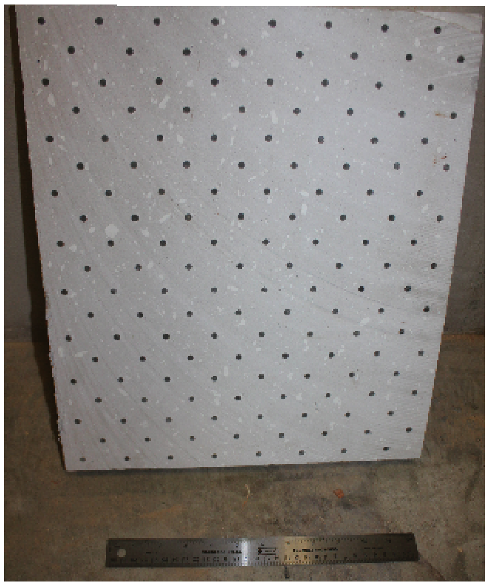


Figure 6.23: A single Translucent Concrete Panel (TCP).

[23]. This work compliments the existing efforts, but has been conducted independently at a later time.

Simulation Setup

Similarly to the Twitchell Textilene® shades which were tested above, the TCP was subjected to a day long simulation, that is, from Nautical Twilight start to end, at a time step equal to one minute. This involves measuring the input illuminance distribution (E_v), exiting luminance distribution (L_v), and computing the associated luminous metrics as part of the Radiance simulation (i_m). Note, TCPs have not been tested on a goniophotometer, hence no BTDF exists, thus comparisons to simulated values and the associated errors cannot be computed as done with the Textilene 80 Black CFS.

One should note, a full TCP was not hung from the CUBE 2.0 to conduct this test. Rather, a single optical fiber of the same variety as in the panels themselves was tested. It had similar length and mounting properties to ensure similar behavior to the real TCP system. By keeping the CFS test aperture area (which is kept opaque except for the optical fiber) to optical fiber area ratio proportional to the optical fiber area, concrete matrix area ratio of that in a TCP, the three-phase method can still effectively be used. That is, the same Reference Office [126] with the glazing replaced by TCPs can be modeled using the measured exiting luminance distribution, L_v, gathered in the CUBE 2.0.

Table 6.3: Outline of the simulation for Translucent Concrete Panels.

Round Number:	Time of Day:	Significance
1	5:50 am	Start
13	6:31 am	1 Min. Time Steps Begin
32	6:49 am	Sunrise
791	7:35 pm	Sunset
808	7:53 pm	1 Min. Time Steps End
820	8:33 pm	Stop

Results: Translucent Concrete Panel

The simulation involving the TCPs was conducted on April 4, 2017 from Nautical sunrise (5:51 am) to sunset (8:33 pm). At one minute intervals this resulted in 820 total time steps. Recall, at low light conditions a step needs more than one minute to execute, hence 820 steps, instead of the calculated 882 steps, comprise the simulation. Note, however, the vast majority of the simulation time horizon, and from a magnitude perspective, the meaningful simulation duration, all have time steps of exactly one minute duration.

First, in Figure 6.24 the exiting luminance distribution as measured by the CUBE 2.0 system, L_v, is presented. Next, in Figure 6.25 the Radiance model output of i_m is presented.

Finally, to add to the intuition of the system, a plan view of the Reference Office showing calculated horizontal illuminance on the 0.8 m workplane for the Shade View Ebony sample, i_m, at time step t=344 (12:05 pm) is presented in Figure 6.26.

Results Discussion

Again, as with the Textilene 80 Black and Shade View Ebony shades, intuition is matched regarding the illuminance distribution being maximum near the CFS and decreasing as one travels deeper into the floor plan perpendicular to the wall with the CFS. While no comparison can be given with respect to simulated values within the CUBE 2.0 itself, intuition regarding the horizontal illuminance values is well confirmed. That is, the maximum horizontal illuminance comes in at approximately 240 lux. Given the CFS undertest is an enclosure comprising only TCPs, meaning an array of optical fibers is the only source of illuminance within the space, this seems quite reasonable.

While 240 lux is a low level for desk work, many other tasks exist which could be performed well with this magnitude of light. Examples include desk work involving a backlit computer screen or hallways with load bearing wall requirements on one side, yet natural light requirements. This thesis, however, is not so much concerned in studying and justifying TCPs, but rather on the testing of these systems in a hardware-in-the-loop architecture.

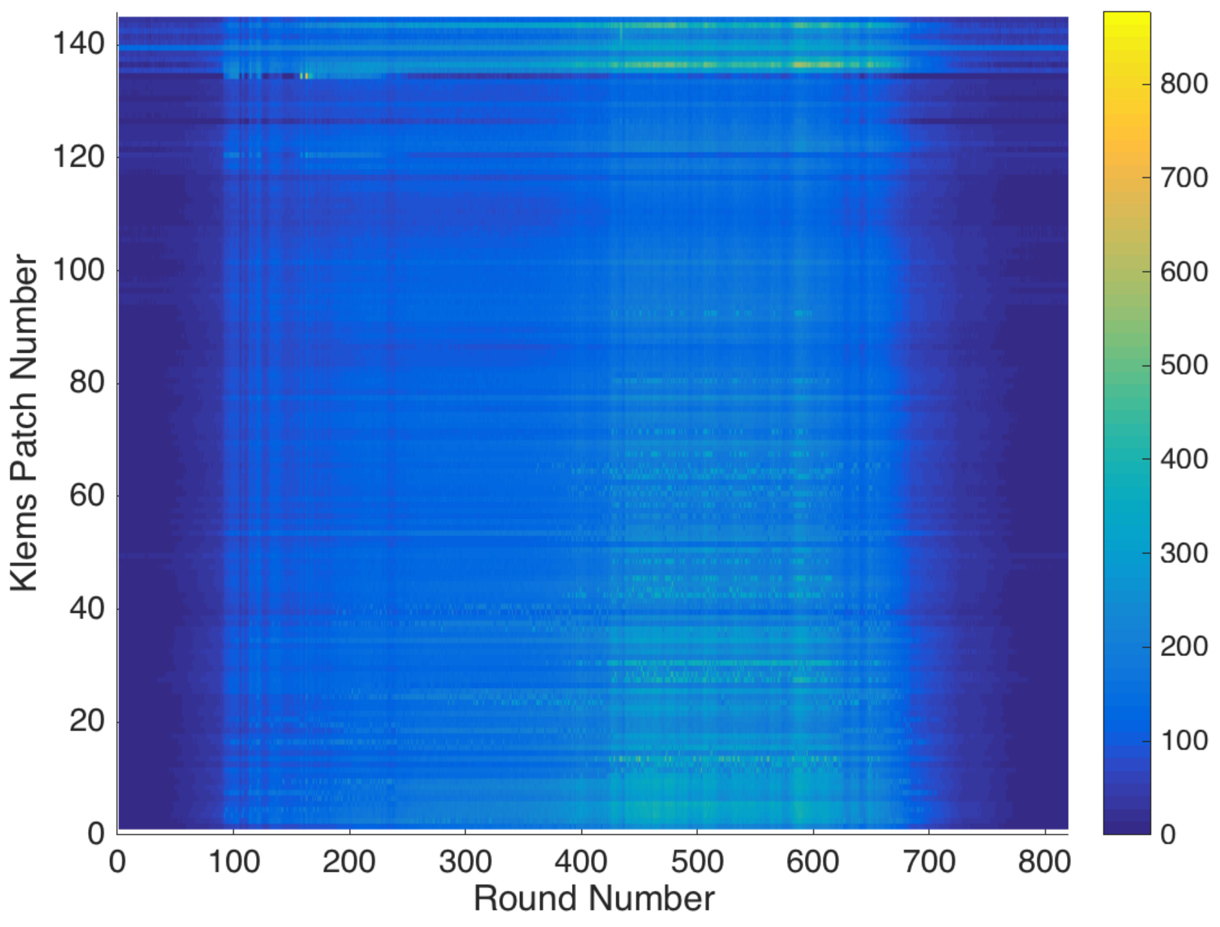


Figure 6.24: Exiting luminance distribution, L_v^t, for all $t \in \{1, \ldots, 820\}$ for TCP, in units of $\frac{cd}{m^2}$.

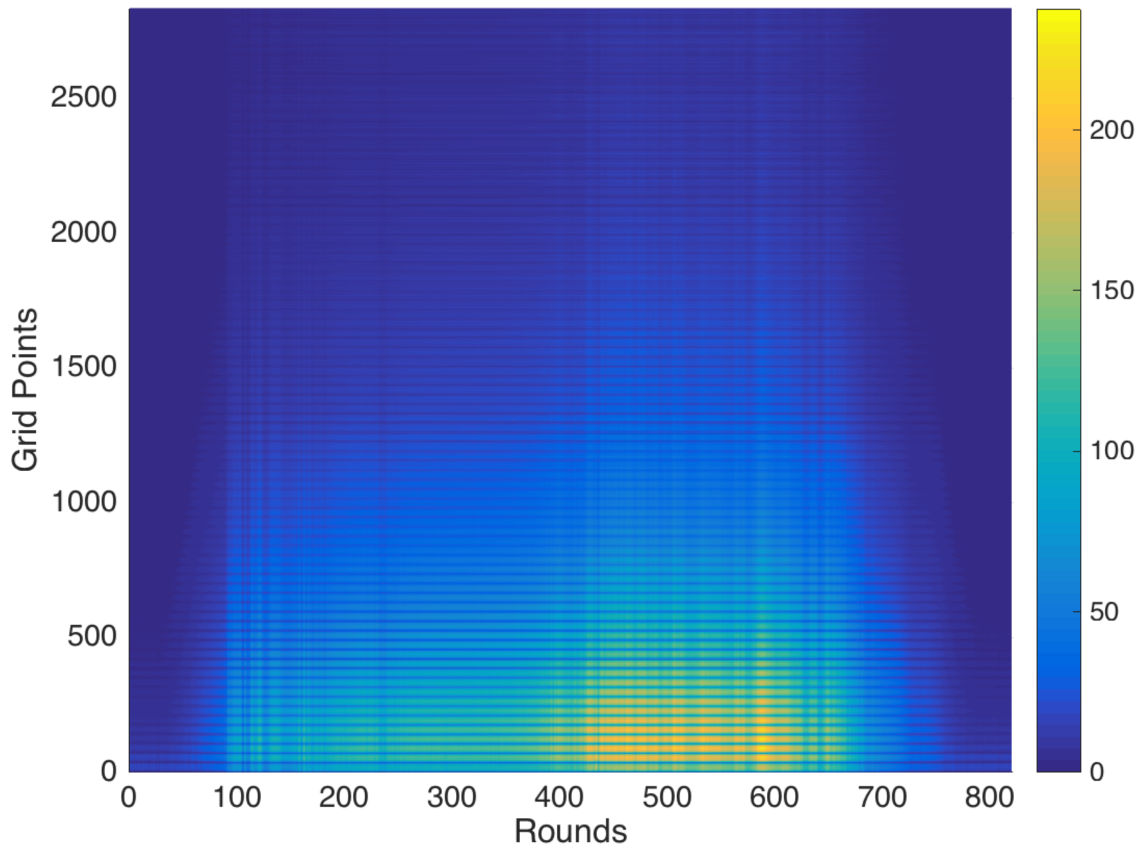


Figure 6.25: Horizontal illuminance at 0.8 m workplane height, i_m^t, for all $t \in \{1, \ldots, 820\}$, oriented as a vector, for TCP, in units of [lux].

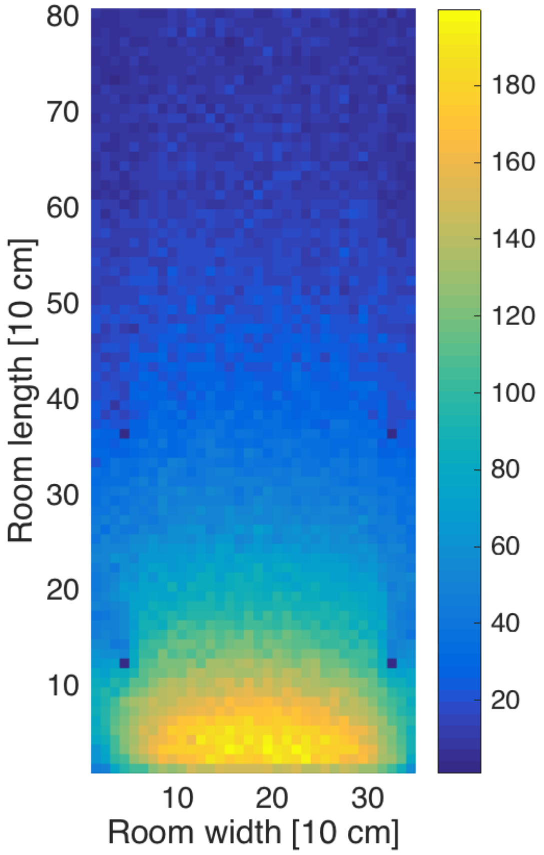


Figure 6.26: Horizontal illuminance at 0.8 m workplane height, i_m^t, for time step t=344, for TCP, in units of [lux].

Chapter 7

Conclusions and Future Directions

In chapter 6, results from HWiL simulations conducted by the CUBE 2.0 were presented. Where applicable, these results were compared to previously measured complex fenestration system (CFS) properties. This lead to the conclusion that the CUBE 2.0 is faithfully representing building daylighting systems within industry acceptable magnitudes.

In this chapter, conclusions regarding the CUBE 2.0 system in total are presented. First, the currently implemented CUBE 2.0 is discussed with remarks concerning its hardware-in-the-loop nature emphasized. Second, attention is turned to future directions for the CUBE 2.0 with ideas presented to improve the current functionality as well as extend the functionality to include additional features, such as a coupled thermal model of the space being studied.

7.1 Conclusions from the Current System

7.1.1 Background

The work described in this thesis started with the abstract goal of "studying energy use in buildings." With this highlevel motivation, work began by investigating the building industry, with a primary focus on commercial buildings.

During this investigation, it was determined that the sun's impact on the built environment is quite substantial. This impact comes in the form of influencing the energy used by a building's systems to maintain indoor environment quality, as well as the actual user visual experience with respect to daylighting. Noticing the sun is impactful has led to much research in the area of controlling sunlight within buildings. One class of solutions which has seen significant attention with this regard is complex fenestration systems (CFS).

In order to analyze CFSs within buildings for design purposes, several methods can be applied, yet there still exist shortcomings for these systems. This thesis presents a new method for analyzing CFSs with respect to building daylighting systems. The hardware-in-the-loop (HWiL) modeling and simulation approach described herein answers some of these shortcomings in the current building daylighting analysis techniques. The HWiL model for daylighting systems is refereed to here as the CUBE 2.0 system.

Conclusions

From this initial investigation it was concluded there was in fact a potential for hardware-in-the-loop testing techniques to be employed with the building performance simulation community for a more optimal representation of certain complex fenestration systems.

7.1.2 HWiL Model Overview: The CUBE 2.0

The proposed hardware-in-the-loop model, referred to as the CUBE 2.0, can be broken down into two classes of components: physical components and cyber components. Physical components exist in the physical world. These components consist of the lighting excitation, the actual CFS specimen being tested, and measurement systems for both the exiting luminance distribution from the panel, L_v, and the input illuminance distribution to the panel, E_v. The second class of components, the cyber components, are those components consisting of elements existing in the digital world. Here, these components are the actual indoor building spaces which are being "day lit" by the physical specimen being excited and modeled using Radiance. In reality, this involves many pieces of code not only simulating the indoor luminous model, but also coordinating the timing and execution of the measurements, E_v and L_v, mentioned above.

Conclusions

From this breakdown, the HWiL model and prospective simulations took shape in an explicit form. Seeing that it is based on the preexisting three-phase method, great confidence is given to the conclusion the system's modeling semantics are in fact valid.

7.1.3 Designing the CUBE 2.0

Various design analyses were conducted in order to build the CUBE 2.0 system. The applied analysis techniques involved all types of investigation from simple "intuition" with respect to the sizing of bolts, to detailed network analysis to ensure the CUBE 2.0 system operates without deadlock and other potentially hazardous situations. Further, empirical investigations were also conducted to determine several system parameters as well as confirm the operation of various components.

Conclusions

These various analysis techniques allowed for a high confidence conclusion that the proposed CUBE 2.0 system could in fact be built and operate in a manner which would allow for true HWiL modeling and simulation for daylighting analysis in the building design community.

7.1.4 Building the CUBE 2.0

Construction of the CUBE 2.0 system was conducted in two distinct phases: first concerning the physical components and second concerning the cyber components. While many physical components were taken "off the shelf," for example the Raspberry Pi and Camera Module, the CUBE 2.0 system is a fully custom machine in its own right. The construction process required some iteration of certain physical components which were determined unsatisfactory in their initial design and implementation, however, the vast majority of the CUBE 2.0 system was determined sufficient when actually realized in physical form. Further, the cyber components were written in code using the languages of Python, Matlab, and C, using both custom made functionality as well as prebuilt libraries.

Conclusions

Overall, lessons learned from the construction process of the CUBE 2.0 can be summarized with respect to not only the two domains, but also their intersection.

First, while design techniques concerning the physical world have developed greatly, whether they be structural engineering models or detailed analysis with respect to daylight, they still often require modification once realized in physical form. This comes in many situations, for example, overlooking certain physical constraints or making bad modeling assumptions. One example from the CUBE 2.0 is the 8020 frame needed to be notched slightly for the very bottom optical fiber which samples Klems Basis Patch 137. The exact spatial geometry of the optical fiber array was never fully realized, but assumed to fit where the error was made.

Second, the cyber domain often requires debugging as mistakes can be made by the programmer as well as the code library documentation errors exist. While code has a faster turn around time for correction than physical systems, often frustrating ordeals can be experienced due to both documentation mistakes and "low-level" library implementation errors. An example of an implementation error in a library encountered with the CUBE 2.0 system was in writing the program fisheye6D. In this execution, nearly four days were spent debugging C code when it finally emerged in the online community the latest software framework had in fact an error. The CUBE 2.0 fisheye6D code was correct as written, but the API had errors within itself which the manufacturer quickly corrected.

Finally, the interaction of the physical world and the cyber world requires careful coordination. That is, all details of the code must be parameterized correctly as the physical system is built. An error occurred in the construction of the CUBE 2.0 which took two days to realize and correct involved one line in activating the RPiCM. The single line horizontally flipped the picture causing confusion in the processing which was very tedious to debug, yet extremely easy to correct. Simply delete the unneeded line which horizontally flipped the photograph.

Thus, conclusions from the construction of cyber-physical systems, take three forms. Those associated purely with the physical system construction, those associated purely with the cyber components of the system, and finally those dealing with the intersection of the physical and cyber components.

7.1.5 Calibrating the CUBE 2.0

With the CUBE 2.0 built, the device was calibrated to produce measurements in meaningful SI units. During the calibration much intuition was learned concerning the actual limits of sensors with respect to luminous phenomenon. Both in the simple measurement of vertical illuminance using the Li-Cor 210 and using the Canon 6D with a fisheye lens, sensors for light are very sensitive to changes. These changes are often tough to realize as the human eye is a very robust sensor. Hence while the human brain and eye are compensating for changes in dynamic range, or "integrating over" pulsating light sources (amongst many other issues your visual system simply "handles"), sensors often pick these changes up, producing non-intuitive readings. Further, built in error limitations can cause frustrating results in the learning process as well.

Conclusions

Hence, in conclusion, optimism as well as a knowledge of the reasonably attainable accuracy with the sensors being utilized are critical to a successful system implementation.

7.1.6 Simulations with the CUBE 2.0

Finally, the CUBE 2.0 has been demonstrated capable of performing daylong simulations repeatedly, with both commercially available CFSs (Textilene 80 Black, Shade View Ebony) as well as custom made CFSs (translucent concrete panels). With this base of experiments, the CUBE 2.0 can now be used with high confidence for novel CFS which are being developed with rapid prototyping techniques, such as 3D printing, as well as the numerous unmeasured CFSs already commercially available. While not demonstrated here, the CUBE 2.0 also has the ability to "close-the-loop" with respect to augmentable CFS. This functionality is realized through standard communications protocols (e.g. PWM, I^2C), as the actual augmentation is the responsibility of the CFS designer. With these capabilities confirmed, the CUBE 2.0 system is now complete for daylighting studies within buildings.

Conclusions

The CUBE 2.0 system can thus indeed be recognized as a true hardware-in-the-loop model of building daylighting systems. The first of it's kind known by the author. Currently, the

CUBE 2.0 is being used to investigate other novel CFS systems with additional work being focused toward the concept of system identification.

7.2 Future Directions

While the CUBE 2.0 is a stand alone system for daylighting analysis using a hardware-in-the-loop modeling and simulation approach, many other ideas and extensions can be derived from this initial work. These ideas are presented next in a high level conceptual form in hopes that future researchers can continue this work.

These ideas not only involve improving the CUBE 2.0 system to attain more confident measurements, but also offer the possibility to extend the project to other experimental goals beyond daylighting, for example including thermal as well as human interaction within the simulations. Given the "indoor space" of the building is computational in nature (i.e. in the cyber-domain), vast freedom can be given to researchers in terms of what to simulate. This is one of the main advantages of HWiL modeling and simulations.

7.2.1 Artificial Light Source Excitation

The CUBE 2.0 is used exclusively in this thesis with real sunlight over daylight simulations. While the information gained from a series of daylight simulations can be quite telling, this fundamentally limits the applicability to the daylighting conditions experienced during those simulations. To answer this, it is suggested to use the CUBE 2.0 within daylight simulators capable of generating various lighting distributions to represent other times and locations throughout the year and Earth. This would allow for a more representative exploration of the CFSs measured on the CUBE 2.0.

A major advantage of this type of analysis would be the drastic increase in the speed of the simulations. While daylong deployments can only move as fast as the sun, due to the nearly instant settling time of daylighting, the only limiting factor for time step size would be how fast conditions can be set and measurements can be taken. An example would be the Daylight Emulator (DE) as part of the Singapore-Berkeley Building Efficiency and Sustainability in the Tropics (SinBerBEST). As part of a different project, the author built an API to control the DE, which can be integrated into the main() program. Hence, the main() project could in fact "control the sun" from its perspective, allowing many daylighting conditions to be realized quickly, and thus making yearlong daylighting simulations practical in as little as a day or two.

Other light sources would be heliodons for direct daylighting component study, overcast sky simulators for uniform excitement distributions, and sky domes for completely custom excitement distributions.

7.2.2 Finer Discretization of Exiting Luminance Measurement

One major limitation of the CUBE 2.0 system is the relatively large discretization of the exiting luminance distribution. This discretization arises from the finite points at which the optical fibers "sample" the exiting luminance distribution. That is, in the current implementation there are 145 $\frac{1}{4}''$ diameter optical fibers, each located at the centroid of a Klems Basis Patch. While 145 may sound like a lot, the vast majority of the exiting hemisphere is in fact "un-sampled" and therefore totally un-impactful in the measurement of light leaving the CFS under test into the building. While most building daylighting systems have smooth exiting luminance distributions, minimizing the chance of missing meaningful light behavior when using interpolation, there is a chance of missing impactful luminance energy leaving a panel. As such, increasing this discretization would be very beneficial to attain more representative measurements.

Several possibilities exist, some from the literature and others novel ideas from the author. One idea which been demonstrated in Andersen's work [6] is the possibility of using a *projection screen* of sorts of which HDRI measurements can be taken. Orientation of the screens and solving for the projection formula associated with these screens would be the difficult part, still further, inter-reflection could also be an issue. Another idea originally proposed by Ward [156] and also implemented by Andersen [6] is the idea of using a hemisphere projection screen on the inside of the CFS under test. This setup may have issues with the camera interfering with the input illuminance distribution. Perhaps small embedded cameras could be placed inside the hemisphere, each responsible for a different portion of the hemisphere. Inter-reflection could still be an issue here. Another idea proposed by the author but abandoned due to cost contraints is to laser cut a very large number of closely spaced holes or slots in an acrylic hemisphere on the exiting luminance side of the CFS under test. These holes, the diameter of which would need to be determined, however, $\frac{1}{16}''$ is thought a good starting point, would be oriented in such a way as to reflect the exiting luminance measurement toward a digital camera for a HDRI measurement. The acrylic would need to be some opaque material allowing for no transmission, rather all reflection and absorption of the incident rays. Issues included here are complex machining or lazer cutting related (i.e. can this reflection hemisphere even be built), polarization of the light caused by the reflection, and solving for the associated projection formula with respect to the CFS undertest, reflection hemisphere, and digital camera.

Other ideas exist, however, the author would suggest staying away from the "simple solution" of increasing the number of optical fibers. While tempting, it is the author's experience that 145 optical fibers is near the upper end of what is practically manageable given the techniques utilized in CUBE 2.0. Even a simple doubling of the optical fibers to 290 is thought to add a vastly disproportional amount of work compared to the gained resolution with respect to the exiting luminance hemisphere.

7.2.3 Input Illuminance Measurement

While analysis was conducted using mathematical models and experimentally, and the errors were determined manageable, it is an inescapable fact that the input illuminance distribution measurement done by the Canon 6D and fisheye lens are not exact. Improving this measurement, however, is not easy. One idea would be to use an array of embedded cameras around the CFS test aperture from which a stitched measurement of the input illuminance distribution is derived. Another idea would be to add additional Canon 6D and fisheye lens setups to the sides of the CUBE 2.0. These two additional cameras could offer a better estimate using interpolation techniques than the one camera used currently.

In addition to this issue, the current CUBE 2.0 system is not calibrated for direct sunlight. Part of this is due to the fact that measuring the solar disk with a digital camera is highly challenging. Using neutral density filters could be one solution for this which should be explored. One problem with this, however, will be the course discretization measurement of the exiting luminance distribution. With direct daylighting now being possible, the chances increase the sampling ends of the optical fibers will miss a meaningful component in measuring the exiting luminance distribution.

7.2.4 System Identification for BTDF

While the idea is outlined here at a high level in Appendix C, the current data sets, as well as new data sets, are far from begin fulling explored. It should be noted, the original concept for system ID with respect to BTDFs was originally proposed by Professor Edward A. Lee.

7.2.5 Adding Thermal Model

While the CUBE 2.0 is solely capable of modeling and simulating daylighting physics, it is well understood the impacts of the daylight have impacts far beyond occupant visual experience. The primary alternative impact is the thermal impact solar gains. In fact, the total solar physics is coupled with the daylighting physics in buildings due to the fact that not only is daylighting carrying thermal energy, but also the amount of daylighting can affect the total thermal load in a space through the supplementary use of electric luminaires. Meaning the heat load given off by electric luminaires must be removed if the building is in cooling mode and adds benefits if the building is in heating mode.

The physics of this interaction can become quite complicated, and in fact are the main goals of so called "building performance simulation" packages such as Energy Plus [29]. An extension of the CUBE 2.0 could be to build a thermal model which can take as input the exiting luminance distribution converted to an exiting radiance distribution. Several challenges would need to be overcome, for example how to solve for initial conditions within the thermal model, what programatic interface to couple the thermal model with the measured data, what building performance simulation package to implement this model in.

Converting from radiance to luminance is straight forward as it is simply a wavelength dependent filtering. However, converting from luminance to radiance is complicated and must be based on some assumption. One can make more informed assumptions with the simple addition of an irradiance sensor to compliment the LI-210 illuminance sensor. Further, one could us digital cameras with no infrared filters, offering insights into the spectral content of the exiting light. One note of caution, the optical fibers don't transmit all wavelengths equally, hence a simple scaling will not work for this luminance to radiance conversion.

It is recommended by the author to use the Modelica Buildings Library [159] for any thermal models built. Modelica is a declarative language allowing for complete investigation and customization of the existing code. This will be necessary to be able to control how the measured radiance interacts with the building fabric at a fine grained level. Further, the exciting work done with functional mock-up units (FMUs) is believed by the author to be a perfect interface. With FMUs export built into the popular Modelica simulation engine Dymola, very complex models can be realized with simple interfaces to the CUBE 2.0. As further motivation, co-simulation work in general is an open research question with some work done in the building performance simulation community [24, 111]. Hence, interesting questions may be generated in this investigation.

Further, incorporating humans into simulations [62], especially with respect to visual comfort and it's impact on the electric luminaire load of a space, could be very interesting work. Especially interesting problems could be realized if augmentable CFSs are used with both human activated and automated control schemes.

7.2.6 Final Comments

While nearly limitless extensions to the CUBE 2.0 to improvement measurements or extend the physics to include thermal response, human interaction, and others can be done, one should ask how does this benefit the engineer or building designer over the current approach. The author believes the case has been made with respect to complex fenestration systems (CFS) and daylighting analysis given the current literature. However, the thermal model literature is complimentary and often quite adjacent to that of daylighting. Hence, due diligence with respect to an extensive literature review should be undertaken.

To begin this search, the second edition of the seminal work "Energy Simulation in Building Design" by Clarke [34] is a solid start. Next, or better yet concurrently, "Building Performance Simulation for Design and Operation" [61] is a must read and will provide context for all building performance simulations in use today. Finally, the Journal of Building Performance Simulation by Taylor and Francis, Building Simulation by Springer, Lighting Research & Technology by Sage, and Leukos by Taylor and Francis are all excellent journals from the field to explore published research. In addition, Buildings and Environment as well as Energy and Buildings, both by Elsevier, are journals with many papers worth exploring. Finally, the largest group of building performance simulation professionals (researchers and practitioners) in the world is the International Building Performance Simulation Association (IBPSA). It is structured by international, national, and regional chapters with an international conference held every other year. The American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE) also is worth exploring.

Hardware-in-the-loop modeling and simulation is truly an ingenious investigation technique. Self aware engineers must ensure, however, they don't become hammers looking for nails.

Bibliography

[1] *80/20 Inc.* https://www.8020.net/. [Online; accessed 14-February-2017]. 2017.

[2] Ansel Adams. "The Negative: Exposure and Development Basic Photo 2". In: *Morgan and Lester* 98 (1948).

[3] Aashish Ahuja, Khalid M Mosalam, and Tarek I Zohdi. "Computational Modeling of Translucent Concrete Panels". In: *Journal of Architectural Engineering* 21.2 (2014), B4014008.

[4] Ashish Ahuja. "Simulations of Innovative Solutions for Energy Efficient Building Facades". In: (2015).

[5] *Air Speed-Temperature-Humidity-Light Meter with "K" Port.* https://www.generaltools.com/air-speed-temperature-humidity-light-meter-with-k-port/. [Online; accessed 3-February-2017]. 2017.

[6] Marilyne Andersen. "Innovative Bidirectional Video-Goniophotometer for Advanced Fenestration Systems". PhD thesis. École Polytechnique Fédérale de Lausanne (EPFL), 2004.

[7] Marilyne Andersen. "Validation of the performance of a new bidirectional video-goniophotometer". In: *Lighting Research & Technology* 38.4 (2006), pp. 295–311.

[8] Marilyne Andersen, Michael Rubin, and Jean-Louis Scartezzini. "Comparison between ray-tracing simulations and bi-directional transmission measurements on prismatic glazing". In: *Solar Energy* 74.2 (2003), pp. 157–173.

[9] Marilyne Andersen et al. "Bi-directional transmission properties of venetian blinds: experimental assessment compared to ray-tracing calculations". In: *Solar Energy* 78.2 (2005), pp. 187–198.

[10] Marilyne Andersen et al. "Experimental assessment of bi-directional transmission distribution functions using digital imaging techniques". In: *Energy and Buildings* 33.5 (2001), pp. 417–431.

[11] Marilyne Andersen et al. "Using digital imaging to assess spectral solar-optical properties of complex fenestration materials: A new approach in video–goniophotometry". In: *Solar Energy* 84.4 (2010), pp. 549–562.

[12] *Standard File Format for the Electronic Transfer of Photometric Data*. standard. Illuminating Engineering Society, 2002.

[13] *Anyhere Software*. http://www.anyhere.com/. [Online; accessed 17-February-2017]. 2017.

[14] *Anyhere Software: Download command-line HDR image builder for Mac OS X*. http://www.anyhere.com/. [Online; accessed 24-March-2017]. 2017.

[15] MBC Aries, MPJ Aarts, and J Van Hoof. "Daylight and health: A review of the evidence and consequences for the built environment". In: *Lighting Research and Technology* 47.1 (2015), pp. 6–27.

[16] Ian Ashdown. "Near-field photometry in practice". In: *IESNA Annual Conference Technical Papers*. 1993, pp. 413–425.

[17] F O Bartell, EL Dereniak, and WL Wolfe. "The theory and measurement of bidirectional reflectance distribution function (BRDF) and bidirectional transmittance distribution function (BTDF)". In: *1980 Huntsville Technical Symposium*. International Society for Optics and Photonics. 1981, pp. 154–160.

[18] *Bentham Instruments Limited. PVE300 Photovoltaic Spectral Response*. http://www.bentham.co.uk/pdf/PVE300.pdf/. [Online; accessed 30-August-2015]. 2017.

[19] *Berkeley Lab WINDOW*. https://windows.lbl.gov/software/window/window.html/. [Online; accessed 31-January-2017]. 2016.

[20] F Bettonvil. "Fisheye lenses". In: *WGN, Journal of the International Meteor Organization* 33 (2005), pp. 9–14.

[21] Magali Bodart et al. "A guide for building daylight scale models". In: *Architectural Science Review* 50.1 (2007), pp. 31–36.

[22] J Breitenbach and JLJ Rosenfeld. "Goniospectrometer measurements of the optical performance of a holographic optical element". In: *Solar energy* 68.5 (2000), pp. 427–437.

[23] *BRIGHT: Building With Radiant And Insulated Green Harvesting Technology*. https://techtransfer.universityofcalifornia.edu/NCD/25071.html/. [Online; accessed 3-April-2017]. 2015.

[24] David Broman et al. "Requirements for hybrid cosimulation standards". In: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM. 2015, pp. 179–188.

[25] Bruno Bueno et al. "A Radiance-Based Building Energy Model to Evaluate the Performance of Complex Fenestration Systems". In: ASHRAE/IBPSA-USA, 2014.

[26] *Building Energy Software Tools (BEST) Directory*. http://www.buildingenergysoftwaretools.com/home/. [Online; accessed 31-January-2017]. 2017.

[27] *Canon: Digital Imaging Developer Programme.* https://www.didp.canon-europa.com/. [Online; accessed 14-February-2017]. 2017.

[28] *Canon - EOS 6D.* https://www.usa.canon.com/internet/portal/us/home/products/details/cameras/dslr/eos-6d/. [Online; accessed 17-February-2017]. 2017.

[29] William L Carroll and Robert J Hitchcock. "Delight2 daylighting analysis in energy plus: Integration and preliminary user results". In: *Lawrence Berkeley National Laboratory* (2005).

[30] Núria Casquero Modrego. "Daylight performance assessment of an innovative energy efficient building envelope". In: (2016).

[31] Coralie Cauwerts, Magali Bodart PhD, and Arnaud Deneyer. "Comparison of the vignetting effects of two identical fisheye lenses". In: *LEUKOS* 8.3 (2012), pp. 181–203.

[32] Francois E Cellier and Jurgen Greifeneder. *Continuous System Mdeling.* Springer Science and Business Media, 2013.

[33] Francois E Cellier and Ernesto Kofman. *Continuous System Simulation.* Springer Science & Business Media, 2006.

[34] Joe A Clarke. *Energy simulation in building design.* Routledge, 2001.

[35] *Complex Glazing and Shading Database (CGDB).* http://windowoptics.lbl.gov/data/cgdb/. [Online; accessed 24-January-2017]. 2017.

[36] Jeff Conrad. *Exposure Metering Relating Subject Lighting to Film Exposure.* Tech. rep. 2003.

[37] John J. Conti and Paul D. Holtberg. *Annual Energy Outlook 2014 with projections to 2040.* Tech. rep. U.S. Energy Information Administration (EIA), Apr. 2014.

[38] Clayton T Crowe et al. *Engineering fluid mechanics.* 9th. Wiley, 2009.

[39] Stanislav Darula and Richard Kittler. "CIE general sky standard defining luminance distributions". In: *Proceedings eSim* (2002), pp. 11–13.

[40] Paul E Debevec and Jitendra Malik. "Recovering high dynamic range radiance maps from photographs". In: *ACM SIGGRAPH 2008 classes.* ACM. 2008, p. 31.

[41] David L DiLaura et al. "The lighting handbook: reference and application". In: (2011).

[42] Philip Dutre, Philippe Bekaert, and Kavita Bala. *Advanced global illumination.* CRC Press, 2016.

[43] L Edwards and Paul A Torcellini. *A literature review of the effects of natural light on building occupants.* National Renewable Energy Laboratory Golden, CO, 2002.

[44] *Emerging Objects.* http://www.emergingobjects.com/. [Online; accessed 12-February-2017]. 2017.

[45] Kelly MJ Farley and Jennifer Ann Veitch. "A room with a view: A review of the effects of windows on work and well-being". In: (2001).

[46] A Fisher. "Tolerances in lighting design". In: *Proceedings of the CIE Seminar on Computer Programs for light and lighting*. 1992.

[47] *FLEXLAB - The World's Most Advanced Building Efficiency Test Bed*. https://flexlab.lbl.gov/. [Online; accessed 2-February-2017]. 2017.

[48] Roger A Freedman. *Sears and Zemansky's University Physics*. Vol. 2. Pearson education, 2008.

[49] Peter Fritzson. *Introduction to modeling and simulation of technical and physical systems with Modelica*. John Wiley & Sons, 2011.

[50] *Gardens by the Bay*. http://www.gardensbythebay.com.sg/en.html/. [Online; accessed 2-February-2017]. 2017.

[51] PJ Gawthrop, MI Wallace, and DJ Wagg. "Bond-graph based substructuring of dynamical systems". In: *Earthquake engineering & structural dynamics* 34.6 (2005), pp. 687–703.

[52] Nicholas Gayeski, E Stokes, and Marilyne Andersen. "Using digital cameras as quasi-spectral radiometers to study complex fenestration systems". In: *Lighting Research & Technology* 41.1 (2009), pp. 7–25.

[53] *GOAL ZERO YETI 400 PORTABLE POWER STATION*. http://www.goalzero.com/p/165/goal-zero-yeti-400-portable-power-station/. [Online; accessed 11-March-2017]. 2017.

[54] Joseph W Goodman. *Introduction to Fourier optics*. Roberts and Company Publishers, 2005.

[55] *Grasshopper - Algorithmic Modeling for Rhino*. http://www.grasshopper3d.com/. [Online; accessed 3-February-2017]. 2017.

[56] Rob Guglielmetti, Shanti Pless, and Paul A Torcellini. "On the use of integrated daylighting and energy simulations to drive the design of a large net-zero energy office building". In: *IBPSA-USA Journal* 4.1 (2010), pp. 301–309.

[57] John Guild. "The colorimetric properties of the spectrum". In: *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* 230 (1932), pp. 149–187.

[58] Wyszecki Günter and S Styles. *Color science: Concepts and methods, quantitative data and formulae*. 1982.

[59] Harry G Harris and Gajanan Sabnis. *Structural modeling and experimental techniques*. CRC press, 1999.

[60] *[Hdri] HDRI capture program for Canon EOS cameras under Mac OS X*. http://www.radiance-online.org/pipermail/hdri/2006-January/000021.html/. [Online; accessed 15-March-2017]. 2006.

[61] Jan LM Hensen and Roberto Lamberts. *Building Performance Simulation for Design and Operation*. Routledge, 2012.

[62] Tianzhen Hong et al. “An occupant behavior modeling tool for co-simulation”. In: *Energy and Buildings* 117 (2016), pp. 272–281.

[63] Mehlika Inanici. “Evalution of high dynamic range image-based sky models in lighting simulation”. In: *Leukos* 7.2 (2010), pp. 69–84.

[64] MN Inanici. “Evaluation of high dynamic range photography as a luminance data acquisition system”. In: *Lighting Research & Technology* 38.2 (2006), pp. 123–134.

[65] R Isermann, J Schaffnit, and S Sinsel. “Hardware-in-the-loop simulation for the design and testing of engine-control systems”. In: *Control Engineering Practice* 7.5 (1999), pp. 643–653.

[66] *Graphic Technology and Photography - Colour Characterisation of Digital Still Cameras (DSCs) - Part 1: Stimuli, Metrology and Test Procedures.* Standard. Geneva, CH: International Organization for Standardization, 2012.

[67] *Photography – General purpose photographic exposure meters (photoelectric type) – Guide to product specification*. Standard. Geneva, CH: International Organization for Standardization, 1974.

[68] *Information technology – Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF)*. Standard. Geneva, CH: International Organization for Standardization, 2013.

[69] *CIE 15: Technical Report: Colorimetry, 3rd edition*. Standard. Viena, Austria: Internation Commission on Illumination (CIE), 2013.

[70] Axel Jacobs. “High dynamic range imaging and its application in building research”. In: *Advances in building energy research* 1.1 (2007), pp. 177–202.

[71] J Alstan Jakubiec et al. “Accurate measurement of daylit interior scenes using high dynamic range photography”. In: *Proceedings of CIE (International Commission on Illumination) 2016 Lighting Quality and Energy Efficiency Conference, Melbourne, Australia, March 3–5*. 2016.

[72] J Alstan Jakubiec et al. “Improving the accuracy of measurements in daylit interior scenes using high dynamic range photography”. In: *Proceedings of Passive and Low Energy Architecture (PLEA) 2016 Conference, Los Angeles, CA, July 11–13*. 2016.

[73] Henrik Wann Jensen. “Global illumination using photon maps”. In: *Rendering Techniques 1996*. Springer, 1996, pp. 21–30.

[74] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. Vol. 364. Ak Peters Natick, 2001.

[75] C Joram. *Transmission curves of plexiglass (PMMA) and optical grease*. Tech. rep. 2009.

[76] Boris Karamata and Marilyne Andersen. "A review and analysis of parallel goniophotometry". In: *LUXEUROPA 2013*. Vol. 12. EPFL-CONF-190056. 2013.

[77] Boris Karamata and Marilyne Andersen. "Limits and artefacts of reflective imaging goniophotometers for complex solar façade systems". In: *Proceedings EuroSun ISES Europe Solar Conference, Croatia*. 2012.

[78] Boris Karamata and Marilyne Andersen. "Revisiting parallel catadioptric goniophotometers". In: *SPIE Optical Metrology 2013*. International Society for Optics and Photonics. 2013, 87881Q–87881Q.

[79] J. H. Klems and J. L. Warner. "A New Method for Predicting the Solar Heat Gain of Complex Fenestration Systems". In: ASHRAE/DOE/BTECC Thermal Performance of the Exterior Envelopes of Buildings V Conference, Dec. 1992.

[80] JH Klems. "New method for predicting the solar heat gain of complex fenestration systems- I. Overview and derivation of the matrix layer calculation". In: *ASHRAE Transactions* 100.1 (1994), pp. 1065–1072.

[81] JH Klems. "New method for predicting the solar heat gain of complex fenestration systems- II. Detailed description of the matrix layer calculation". In: *ASHRAE Transactions* 100.1 (1994), pp. 1073–1086.

[82] JH Klems, JL Warner, and GO Kelly. "A new method for predicting the solar heat gain of complex fenestration systems". In: *LBL-36995, Lawrence Berkeley National Laboratory, Berkeley, CA* (1995).

[83] Tilmann E Kuhn et al. "Solar control: A general method for modelling of solar gains through complex facades in building simulation programs". In: *Energy and Buildings* 43.1 (2011), pp. 19–27.

[84] Greg Ward Larson and Rob Shakespeare. *Rendering with radiance: the art and science of lighting visualization*. Booksurge Llc, 2004.

[85] T Lautzenheiser, G Weller, and S Stannard. "Photometry for near field applications". In: *Journal of the Illuminating Engineering Society* 13.2 (1984), pp. 262–269.

[86] *Lawrence Livermore National Laboratory*. https://www.llnl.gov/news/americans-used-less-energy-2015/. [Online; accessed 14-January-2017]. 2017.

[87] Norbert Lechner. *Heating, cooling, lighting: Sustainable design methods for architects*. John wiley & sons, 2014.

[88] Edward A Lee. "Modeling concurrent real-time processes using discrete events". In: *Annals of Software Engineering* 7.1-4 (1999), pp. 25–45.

[89] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. Lee & Seshia, 2011.

[90] *Lensfun*. http://lensfun.sourceforge.net/. [Online; accessed 20-July-2016]. 2017.

[91] Clive Staples Lewis. "Miracles; a preliminary study". In: (1947).

[92] *LI-210R Photometric Sensor*. https://www.licor.com/env/products/light/photometric.html/. [Online; accessed 3-February-2017]. 2017.

[93] *LI-COR Terrestrial Radiation Sensors*. English. Li-Cor Biosciences. 38 pp.

[94] *Goniophotometer Types and Photometric Coordinates (Reaffirmed 2012)*. standard. Illuminating Engineering Society, 2012.

[95] Steven Lockley. "Circadian rhythms: Influence of light in humans". In: (2010).

[96] Brian C Madden. "Extended intensity range imaging". In: (1993).

[97] *MakerBot Replicator 2X*. https://store.makerbot.com/printers/replicator2x/. [Online; accessed 1-March-2017]. 2017.

[98] J Mardaljevic. "Validation of a lighting simulation program under real sky conditions". In: *International Journal of Lighting Research and Technology* 27.4 (1995), pp. 181–188.

[99] John Mardaljevic. "Quantification of parallax errors in sky simulator domes for clear sky conditions". In: *Lighting Research and Technology* 34.4 (2002), pp. 313–327.

[100] Francisco Martínez-Verdú, Jaume Pujol, and P Capilla. "Characterization of a digital camera as an absolute tristimulus colorimeter". In: *Journal of Imaging Science and Technology* 47.4 (2003), pp. 279–295.

[101] Andrew McNeil and ES Lee. "A validation of the Radiance three-phase simulation method for modelling annual daylight performance of optically complex fenestration systems". In: *Journal of Building Performance Simulation* 6.1 (2013), pp. 24–37.

[102] Andrew McNeil et al. "A validation of a ray-tracing tool used to generate bi-directional scattering distribution functions for complex fenestration systems". In: *Solar Energy* 98 (2013), pp. 404–414.

[103] AR Mead and KM Mosalam. "A Portable Laboratory-Radiance Cyber-Physical= System For Advanced Daylighting Simulation". In: *Building Simulation 2015*. Hyderabad, India, 2015, pp. 146–152.

[104] AR Mead and KM Mosalam. "Ubiquitous luminance sensing using the Raspberry Pi and Camera Module system". In: *Lighting Research & Technology* (2016). DOI: 1477153516649229.

[105] *Measuring Optically Complex Fenestration Systems*. https://facades.lbl.gov/measuring-optically-complex-fenestration-systems/. [Online; accessed 1-February-2017]. 2017.

[106] *ML-020S Lux Sensor*. https://eko-eu.com/products/solar-radiation-and-photonic-sensors/small-sensors/ml-020s-lux-sensor/. [Online; accessed 3-February-2017]. 2017.

[107] Germán Molina et al. "Evaluation of radiance's genBSDF capability to assess solar bidirectional properties of complex fenestration systems". In: *Journal of Building Performance Simulation* 8.4 (2015), pp. 216–225.

[108] German Ardul Munoz-Hernandez, Dewi Ieuan Jones, et al. *Modelling and controlling hydropower plants*. Springer Science & Business Media, 2012.

[109] Fred E Nicodemus. "Directional reflectance and emissivity of an opaque surface". In: *Applied optics* 4.7 (1965), pp. 767–775.

[110] *NOOBS - New Out Of Box Software*. https://www.raspberrypi.org/documentation/installation/noobs.md/. [Online; accessed 17-February-2017]. 2017.

[111] Thierry Stephane Nouidui. "Functional mock-up unit import in energyplus for co-simulation". In: (2014).

[112] Thierry Stephane Nouidui, Michael Wetter, and Wangda Zuo. "Validation of the window model of the Modelica Buildings library". In: *IBPSA-USA Journal* 5.1 (2012), pp. 529–536.

[113] Carlos E Ochoa, Myriam BC Aries, and Jan LM Hensen. "State of the art in lighting simulation for building science: a literature review". In: *Journal of Building Performance Simulation* 5.4 (2012), pp. 209–233.

[114] James M Palmer and Barbara Geri Grant. *The art of radiometry*. SPIE Press Bellingham, WA, USA, 2010.

[115] *PiCamera*. https://picamera.readthedocs.io/en/release-1.12/index.html/. [Online; accessed 17-February-2017]. 2017.

[116] *Prolemy II*. http://ptolemy.eecs.berkeley.edu/ptolemyII/. [Online; accessed 28-February-2017]. 2017.

[117] Claudius Ptolemaeus. *System design, modeling, and simulation: using Ptolemy II*. Ptolemy. org Berkeley, 2014.

[118] *Python*. https://www.python.org/about/. [Online; accessed 17-February-2017]. 2017.

[119] *RADSITE — radiance-online.org*. http://www.radiance-online.org/. [Online; accessed 11-January-2017]. 2017.

[120] *Raspberry Pi 2 Model B*. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. [Online; accessed 28-February-2017]. 2017.

[121] *Raspberry Pi Camera Module*. https://www.raspberrypi.org/documentation/hardware/camera/. [Online; accessed 22-February-2017]. 2017.

[122] Sidney F Ray et al. *The Manual of Photography: Photographic and Digital Imaging*. Elsevier Science & Technology, 2000.

[123] Erik Reinhard et al. *High dynamic range imaging: acquisition, display, and image-based lighting*. Morgan Kaufmann, 2010.

[124] Christoph F Reinhart. "A simulation-based review of the ubiquitous window-head-height to daylit zone depth rule-of-thumb". In: *Building Simulation*. Vol. 106. 3. 2005.

[125] Christoph F Reinhart and Marilyne Andersen. "Development and validation of a Radiance model for a translucent panel". In: *Energy and Buildings* 38.7 (2006), pp. 890–904.

[126] Christoph F Reinhart, J Alstan Jakubiec, and Diego Ibarra. "Definition of a reference office for standardized evaluations of dynamic façade and lighting technologies". In: *Proceedings of BS2013: 13th Conference of International Building Performance Simulation Association, Chambéry, France, August 26*. Vol. 28. 2013, pp. 3645–3652.

[127] Christoph Reinhart and Annegret Fitz. "Findings from a survey on the current use of daylight simulations in building design". In: *Energy and Buildings* 38.7 (2006), pp. 824–835.

[128] Christoph Reinhart and Ria Stein. *Daylighting Handbook I: Fundamentals, Designing with the Sun*. 2014.

[129] International Agency for Research on Cancer et al. "IARC Monographs Programme finds cancer hazards associated with shiftwork, painting and firefighting". In: *World Health Organization, Lyon* (2007).

[130] *Rhino - design, model, present, analyze, realize...* https://www.rhino3d.com/. [Online; accessed 3-February-2017]. 2017.

[131] *Richmond Field Station Laboratories (RFS)*. http://www.ce.berkeley.edu/programs/semm/facilities/. [Online; accessed 6-January-2017]. 2017.

[132] *RISA - Products*. https://risa.com/products.html/. [Online; accessed 14-February-2017]. 2017.

[133] JL Scartezzini et al. "Bidirectional photogoniometer for advanced glazing materials based on digital imaging techniques". In: *International Journal of Lighting Research and Technology* 29.4 (1997), pp. 197–205.

[134] János Schanda. *Colorimetry: understanding the CIE system*. John Wiley & Sons, 2007.

[135] Andreas Heinrich Schellenberg. *Advanced implementation of hybrid simulation*. University of California, Berkeley, 2008.

[136] P Benson Shing, Masayoshi Nakashima, and Oreste S Bursi. "Application of pseudodynamic test method to structural research". In: *Earthquake Spectra* 12.1 (1996), pp. 29–56.

[137] *Sigma - 8mm F3.5 EX DG Circular Fisheye*. https://www.sigmaphoto.com/8mm-f35-ex-dg-circular-fisheye/. [Online; accessed 17-February-2017]. 2017.

[138] *Single Patch Artificial Sky*. http://www.betanit.com/video/single-patch-artificial-sky/. [Online; accessed 3-February-2017]. 2017.

[139] Mettupalayam V Sivaselvan. "A unified view of hybrid seismic simulation algorithms". In: *8th US National Conference on Earthquake Engineering, San Francisco, California*. 2006.

[140] Steve Stannard and John Brass. “Application distance photometry”. In: *Journal of the Illuminating Engineering Society* 19.1 (1990), pp. 39–46.

[141] Eleanor C Stokes, Nicholas Gayeski, and Marilyne Andersen. “Estimating spectral information of complex fenestration systems in a video-goniophotometer”. In: *Lighting Research and Technology* 40.4 (2008), pp. 269–285.

[142] Anothai Tanachareonkit. “Comparing Physical and Virtual Methods for Daylight Performance Modelling Including Complex Fenestration Systems”. PhD thesis. École Polytechnique Fédérale de Lausanne (EPFL), 2008.

[143] *Tessellate®*. https://www.azahner.com/labs/tessellate/. [Online; accessed 10-April-2017]. 2017.

[144] Anothai Thanachareonkit and Jean-Louis Scartezzini. “Modelling complex fenestration systems using physical and virtual models”. In: *Solar Energy* 84.4 (2010), pp. 563–586.

[145] Anothai Thanachareonkit, J-L Scartezzini, and Marilyne Andersen. “Comparing daylighting performance assessment of buildings in scale models and test modules”. In: *Solar Energy* 79.2 (2005), pp. 168–182.

[146] *The Artificial Sky*. https://www.bartenbach.com/en/research-development/the-artificial-sky.html/. [Online; accessed 3-February-2017]. 2017.

[147] *The Marine Hydrodynamic Laboratory (MHL)*. http://mhl.engin.umich.edu/. [Online; accessed 6-January-2017]. 2017.

[148] *Transonic Cryogenic Tunnel (CTC)*. https://crgis.ndc.nasa.gov/historic/Transonic_Cryogenic_Tunnel/. [Online; accessed 6-January-2017]. 2017.

[149] PR Tregenza and IM Waters. “Daylight coefficients”. In: *Lighting Research and Technology* 15.2 (1983), pp. 65–71.

[150] *Twitchell Leading Brands, Responsive Solutions*. http://www.twitchellcorp.com/. [Online; accessed 3-April-2017]. 2017.

[151] *U.S. Solar Radiation Resource Maps*. http://rredc.nrel.gov/solar/old_data/nsrdb/1961-1990/redbook/atlas/Table.html/. [Online; accessed 11-January-2017]. 2017.

[152] *USB Powered 5-Port 10/100Base-T Ethernet Switch*. http://www.dual-comm.com/5_Port_LAN_Switch.htm/. [Online; accessed 11-March-2017]. 2017.

[153] Michael J Vrhel and H Joel Trussell. “Color device calibration: A mathematical formulation”. In: *IEEE Transactions on Image Processing* 8.12 (1999), pp. 1796–1806.

[154] Greg Ward. *Graphics Gems II*. 1991.

[155] Greg Ward and Cindy Larson. “Radiance User’s Manual”. In: *Adeline 2.0 Advanced Daylighting and Electric Lighting Integrated New Environment. International Energy Agency* (1996).

[156] Gregory J Ward. “Measuring and modeling anisotropic reflection”. In: *ACM SIGGRAPH Computer Graphics* 26.2 (1992), pp. 265–272.

[157] Gregory J Ward. “The RADIANCE lighting simulation and rendering system”. In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM. 1994, pp. 459–472.

[158] Greg Welch and Gary Bishop. “An introduction to the Kalman filter”. In: (1995).

[159] Michael Wetter, Marco Bonvini, and Thierry S Nouidui. “Equation-based languages–A new paradigm for building energy modeling, simulation and optimization”. In: *Energy and Buildings* 117 (2016), pp. 290–300.

[160] Andrew J Willmott and Paul S Heckbert. *An empirical comparison of radiosity algorithms*. Tech. rep. DTIC Document, 1997.

[161] *Workshops*. https://www.radiance-online.org/community/workshops/. [Online; accessed 21-January-2017]. 2017.

[162] William David Wright. “A re-determination of the mixture curves of the spectrum”. In: *Transactions of the Optical Society* 31.4 (1930), p. 201.

[163] William David Wright. “A re-determination of the trichromatic coefficients of the spectral colours”. In: *Transactions of the Optical Society* 30.4 (1929), p. 141.

Appendix A

Glossary

Included here are definitions used throughout this thesis in a consistent manner.

actuator a device which takes input from a digital state value, either directly or via a digital-to-analog converter, and based on that value causes some excitation in the physical domain. Examples include: analog and digital servor motors, hydraulic-cylinders, and lamps.

actuator slip jumps in displacement which are not planned for which a hydraulic actuator can undergo. Causes could be hydraulic pump and distribution system issues, or servo value limitations.

aperture priority mode a method in photography of adjusting the exposure of an image by leaving the aperture static and adjusting only the shutter speed. Denoted "Av" mode on a Canon camera, such as the Canon 6D used in the CUBE 2.0 system.

artificial light light whose origin is anything other than the sun, typically this is luminaires in buildings (e.g. fluorescent, incandescent, metal halides)

Bayer Pattern the complete covering of base pixels on a digital camera sensing chip with odd rows being green, blue, green, blue, etc. and even rows being red, green, red, green, etc.

Bayer Pattern Array a group of four pixels, arraigned two by two, with the top two pixels being green and blue and the bottom two pixels being red and green. Vertical and horizontal concatenations of BPAs on the base pixels of camera sensing chip make up the full Bayer Pattern.

birefringement the phenomena in which a material has a different *index of refraction* depending on the *polarization* of the light passing through it

bidirectional reflection distribution function (BRDF) a function whose domain is incoming light direction and magnitude and whose range is the direction and magnitude of the non-specular reflection of that incoming light.

Bi-directional Transmission Distribution Function (BTDF) a function whose domain is incoming light direction and magnitude and whose range is the direction and magnitude of the non-specular transmission of that incoming light through the material being described.

Bi-directional Scatter Distribution Function the joint set of both a BRDF and BTDF for a given material, thus describing both the reflection and transmission of light striking the surface of a material being described.

Black Body Radiator Theory a consequence of quantum theory which describes the relative wavelength dependent electromagnetic radiation for an object based on its surface temperature, codified by Plank's Law

building a subspace of the three dimensional world, in which the environment is controlled to be more conducive for some desired task, what architects call the 'building program'

central radiance equation $L_r(\theta_r, \phi_r) = L_e + \iint L_i(\theta_i, \phi_i) f_r(\theta_i, \phi_i; \theta_r, \phi_r) cos(\theta_i) sin(\theta_i) d\theta_i d\phi_i$. θ is the altitude angle of some ray measured from the zenith toward the horizon and ϕ is the azimuth angle measured counter-clockwise from the x-axis. The i, r, and e stand for "incident," "reflected," and "emitted". L is the radiance in some specific direction and f_r is the reflective-transmission function relating incident radiation to reflected and transmitted radiation. The $cos(\cdot)$ and $sin(\cdot)$ terms are geometric factors which account for angle of incident and the changing of coordinate systems from steradians to θ and ϕ respectively. This equation is the primary mathematical-model used in the industry standard computational program Radiance. Radiance uses a novel deterministic-stochastic, recursive algorithm for its solution. See chapters 10, 11, 12, and 13 of Rendering with Radiance for an overview description of the algorithm [84].

CFS test aperture the 1 inch diameter circular whole on the front of the CUBE 2.0 system over which CFS specimens undertest are placed.

complex fenestration system (CFS) a fenestration system which exhibits non-specular reflection/transmission as opposed to specular reflection/transmission associated with classical fenestration systems. CFSs are typically characterized by their *Bi-directional Reflection Distribution Function* (BRDF) and *Bi-directional Transmission Distribution Function* (BTDF), jointly refereed to as the *Bi-directional Scatter Distribution Function* (BSDF)

concurrent execution execution of tasks which happen logically at the same time, however, given a thread scheduler's control, they may in fact execute at different real times

CUBE 2.0 the proposed hardware-in-the-loop model, the central topic of this thesis, which is used to represent a building daylighting system under test. It is comprised of physical-models (i.e. CFS) and mathematical-models (i.e. Radiance) interacted during simulations to represent the total building daylighing system under test

cyber-domain the "world" of computation taking place inside digital computers, often abstracted by synchronous digital logic models. It should be noted, the physics occurring in a processor are also of the physical-domain, however, they are representations of digital abstractions from the human perspective, and hence are distinctly different than continous systems.

cyber-physical system a system comprising both *physical-domain* and *cyber-domain* components, mandating concurrent analysis of computers, software, networks, and physical processes [89].

daylight sunlight with wavelength of 380-780 nm to which the human eye is sensitive [134], also called *visible sunlight*

daylighting the act of using daylight to provide illumination within a building

daylight factor (DF) one of the simplest daylighting metrics, expressing the ratio of indoor illuminance (I_{in}) with in a building to the unobstructed outdoor illuminance (I_{out}) on the roof of that same building: $DF = \frac{I_{in}}{I_{out}}$ [87]. Ideal DF values vary depending on space type, yet are almost always less than 5%.

demosaicing algorithm an algorithm, often proprietary, which takes the raw pixel values from a CCD or CMOS chip (often arranged in a Bayer Pattern) and interpolates them across the chip, resulting in each pixel having three channels when in reality each pixel can only measure one channel

diffuse reflection reflection at a *rough* surface such that the reflected ray leaves the surface at several angles; compliments *specular reflection*

dispersion the phenomena of wavelength dependent *index of refraction* for a given material

environment the remaining set when considering the Universe minus some corresponding system under study.

erradict random beams of light within a building only visible for certain periods of the year.

experiment the process of extracting information from a system by exercising its inputs and observing its outputs;

exposure compensation relatively the same scale as EV, however, EC=0 refers to an optimal exposure value in some sense, while EC=-1 is an under exposed image and EC=+1 is an over exposed image. It should be noted EC is a sliding scale where as EV is a fixed scale.

exposure value a log base two number which represents the shutter speed and aperture size uniquely, such that any shutter speed and aperture size combination with the same EV will result in the same exposure. Due to log base two, an increase or decrease of one means halving or doubling the exposure of the image respectively.

Far Field Photometry photometry which treats sources as points without physical dimension, greatly reducing the mathematical complexity required for their analysis

fenestration system the openings in a building's walls, traditionally including doors and glazing, herein referring to exclusively glazing

geometric optics the branch of optics for which *ray models* are used as a system representation of light. Contrasts with *physical optics*.

glazing a word used in the architecture community for windows

hardware-in-the-loop model (HWiL model) a representation of a system with certain sub-systems represented using mathematical-models and other sub-systems represented using physical-models, where all sub-system model components are interacted using actuators and transducers to allow for the representation of the larger super-system.

hardware-in-the-loop simulation (HWiL simulation) the act of running an experiment on a hardware-in-the-loop model of a system.

high dynamic range image (HDRI) an image that stores a depiction of a scene in a range of luminance intensities commensurate with the actual physical scene [123]

input state(s) within the environment which influence system behavior

interface the inputs from some environment and the outputs of some corresponding system which together define the interaction of the system-environment pair; note, for a given system there may exist several interfaces depending on the desired analysis. For example, an automobile has a (tire,asphalt) interface, but also (body panel,air) interface which can be considered independently.

Klems Basis a set of solid angles which discretize the incident and exiting hemispheres of a complex fenestration system (CFS). Originally proposed by Klems [80], for the mathematical modeling of daylight and heat transmission through building envelopes, this discretization scheme is the standard for CFS analysis.

lamp the physical element within a luminaire which emits light

light any form of electromagnetic radiation (EMR) with wavelength between 380 and 780 nanometers, hence can be sensed by the human visual system

linearly polarized light polarized light with the electrical field oriented parallel to a Y axis

low dynamic range image (LDRI) an image that stores a depiction of a scene suitable for display with current display technology [123]

luminaire a complete unit used to provide light within buildings, usually comprising a lamp, reflectors, mounting bracket, etc.

mathematical-model a mathematical description of the relationship between variables making up a system

measurement cone the "light proof" enclosure of the CUBE 2.0 in which the RPiCM captures a high dynamic range image of the ends of the optical fibers, producing a measurement of the *exiting luminance distribution*, L_v

model anything representing a system of interest on which an "experiment" can be applied in order to answer questions about that system

model time the time used in a mathematical model to represent "real time". It can run slower, the same as, or faster than real time all depending on modeling/simulation specific goals. For example, if studying the formation of the Universe, faster then real time is desired, yet if studying nuclear chemistry, slower than real time will almost certainly be the only form possible due to the extreme speeds with which these physical processes occur.

Near Field Photometry photometry which treats light sources as having physical dimension (as opposed to point sources which are studied using Far Field Photometry), meaning the mathematics used for their analysis are much more complex

output state(s) within the system which influence environment behavior

overshoot the phenomena in controls that a state value is reached, but due to "momentum" of the system keeping increasing or decreasing past the desired value, only to reach the desired value after more time. Proportional, Integral, Derivative (PID) controllers can be used to limit overshoot.

parallel execution execution of tasks which happen both logically at the same time and in real time at the same instant

parasitic light light which is not meant to be in a certain domain, yet finds itself there due to poor barrier constructions or optical "flaws" within instruments. For example, in daylighting scale models if walls aren't joined together properly, light can enter where it wouldn't in a real building.

photon map method a light modeling/simulation method using multiple passes in which packets of energy (photos) are distributed to the surfaces within the scene to predict the luminous environment

physical domain the space where all observable matter and phenomena occur (i.e. the *Universe*), often modeled using continuous time based mathematics

physical-model a physical construction, governed by the laws of similitude, designed to represent some system

physical optics the branch of optics for which *wave models* are used as a system representation of light. Contrasts with *geometric optics*.

polarized light light where all the constituent waves have their electrical field component oriented in the same direction

prototype a full scale physical model of a system

projection formula for a specific lens type, a function that relates an incident ray angle, with respect to the lens optical axis, to a distance from the same optical axis on the image plane [20]

RAW pixel value the value of a pixel right out of the analog to digital converter, previous to any post processing (e.g. gamma encoding, demosaicing)

Radiance the industry standard modeling/simulation program for lighting systems in buildings, it is an open source framework originally developed at Lawrence Berkeley National Laboratory (LBNL) by Greg Ward in the late 1980's, which solves the central radiance equation using a parameterized, mixed deterministic-stochastic solution scheme. See Rendering with Radiance [157] or home page RADSITE [119]

radiosity method a light modeling/simulation method, originating in the heat transfer community, which uses a discretization scheme and subsequent calculation of the radiance of each discretized element in the scene to predict the luminous environment

real time the rate at which time proceeds in the physical domain, often called "wall clock time" due to the fact this is the time a clock on the wall measures.

relaxation the phenomena where a physical model or actuator moves or changes imparted force, even though it is trying to maintain position or load.

sensor a device which converts some physical excitation into a digital state value through the use of an analog-to-digital converter. Examples include: thermocouples, strain gages, and accelerometers.

simplified techniques a broad class of modeling/simulation techniques where experience and judgement rather than analytical methods are used to gain information about a system. Also referred to as "rules-of-thumb," the systems in which these techniques are applied for investigation are often called "unengineered systems"

similitude the concept of two systems having the same value for some dimensionless number; an important concept in reduced scale physical model modeling and simulation for engineering applications.

simulation an experiment performed on a model; originating from the Latin word *simular*, meaning to pretend

specimen relaxation the reduction in resistance to deformation of a physical model when held at some constant displacement, often caused by micro-cracking within the physical model or other relaxation phenomena.

specular reflection reflection at a *smooth* surface such that the reflected ray leaves the surface a definite angle as defined by the Law of Reflection; compliments *diffuse reflection*

specular transmission the transmission of rays through an object such that the rays exiting the object out the opposite side of incidence, are roughly in the same direction they entered, without spreading of the beam of light

state variables from the Universe; For example: dry bulb air temperature, solar irradiance, price of crude oil per barrel

state-estimation framework any of a series of mathematical frameworks combining mathematical representations of a system's state variables (e.g. state-space model) with sensor measurements of a system's state variables (e.g. temperature, acceleration) with the aim of improving the confidence in the estimate of the state variables of concern. Classic examples include the Luenberger Observer and the Kalman Filter.

sub-system a system that is a subcomponent of another systems; usually used with reference to a "super-system" for hierarchal clarity

sunlight electromagnetic radiation (EMR) produced by the sun whose continuous spectral distribution, referred to as the *solar spectrum*, is well approximated as a Black Body at 5550^o Kalvin. It is typically though of as being comprised of three main wavelength defined components: ultraviolet ($10 < \lambda < 380nm$), visible ($380 < \lambda < 780nm$), and near-infrared ($780 < \lambda < 10^6nm$).

super-system a system where its subcomponents are also systems; usually used with reference to a "sub-system" for hierarchal clarity

system an object or collection of objects whose properties we want to study, which are a subset of the Universe. Examples include: automobiles, a building, an airplane, a pacemaker, a human heart.

transmission method a simplistic method for measuring daylight transmission, τ, through a panel as $\tau = \frac{E^{In}_{v,T}}{E^{Out}_{v,T}}$

Universe the set of all observable matter and phenomena, i.e. "the whole show" [91]

unpolarized light light which is comprised of a random selection of linearly polarized light in all possible directions

vignetting the property of a lens which causes the attenuation of light with respect to the central axis of the lens as the entrance angle increases within the created image

Appendix B

Nomenclature

Included here are mathematical nomenclature used throughout this thesis in a consistent manner.

A_s area on a sphere used to calculate a solid angle

$A_{\mu\theta}$ the area associated with a micro-theta band used to calculate the solid angle of the pixels

bp brightest point in a scene being captured by an HDRI

$calPt_n^t$ a calibration point from time step t, consisting of an input luminance, $L_v^{in,n}$, and arbitrary RPiCM output luminance measurement, $\mathcal{OFM}_n^i$

$calPt_n^t|_e$ the $LDRI_{RPi,e}$ exposure from which the $calPt_n^t$ was taken

dp darkest point in a scene being captured by an HDRI

D a member of $\mathbb{R}^{145\times 145}$ this is the "daylight matrix" mapping sky luminance to CFS illuminance

e an exposure within the set $LDRI_{RPi}^j$, with $e \in \{1, \ldots, j\}$

err^t mean element wise percentage error of i_m and $i_{m,s}$

E_b fourth byte in a Radiance XYZE pixel ecoding scheme

E_v $E_v \in \mathbb{R}^{145\times 1}$, directionally varying, vertical illuminance distribution, based on the Klems Basis, striking the outside of the complex fenestration system (CFS) undertest coming from the sky and surrounding objects

E_v^t an E_v measurement corresponding to round t of a simulation of the CUBE 2.0 system

$E_{v,T}^{In}$ vertical illuminance total on the inside of a CUBE 1.0 testing bay

$E_{v,T}^{Out}$ vertical illuminance total on the outside of a CUBE 1.0 testing bay

E_T^a, E_T^b, E_T^c real numbers representing total vertical illuminance falling on the *CFS test aperture*, *fisheye lens*, and *LI-210* respectively

E_K^a, E_K^b, E_K^c members of $\mathbb{R}^{145\times 1}$ representing Klems Basis discretized vertical illuminance on the *CFS test aperture*, *fisheye lens*, and *LI-210* respectively. Note for $i = \{a, b, c\}$, $E_T^i = \sum_{c=1}^{145} E_K^i(c)$.

$E_K^{a,r}, E_K^{b,r}, E_K^{c,r}$ members of $\mathbb{R}^{145\times 1}$ representing Klems Basis discretized vertical illuminance for sky condition r on the *CFS test aperture*, *fisheye lens*, and *LI-210* respectively

f focal length of a given lens

f_{pixel} focal length of a given lens in the units of pixels as measured and calculated using a *projection formula*

ϕ azimuth angle as defined in the spherical coordinate system

ϕ_e the effective ϕ associated with some pixel (i, j) within an image plane of a fisheye lens

ϕ_p an intermediate ϕ used for quadrants II, III, and IV of the image plane

ϕ_{min}, ϕ_{max} the respective minimum and maximum ϕ values for some Klems Basis Patch

$HDRI_{input}$ a high dynamic range image which is an input measurement for the CUBE 2.0 system.

i, j coordinates of some pixel in an image; note: bottom left is the origin

i_{of}^n, j_{of}^n pixel location of the center of optical fiber n within an RPiCM generated $LDRI_{RPi}$

i_c, j_c pixel coordinates of the center location of a circular fisheye lens image

i_m^t the luminous metrics calculated via the three-phase method as $i_{measure}^t = V_{csf} \cdot L_v^t$

$i_{m,s}^t$ the luminous metrics calculated via the three-phase method as $i_{measure}^t = V_{csf} \cdot L_{v,s}^t$

$i_m^{t,g}$ the luminous metric calculated at some point g for time step t from the measured data

$i_{m,s}^{t,g}$ the luminous metric calculated at some point g for time step t from the simulated data

ill_{HDRI} a vertical illuminance measurement calculated from an $HDRI_{input}$

ill_{LC} a vertical illuminance measurement from a LI-210 associated with some HDRI

k_n the set of pixels associated with Klems Basis Patch n

K_I HDRI SI unit correction factor based on vertical-illuminance calibration method

K_L HDRI SI unit correction factor based on spot-luminance calibration method

K_n the θ and ϕ dimensions of the Klems Patch n

$\mathcal{K}_r$ the set of integers with corresponding Klems Basis Patches which are excited by the sky vault (i.e. can be used for calibration) for a given calibration data collection deployment session, for which eight are used to calibrate the CUBE 2.0 system

$L(\theta, \phi)$ luminance distribution function independant of location, dependent on direction

$L(x, y, \theta, \phi)$ a luminance distribution function dependent on location, here x and y are cartesian coordinates, not pixel sizes

$L_{fish}(\theta, \phi), L_{CFS}(\theta, \phi), L_{LiCor}(\theta, \phi)$ the luminance distributions "seen" by the Canon 6D fisheye lens, CFS test aperture, and LI-210 respectively

$\mathcal{L}_{i,j}$ a spot-luminance measure, $[\frac{cd}{m^2}]$, associated to an HDRI through a set of pixels $l_{i,j}$ centered about the pixel (i, j)

L_v a member of $\mathbb{R}^{145\times1}$, directionally varying, exiting luminance distribution, based on the Klems Basis, leaving from the inside of the complex fenestration system (CFS) under-test into the space

L_v^t an L_v measurement corresponding to round t of a simulation of the CUBE 2.0 system

$L_{v,s}^t$ an L_v^t which is attained for a CFS with known BTDF by $L_{v,s}^t = T \cdot E_v^t$

L_v^{in} directionally varying input luminance distribution to the CFS test aperture as measured by the Canon 6D for calibration purposes

$l_{i,j}$ a set of pixels in an HDRI image centered around the pixel (i, j) associated with a spot-luminance measurement $\mathcal{L}_{i,j}$

L_{pts} the set of exiting luminance values where 'x' is the measured and 'y' is the simulated value for a particular Klems Basis Patch

$LDRI$ a monochromic low dynamic range image complete with pixel array size, bit depth, aperture, and shutter speed.

$LDRI_{RPi}$ a LDRI which measures color instead of a monochromic image, meaning $\mathcal{P}$ is replaced with $\mathcal{P}_c = (\mathcal{P}, \mathcal{P}, \mathcal{P})$, where each $\mathcal{P}$ is itself a monochromic channel

$LDRI_{RPi,e}$ the e^{th} $LDRI_{RPi}$ in $LDRI_{RPi}^j$

$LDRI^j$ a set of LDRI, $j = 1, 3, 5, \ldots$ in size, with varying EC chosen to span the luminance range of the scene.

$LDRI^{j}_{RPi}$ a set of $LDRI_{RPi}$ taken by the RPiCM each with varying shutter speeds, j in size

$\mathbb{M}$ a member of $\mathbb{R}^{3\times 3}$ which transforms RGB_{RPi} color space to CIE-XYZ color space

$\mathbb{N}$ the set of natural numbers, here this includes 0

n an optical fiber or Klems Basis solid angle; an integer between 1 and 145

n_a, n_b the index of refraction of materials a and b

$\mathcal{OF}^{e}_{n}$ the 144 BPA set associated with optical fiber n and from $LDRI_{RPi,e}$

$\mathcal{OFM}^{e}_{n}$ the final CIE-Y measurement for optical fiber n from $LDRI_{RPi,e}$

$\mathcal{OFY}^{e}_{n}$ the 144 CIE-Y values set of optical fiber n from $LDRI_{RPi,e}$

θ elevation angle as defined in the spherical coordinate system; for zenith $\theta = 0$.

θ_i incident angle with respect to the optical axis of a lens

θ_e an effective incident angle associated with some pixel (i, j) within an image plane of a fisheye lens

$\theta_{up}, \theta_{down}$ the respective upper and lower limit of a micro-theta band

$\theta_{min}, \theta_{max}$ the respective minimum and maximum θ values for some Klems Basis Patch

$p_{\mu\theta}$ the number of pixels in a micro-theta band, i.e. the cardinality of $pix_{\mu\theta}$

$pix_{\mu\theta}$ the set of pixels associated with a micro-theta band

$\mathcal{P}$ a monochromic space of pixel values for a LDRI

$\mathcal{P}_c$ a three channeled space for pixel values within an LDRI, where each channel is in fact its own monochromic pixel value

$\mathcal{P}_h$ real numbers which represent luminance in an HDRI

$\mathcal{P}_Y$ a single channeled space where the value is the CIE-Y value (i.e. luminance)

$\mathcal{P}_{XYZ}$ a three channeled space for pixel values within an LDRI where the channels are specifically the CIE-XYZ space

Ω a solid angle, measured in steradians $[sr]$

$\Omega_{\mu\theta}$ the solid angle associated with a given micro-theta band

$\mathbb{R}$ the set of real numbers

r the distance from the optical axis within an image corresponding to some incident angle θ_i, related by a projection formula for a specific lens type

r_e an effective image plane distance of some pixel (i, j) with respect to the change in coordinates for moving the origin from bottom left to the center pixel (i_c, j_c)

r_s the radius of a sphere which is used for calculating solid angle

r_{90} the effective image plane distance of an object placed at the very edge of the input hemisphere, it is used define what pixels are in and out of the input measurement on the fisheye image

R_c a positive integer corresponding to the number of rounds in a calibration data deployment

s a member of $\mathbb{R}^{145\times1}$ which represents luminance values of the sky vault in the form of a Tregenza Basis

RGB_{rpi} the RPiCM hardware specific color space

RGB_λ the 3×33 matrix with the measured RPiCM response data for the spectral characterization

τ transmission of a complex fenestration system panel using the “transmission method”

t_{end} a positive integer representing the number of time steps in a simulation

$\mathcal{T}_r$ the set of integers from one to the number of rounds in a calibration data collection deployment session, for which eight are used to calibrate the CUBE 2.0 system

T a member of $\mathbb{R}^{145\times145}$, this is the bidirectional transmission distribution function

V a member of $\mathbb{R}^{p\times145}$, the “view matrix” mapping CFS exiting luminance to whatever Radiance metrics are calculated as part of the three-phase method

V_{cfs} a view matrix, V, for a specific daylighting system with a CFS

x, y dimensions of an image in pixels

x_e, y_e effective pixel coordinates of some pixel within a circular fisheye lens image

XYZ_λ the 3×33 matrix with the CIE-XYZ color space defined response corresponding to the RGB_λ response data

ω a solid angle measured in steradians $[sr]$

$\omega(i, j)$ the solid angle associated with some pixel (i, j)

Y_b second byte in a Radiance XYZE pixel ecoding scheme

Appendix C

Complex Fenestration System (CFS) System Identification: A New Type of Parallel Goniophotometer

The CUBE 2.0 system as presented in this thesis measures the input illuminance falling upon the outside of the complex fenestration system (CFS) under test, E_v, using the Canon 6D equiped with a fisheye lens. In addition, it measures the exiting luminance distribution leaving the back side of the panel, L_v, using the optical fiber array and RPiCM.

With respect to the three-phase method [101] and outlined in chapter 3, these terms can also be expressed as,

$$E_v = Ds, \tag{C.1}$$

and

$$L_v = TDs. \tag{C.2}$$

Here, $T \in \mathbb{R}^{145\times145}$ is the bidirectional transfer distribution function (BTDF) matrix, $D \in \mathbb{R}^{145\times145}$ is the dayligihing matrix mapping Tregenza sky patches to input Klems Basis Patches, and $s \in \mathbb{R}^{145\times1}$ is the sky luminance vector descretized using the Tregenza Basis. Further, one of each of these terms will be produced for each time step, resulting in a set of E_v and L_v pairs,

$$\mathcal{IO} = \{(E_v^1, L_v^1), \ldots, (E_v^{t_{end}}, L_v^{t_{end}})\}, \tag{C.3}$$

where $t_{end} \in \mathbb{N}\backslash\{0\}$ is the number of time steps in a partcular simulation.

Examining the set $\mathcal{IO}$, one notices the only difference between the terms in each two-tuple is the BTDF matrix T. As stated in the thesis above, measurement of T is a tedeous and involved process using specialized machines, of which only a handful exist in the world. Hence the notion of system identification may be able to be used to "learn" T using the data generated during a simulation[1]. This would expand the CUBE 2.0 system from a hardware-in-the-loop model capable of simulating daylighting systems, into a novel type of parallel goniophotometer capable of estimating the BTDF of novel CFS which are being tested.

[1]This idea was orginally proposed by Professor Edwared A. Lee.

System ID is a large field in itself and often requires specialized knowledge of algorithms and model types to best fit the situation at hand. A cursory investigation of this phenomena is presented here with the hope future work can be devoted to exapand the findings to more general application.

C.1 Formulating the System Identification Problem

To explore the potential of the CUBE 2.0 being used as a novel parallel goniophotometer based on system ID techniques, a previously measured CFS specimen is used. In this case, the shading material from Twitchell [150], Textilene 80 Black as presented in chapter 6, is used. Textilene 80 Black has been measured on an industry standard goniophotometer and a BTDF, T_{80} is available through the Complex Glazing and Shading Database [35].

While not typically an option, to simplify the intial investigation, the structure of the BTDF T_{80} is examined for structural features. These features are hoped to simply the system ID problem making estimation easier. Immediatly upon examing T_{80} it is clear the diagonal of the 145×145 matrix dominates the transmission, accounting for over 93% of the total coefficient values. That is,

$$trans_{diag} = 93.5\% = 100 \times \left(\frac{\sum_{i=j} T_{i,j}}{\sum_{i,j} T_{i,j}} \right). \tag{C.4}$$

Further, one notices the diagonal values are assumed equal per theta ring, hence there exists only nine different values along the diagonal. Put formally, $T_{1,1}, T_{2,2} = \cdots = T_{9,9}$, $T_{10,10} = \cdots = T_{25,25}, T_{26,26} = \cdots = T_{45,45}, T_{46,46} = \cdots = T_{69,69}, T_{70,70} = \cdots = T_{93,93}, T_{94,94} = \cdots = T_{117,117}, T_{118,118} = \cdots = T_{133,133}, T_{134,134} = \cdots = T_{145,145}$, with the free variables organized in vector form as, $T_{diag} = [T_{1,1} \ \cdots \ T_{145,145}]^T$, where superscript T denotes transpose.

Using this information, the system ID problem can be reasonably formulated using these nine terms as free variables in a least squares sense.

C.2 Initial Data Examination

The data for select Klems Basis Patches is plotted for both the input and the output to determine if there exists a hope of using system ID. Initial intuition based on the comparison of L_v and $L_{v,s}$ from the HWiL simulation suggestions there is, yet further data is presented in another form below.

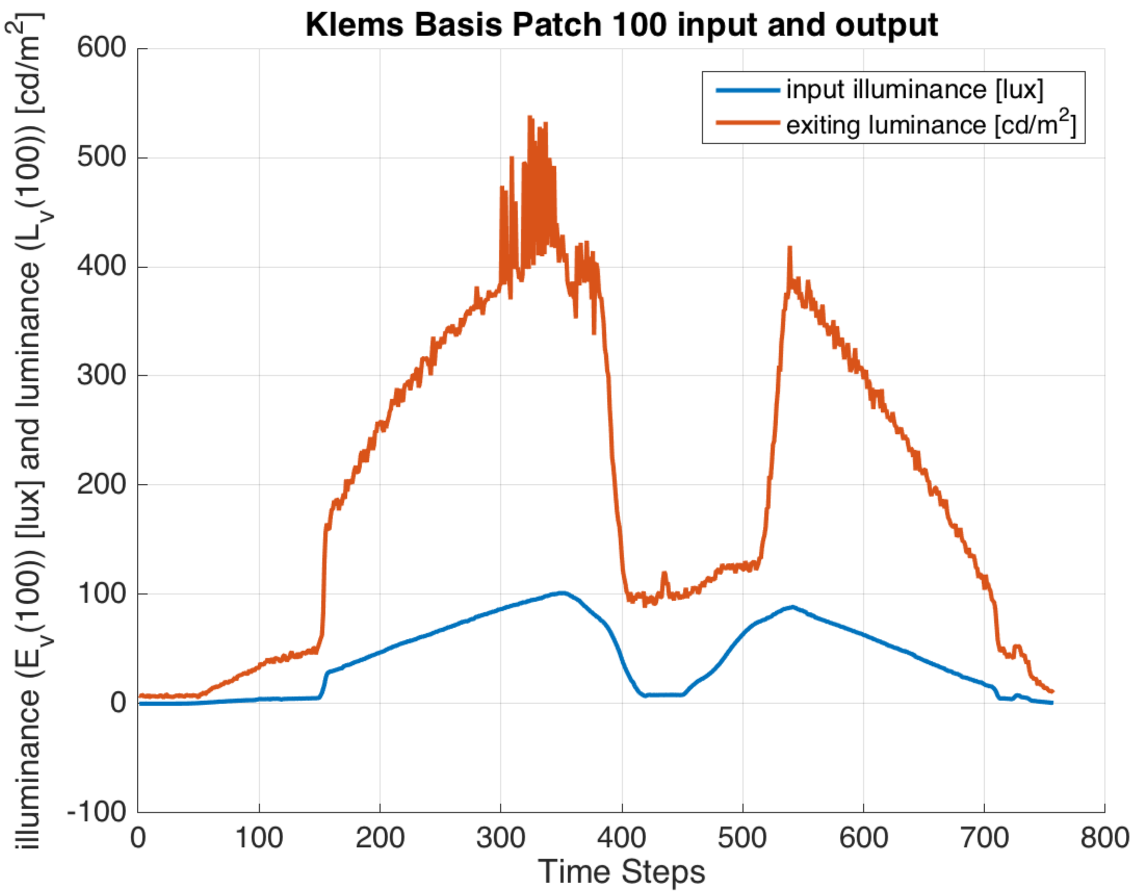


Figure C.1: Input illuminance and output luminance for Klems Basis Patch 100 for Textilene 80 Black.

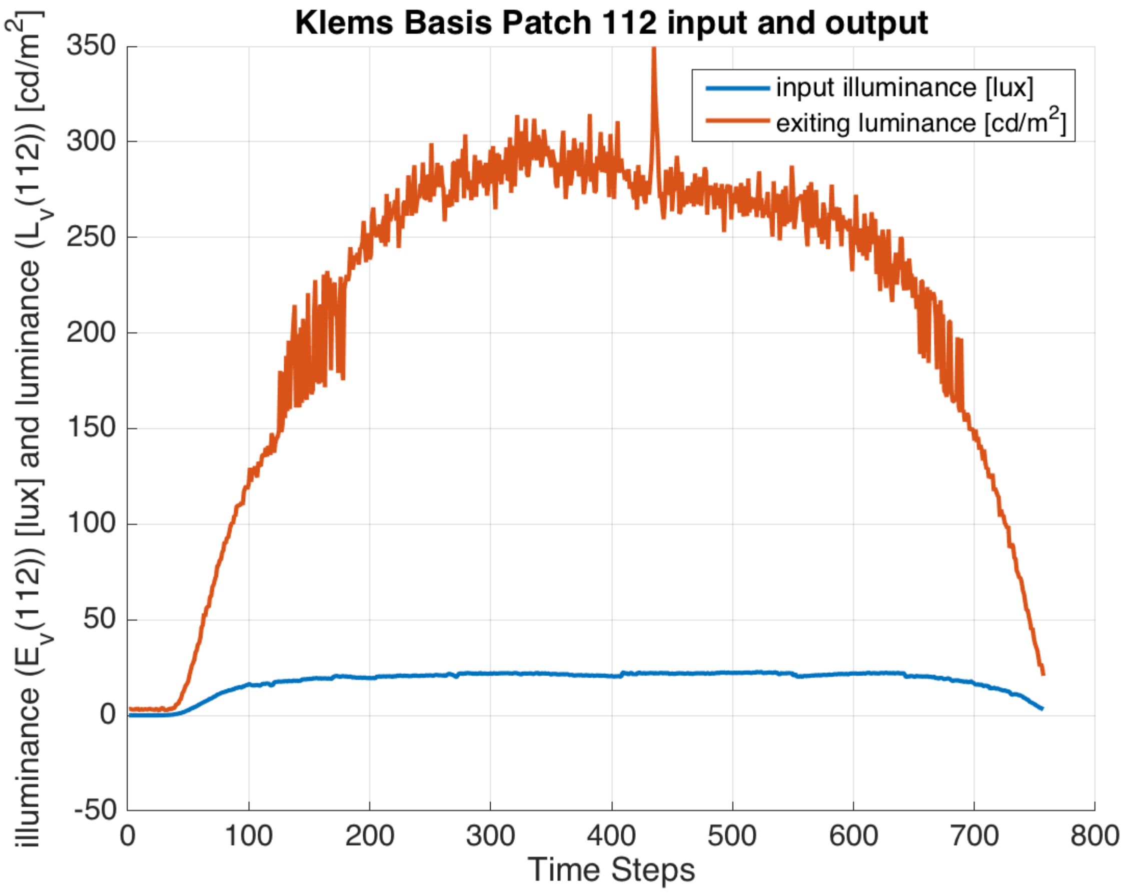


Figure C.2: Input illuminance and output luminance for Klems Basis Patch 112 for Textilene 80 Black.

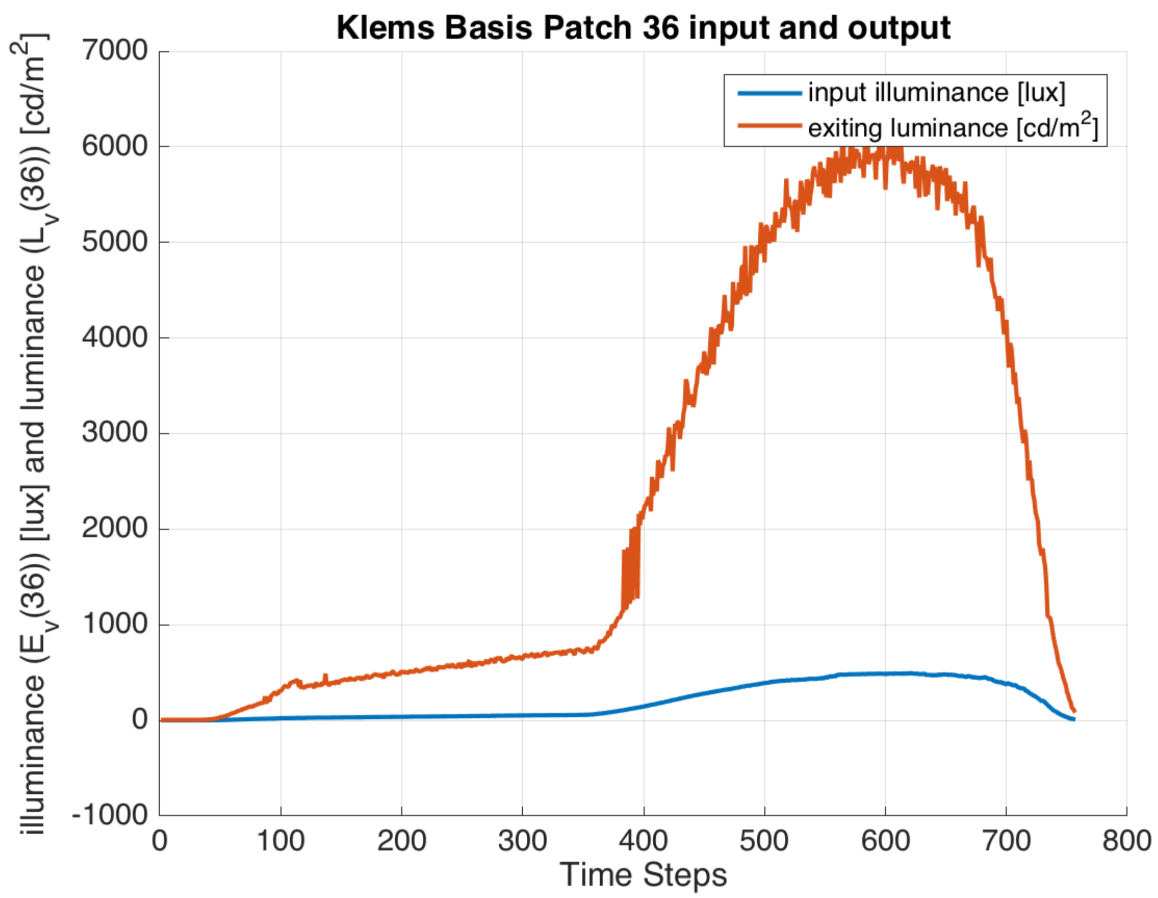


Figure C.3: Input illuminance and output luminance for Klems Basis Patch 36 for Textilene 80 Black.

Examing Figures C.1, C.2, and C.3, it is clear there exists at least a qualitative agreement between the input, E_v, and output, L_v, measurements. Hence, the formulation of the optimization problem is presented next. Note, the three previous Klems Basis Patches were selected somewhat at random, with most of the patches exhibiting very similar behavior.

C.3 System Identification Algorithm: Quadradic Program

Considering a generic time step, t, the objective function can be formulated as,

$$e_t = (L_v^t(1) - T_1 * E_v^t(1))^2 + \cdots + (L_v^t(145) - T_{145} * E_v^t(145))^2, \qquad (C.5)$$

where for the presented data in chapter 6 involving 757 times steps the total objective function is,

$$objFunc \ = \ \sum_{t=1}^{757} e_t \ = \ \sum_{t=1}^{757}\left(\sum_{k=1}^{145}\left(L_v^t(k) - T_k E_v^t(k)\right)^2\right). \tag{C.6}$$

Expanding the inside terms of the double summation one gets,

$$objFunc \ = \ \sum_{t=1}^{757}\left(\sum_{k=1}^{145}\left(T_k^2 E_v^t(k)^2 - 2L_v^t(k)E_v^t(k) + L_v^t(k)^2\right)\right), \tag{C.7}$$

and by removing the constant term one is left with

$$objFunc \ = \ \sum_{t=1}^{757}\left(\sum_{k=1}^{145}\left(T_k^2 E_v^t(k)^2 - 2L_v^t(k)E_v^t(k)\right)\right). \tag{C.8}$$

This is a quadratic program with respect to the T_{diag} terms, which can be expressed in the canonical form:

$$\underset{T_{diag}}{min} \ \frac{1}{2}T_{diag}^T H T_{diag} + f^T T_{diag} \ \begin{cases} A \cdot T_{diag} \leq b, \\ A_{eq} \cdot T_{diag} = b_{eq}. \end{cases} \tag{C.9}$$

As stated before, several of the T_{diag} terms are equal, a constraint realized in $A \cdot x \leq b$ via two inequalities per equality. Further, each T_{diag} must be nonnegative, a constraint also realized in $A \cdot x \leq b$.

Formulating the matrix H and vector f requires gathering like terms in equation C.8, each of which can be expresssed respectively as,

$$H \ = \ \begin{bmatrix} E_v(1,:)^T E_v(1,:) & 0 & \cdots & 0 \\ 0 & E_v(2,:)^T E_v(2,:) \ 0 & \cdots & 0 \\ \vdots & \cdots & \ddots & \vdots \\ 0 & \cdots & 0 & E_v(145,:)^T E_v(145,:) \end{bmatrix}, \tag{C.10}$$

and

$$f \ = \ \begin{bmatrix} -L_v(1,:)^T E_v(1,:) \\ -L_v(2,:)^T E_v(2,:) \\ \vdots \\ -L_v(145,:)^T E_v(145,:) \end{bmatrix}, \tag{C.11}$$

where $E_v(1,:) = [E_v^1(1) \ \ldots \ E_v^{757}(1)]^T$, with likewise notation used for exiting luminance, $L_v(1,:)$.

The minimization is thus solved using the Matlab routine *quadprog* and the results are presented below.

C.4 Results and Discussion

In examining Figure C.4, the first feature noticed of the estimated coefficients is the rather stirking qualitative agreement with the measured data. Overall the relative spacing of each theta band's coefficient is very similar. In general the system ID estimated coefficients have lesser values than the measured inputs. Exceptions to this are the first, second, and last theta band. Overall this is consistent with the validation results of the exiting luminance, $L_{v,s} = T_{80}E_v$, which was calculated in chapter 6. That is, the simulated value is greater due to the larger value of the BTDF matrix used that is estimated by the input and output data.

While this is just a preliminary study, the results are promising that with further anlaysis the CUBE 2.0 system could indeed be used as a parallel goniophotometer using system identification techniques. This quadratic program was setup with raw data. It is hypothesized with various filtering techniques and advanced system identification algorithms utilized given the physics of CFSs, the estimated BTDF can improve.

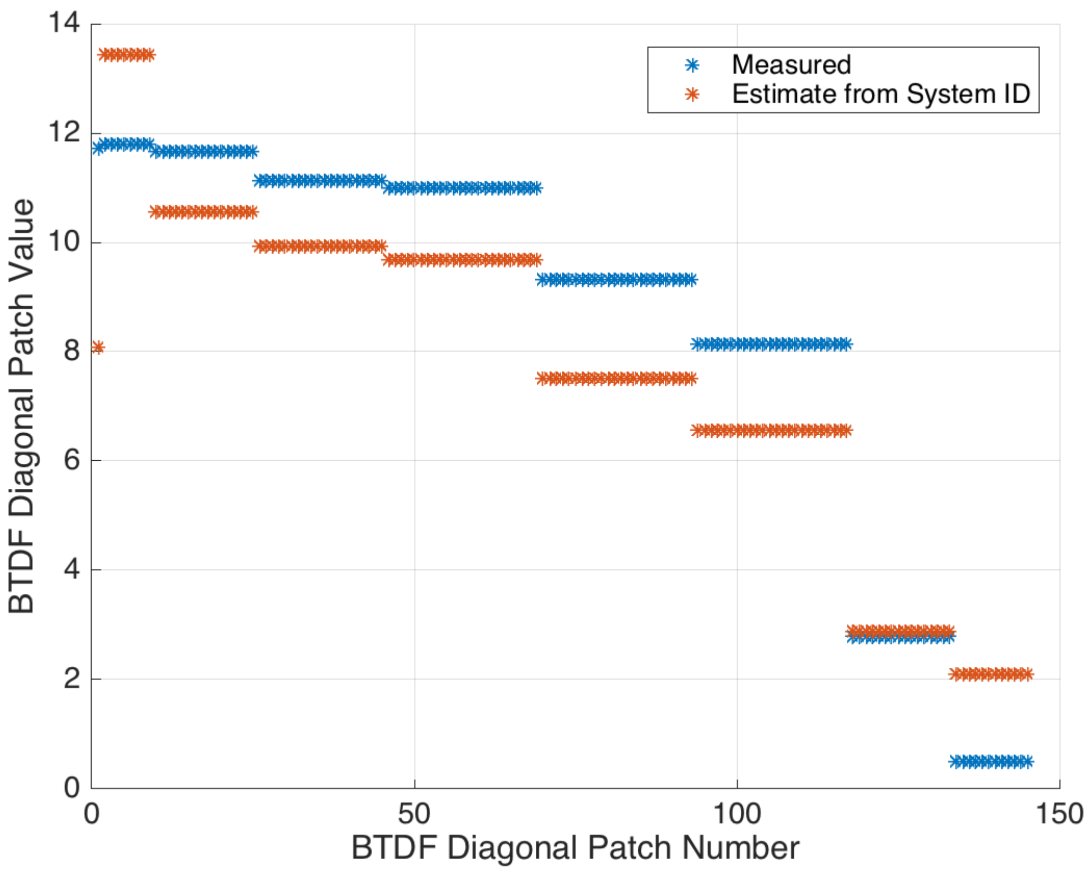


Figure C.4: Measured and System ID Estimated BTDF for Textilene 80 Black.

Appendix D

Code Used in the CUBE 2.0 System

This appendix shows the code which is used to run the CUBE 2.0 system. The author is well aware much of the code is "sub-optimal," however, it is sufficient for the needed task.

In the time since this implementation, many changes to increase speed, as well as increase system operational robustness, have been identified. If implementing another system of this type, it is suggested a second look be given to the structure of some functions as well as some programs in total. Specifically, the C programs could use improvement in general.

D.1 main()

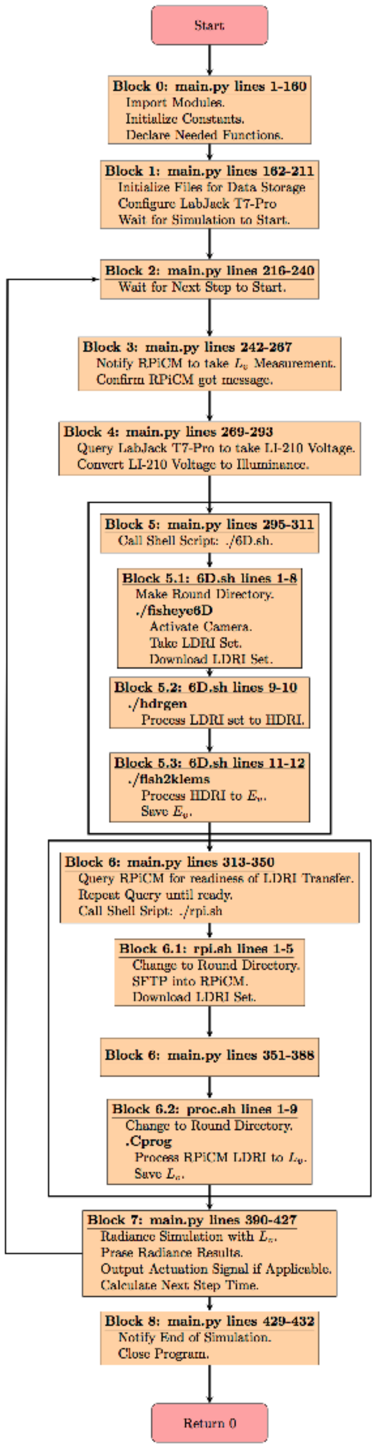
The following code is written in Python 2.7 and runs on the MacBook Pro. It runs the entire CUBE 2.0 system from initializing measurements, transporting data, processing data, and executing simulations, to saving results.

```
# Author:  Alex R. Mead
# Date: May 2016
# Description:
# Contained here is the python code for the CUBE project. The is
    the main() function which will run on the MacBooook Pro
# (MBP).  It opens sockets to talk to the RPiCM and also the
    LabJack connected to the LI-COR 210.  Finally, it connects
# also to the 6D through a C program which is called from this
    script. After that it downloads the fisheye6D photos locally
# and creates and HDR photo (.hdr) with hdrgen.  Then groups both
    the input and output luminance measurements into files
# which are used for the system ID process.

import sys

...
```

```

import numpy as np

from labjack import ljm
from time import sleep

# Constants
RPiCM_IP = "10.0.0.13"

...

names = ["AIN0_RANGE", "AIN0_RESOLUTION_INDEX", "AIN0_SETTLING_US"
    ]

# Timer constants
loopStartTime = 0

...

loopTimeInSeconds = 60 # 1 minute

# Cyber Room Dimensions
X=3.6

...

W = int(round((X-dx)/dx))

# Closed Loop: True is server is connected, false otherwise
closeLoop = False
BSDF_front = '/Users/alexmead/workspace/python/CUBE_RPi_HP/BSDF/
    twitchell_80.csv' # Larger hole

...

RequeryIntervalInSeconds = 1

# Function for Three-Phase with output measurem
def illumMeasure(result,out,W,L,outfile):

...

```

```
  return 0

# Function for Three-Phase with input measurem
def illumSimulate(result,out,W,L,outfile,BSDF_front,round):

...

  return 0

# Function for dark current measurement
def darkCurrent(handle):
...
    return darkSignal

def main():
    loopWaitTime = 0
    # The main is a state machine architecture, looping though
        various states of action based on interaction with

    ...

        print("    %s : %f" % (names[i], aValues[i]))

    # The actual Finite State Machine (FSM) while loop.
    while(RUNNING[0] == True):

        if("idle" == STATE):
            # Initialize the loop timer

                            ...

            # Change state
            STATE = "NotifyRPiCM"

        elif("NotifyRPiCM" == STATE):
            # Send signal to RPiCM to take measurement.

            ...

            # Change state to "LJ" for LabJack to get the
                illuminance measurement.
            STATE = "LJ"

        elif ("LJ" == STATE):
            # Get readings from LabJack of illuminance

            ...

            # Change state to "6D" to contact the fisheye-6D
                camera and take pictures.
            STATE = "6D"

        elif("6D" == STATE):
            # Process the created HDR photo (.hdr): (1) Calibrate
                the photo using the LI-210, (2) Bin luminances into
                 Klems solid angles

                            ...

            # Change state
            STATE = "RetrieveRPiCM"


        elif("RetrieveRPiCM" == STATE):
            print("Retrieve the RPiCM measurement.")

            ...

            # Change state
            STATE = "Radiance"

        elif("Radiance" == STATE):
            print("Executing Radiance simulation with newly
                collected measurement.\n")

            ...

            # Change State
            STATE = "idle"

        # Increments the counter so we can see how many times the
            FSM "ticks".
        RUNNING[1] = RUNNING[1] + 1
        print("Cycle number: " + str(RUNNING[1]))
```

```

    print("Big John has called it...")
    ljm.eWriteName(handle, "DIO0_EF_CONFIG_A", 120000)  # Open the
        window

    sys.exit(0)

# Boiler plate used to run script as function.
if __name__ == "__main__":
    main()
```


Start
Block 0: main.py lines 1-160
Import Modules.
Initialize Constants.
Declare Needed Functions.
Block 1: main.py lines 162-211
Initialize Files for Data Storage
Configure LabJack T7-Pro
Wait for Simulation to Start.
Block 2: main.py lines 216-240
Wait for Next Step to Start.
Block 3: main.py lines 242-267
Notify RPiCM to take L_v Measurement.
Confirm RPiCM got message.
Block 4: main.py lines 269-293
Query LabJack T7-Pro to take LI-210 Voltage.
Convert LI-210 Voltage to Illuminance.
Block 5: main.py lines 295-311
Call Shell Script: ./6D.sh.
Block 5.1: 6D.sh lines 1-8
Make Round Directory.
./fisheye6D
Activate Camera.
Take LDRI Set.
Download LDRI Set.
Block 5.2: 6D.sh lines 9-10
./hdrgen
Process LDRI set to HDRI.
Block 5.3: 6D.sh lines 11-12
./fish2klems
Process HDRI to E_v.
Save E_v.
Block 6: main.py lines 313-350
Query RPiCM for readiness of LDRI Transfer.
Repeat Query until ready.
Call Shell Sript: ./rpi.sh
Block 6.1: rpi.sh lines 1-5
Change to Round Directory.
SFTP into RPiCM.
Download LDRI Set.
Block 6: main.py lines 351-388
Block 6.2: proc.sh lines 1-9
Change to Round Directory.
.Cprog
Process RPiCM LDRI to L_v.
Save L_v.
Block 7: main.py lines 390-427
Radiance Simulation with L_v.
Prase Radiance Results.
Output Actuation Signal if Applicable.
Calculate Next Step Time.
Block 8: main.py lines 429-432
Notify End of Simulation.
Close Program.
Return 0

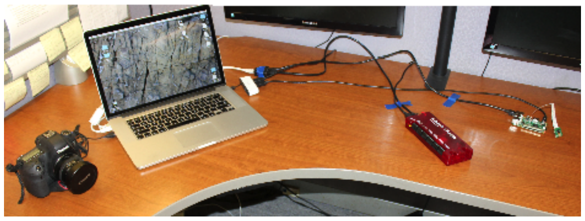


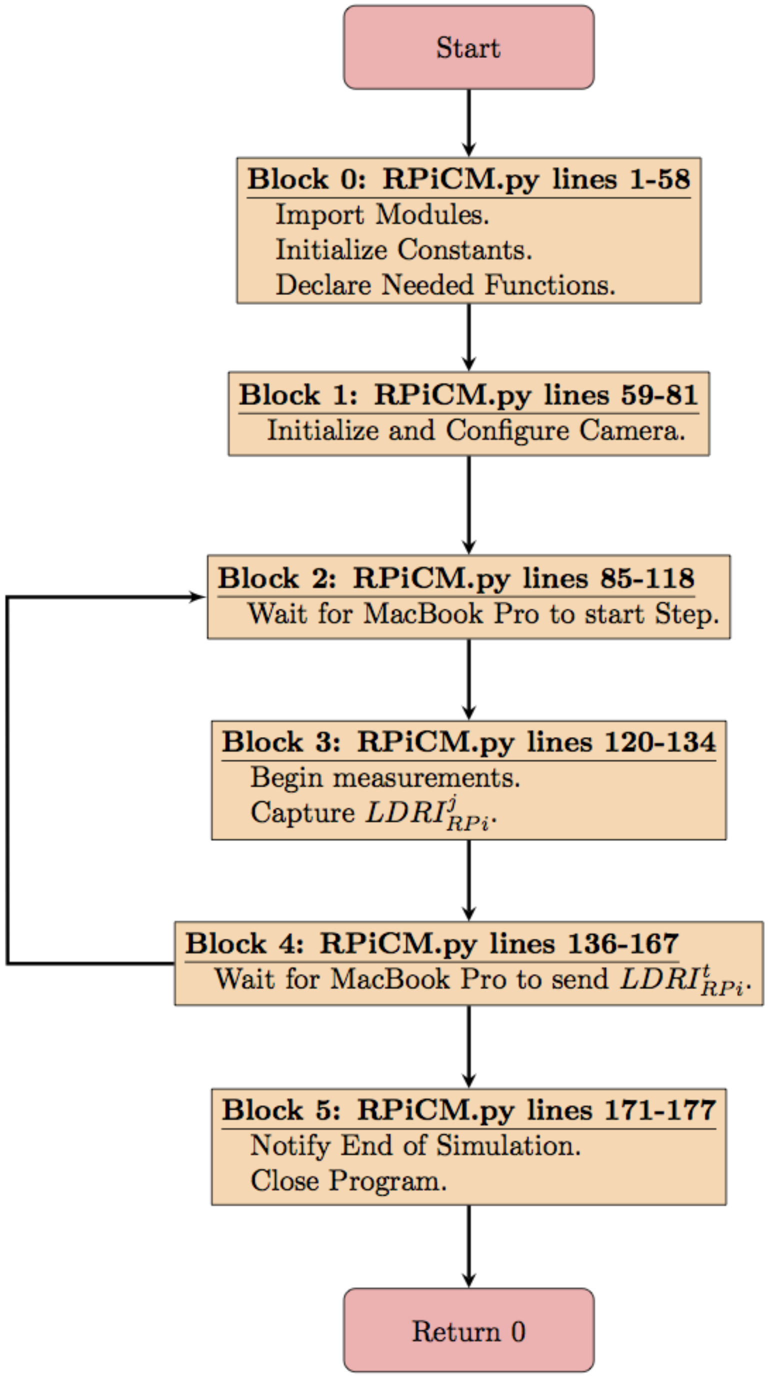
Figure D.2: Bench-top deployment of CUBE 2.0 hardware and software components.

D.2 RPiCM()

The following code is written in Python 2.7 and runs on the RPiCM. It acts as a server and takes the $LDRI^j_{RPi}$ measurements.

```
# Author:  Alex R. Mead
# Date: May 2016
# Description:
# This code will run on the Raspberry Pi 2 with Camera Module (
    RPiCM) to interact with the MacBookPro (MBP). It will
# wait for the MBP to request a measurement, take a measurement,
    then wait for the MBP to request the latest
# measurement, at which time it will send the measurement to the
    MBP.

# UPDATE: The MBP now logs into the RPiCM using an SFTP script and
    downloads the LDRI set to itself. This
# has been found to be far faster than processing the LDRI set
    locally on the RPiCM.

# Same as the MBP this system is designed to run like finite state
    machine.

from __future__ import (
    unicode_literals,
    absolute_import,
    print_function,
    division,
    )

import sys

```

```
...

from time import sleep

# Constants
#IP = '127.0.0.1' # Testing on a single machine
IP = '0.0.0.0' # Actual run time.

...

PIXEL_WIDTH = 2592

# Takes in the Camera object instantiated at the beginning of the
    run and takes the desired SS expsoures with it.
def HDRI(SS,camera):

    ...

    # Return the list of filenames to the calling function.
    return names

def main():

    STATE = "idle"
    RUNNING = [True,0]

    ...

    sleep(2.0)

    while(RUNNING[0]):

        if("idle" == STATE):
            print("We're␣in␣idle␣state␣waiting␣to␣hear␣from␣the␣
                MacBookPro...")

            ...

            # Change state: Continue on and take measurement
            STATE ="Measuring"

        elif ("Measuring" == STATE):
                print("Engaging the Camera Module to measure the output of the CFS under test...")

                ...

                # Continue to the next state and wait for MBP to query for the 145 measurements.
                STATE = "waitToSend"

            elif("waitToSend" == STATE):
                print("Waiting to send to MBP...")

                ...

                # Change state: measurement has been sent, change state back to idle
                STATE = "idle"

            # Cycle counter for the finite state machine
            RUNNING[1] = RUNNING[1] + 1
            print("Cycle number: " + str(RUNNING[1]))

        print("Little John called it...")

        sys.exit(0)

if __name__ == "__main__":
        main()
```


Start
Block 0: RPiCM.py lines 1-58
Import Modules.
Initialize Constants.
Declare Needed Functions.
Block 1: RPiCM.py lines 59-81
Initialize and Configure Camera.
Block 2: RPiCM.py lines 85-118
Wait for MacBook Pro to start Step.
Block 3: RPiCM.py lines 120-134
Begin measurements.
Capture $LDRI^j_{RPi}$.
Block 4: RPiCM.py lines 136-167
Wait for MacBook Pro to send $LDRI^t_{RPi}$.
Block 5: RPiCM.py lines 171-177
Notify End of Simulation.
Close Program.
Return 0

D.3 6D()

```
#!/bin/sh
  # $1 : round number
  # $2 : illuminance value
  # $3 : klemsINx.csv
  mkdir $1
  cd $1
  # Take the +-5 EC pictures and download to local directory
  ./../fisheye6D_11
  # Construct the .hdr from the EC varying pictures
  ./../hdrgen_macosx/bin/hdrgen -o fisheye_6D.hdr -r ../6Dcam.rsp
     -c XYZ IMG_*
  # Process fisheye image (.hdr) into a klems input vector
  ./../fish2klems19D fisheye_6D.hdr $2 $3
  cp $3 ../$3
  exit
```

D.4 fisheye6D

```
//
//  main.c
//  fisheye6D
//
//  Created by Alex Mead on 6/7/16.
//  Copyright    2016 Alex Mead. All rights reserved.
//
//  First try with the new EDSDK 3.4 for Mac OSX 10.11 (El Capitan
   ).
//  Will try something simple to start, then get more complicated.
//

#include <stdio.h>
...
#include "EDSDK.h"

EdsError takePicture(EdsCameraRef camera)
{
    ...

    return error;
}

EdsError movePictures(EdsCameraRef camera, int N)
{
    EdsError error = EDS_ERR_OK;
    ...
    return error;
}

// Declaring the callback function pointer
typedef EdsError (EDSCALLBACK *EdsObjectEventHandler)(
   EdsObjectEvent event, EdsBaseRef object, EdsVoid * context);

// Global Variables - I'm not sure if this is 'best practice' but
   it works at this stage.
int count = 0;

bool looper = true;
bool pictureAgain = false;
CFRunLoopRef rf;

// Note: In reality exposure value (EV) is not correct here,
   rather exposure compensation (EC) should be used.
//       This error was caused by the Harvard/MIT documents
   refering to the wrong values and the correction
//       being discovered by me when I wrote the illuminance vs.
   luminance calibration paper.
// EV=-5,..,+4 <- uncomment the next two lines
int picturesNeeded = 11;
EdsUInt32 EV[11] = {0xD8,0xE0,0xE8,0xF0,0xF8,0x00,0x08,0x10,0x18,0
   x20,0x28}; // EV = -5,..,+5

// EV=-4,..,+4 <- uncomment the next two lines
//int picturesNeeded = 9;
//EdsUInt32 EV[9] = {0xE0,0xE8,0xF0,0xF8,0x00,0x08,0x10,0x18,0x20
   }; // EV = -4,..,+4

// EV=-3,..,+3 <- uncomment the next two lines
//int picturesNeeded = 7;
//EdsUInt32 EV[7] = {0xE8,0xF0,0xF8,0x00,0x08,0x10,0x18}; // EV =
   -3,..,+3

```

```
EdsUInt32 EVSize;
EdsCameraRef camera = NULL;

EdsError EDSCALLBACK handleObjectEvent(EdsObjectEvent event,
    EdsBaseRef object, EdsVoid * context)
{
    EdsError error = EDS_ERR_OK;

    ...

    return error;
}

int main(int argc, const char * argv[]) {

    ////////////// 1/4 //////////////
    // First declare the needed variables we'll need to talk to a
        camera
    bool isSDKLoaded = false;
    ...
    EdsUInt32 counT = 0;
    ////////////// 1/4 //////////////

    ////////////// 2/4 //////////////
    // Initialize the EDSDK
    error = EdsInitializeSDK();
    ...
    }
    ////////////// 2/4 //////////////

    ////////////// 3/4 //////////////
    // Get the first camera
    if(error == EDS_ERR_OK){

        ...

    }
    ////////////// 3/4 //////////////

    ////////////// 4/4 //////////////
    // Get first camera retrieved
    if (error == EDS_ERR_OK) {
        error = EdsGetChildAtIndex(cameraList, 0, &camera);
    }
    ////////////// 4/4 //////////////

    // Code will connect to camera, take a series of pictures,
       then disconnect
    //
       /////////////////////////////////////////////////////////////////

    // Open session with camera
    if(error == EDS_ERR_OK)
    {
        error = EdsOpenSession(camera);
        sleep(0.5);
    }

    // Set object event handler
    if(error == EDS_ERR_OK)
    {
        error = EdsSetObjectEventHandler(camera,
           kEdsObjectEvent_All, handleObjectEvent, NULL);
    }

    // Get the current runloop
    rf = CFRunLoopGetCurrent();

    // Set EV & take first photo
    error = EdsSetPropertyData(camera,
       kEdsPropID_ExposureCompensation, 0, sizeof(EVSize), &EV[0])
       ;  // Set EV[0]
    error = takePicture(camera);  // Take Picture

    // Loop through for each picture
    while(looper){
        // Run-loop activation
        CFRunLoopRunInMode(kCFRunLoopDefaultMode, 0.01, true);

        // Take another picture
        if(pictureAgain){

            // Check error
            if(error == EDS_ERR_OK)
            {
                error = EdsCloseSession(camera);
            }

            // Open session with camera
            if(error == EDS_ERR_OK)
            {
                //sleep(1);
                error = EdsOpenSession(camera);
            }
            error = EdsSetPropertyData(camera,
               kEdsPropID_ExposureCompensation, 0, sizeof(EVSize),
                &EV[count]); // Set EV[0]

            error = takePicture(camera);  // Take Picture
            pictureAgain = false;
        }

    // Download the photos from the 6D to the MacBook Pro
    error = movePictures(camera, picturesNeeded);


    return 0;
}
```

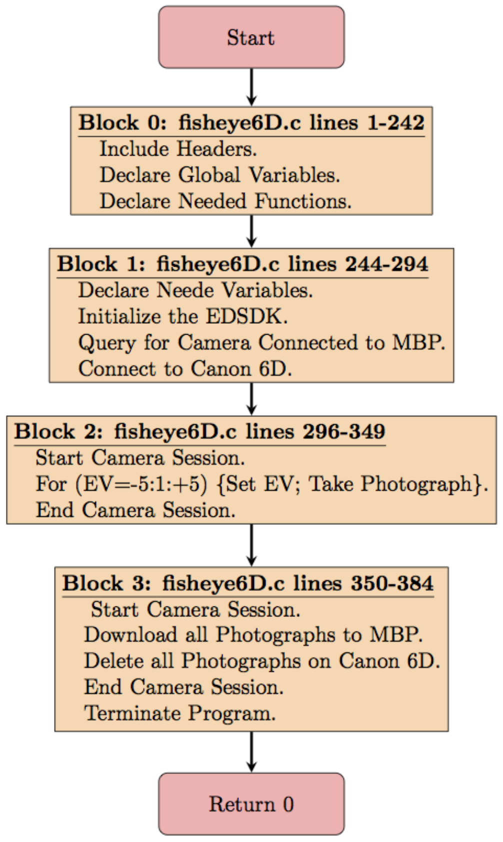

Start
Block 0: fisheye6D.c lines 1-242
Include Headers.
Declare Global Variables.
Declare Needed Functions.
Block 1: fisheye6D.c lines 244-294
Declare Neede Variables.
Initialize the EDSDK.
Query for Camera Connected to MBP.
Connect to Canon 6D.
Block 2: fisheye6D.c lines 296-349
Start Camera Session.
For (EV=-5:1:+5) {Set EV; Take Photograph}.
End Camera Session.
Block 3: fisheye6D.c lines 350-384
Start Camera Session.
Download all Photographs to MBP.
Delete all Photographs on Canon 6D.
End Camera Session.
Terminate Program.
Return 0

Figure D.4: The program fisheye6D represented as a graph.

D.5 hdrgen

The program hdrgen comes precompiled and ready for use, therefore,no source code is provided here.

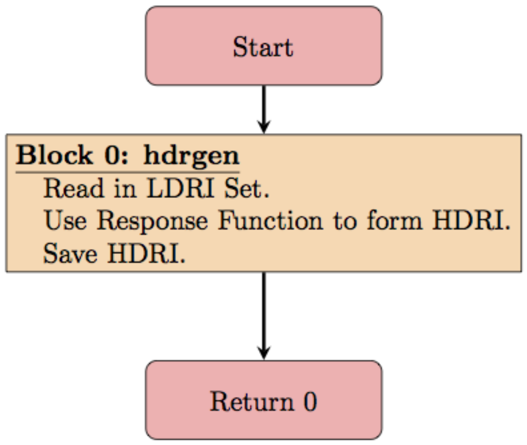


Figure D.5: The program program hdrgen represented as a graph.

D.6 fish2klems

```
// Author: Alex R. Mead, some functions taken from Greg Ward work.
// Date: May 2016
// Description: This program reads in an HDR image made by hdrgen
    from the fisheye lens
// equipped 6D.  At first stage it will just read in picture and
    save the Y of the
// CIE-XYZ valus to a csv.  Eventually it will average the values
    with respect ot the
// Klems input solid angles.

#include <stdio.h>
...
#include "fisheyeFinal.h"

// Declared needed global variables
int xmax, ymax;
double scale;

// Correct for vignetting and cosine
```

```
bool BOOLcorCos = false;
bool SLIM = false;

/* print message and exit */
int quiterr(err)
char  *err;
{
        ...
}

/* Reads the hdr photo produced by hdrgen, outputs a .CSV file for
     easy Matlab viewing. */
int ra2csvSLIM(FILE *inPutpt, picRow *picture){
        ...
        return 0;
}

/* bins the luminance values into klems solid angles */
int lum2klemsCalibrate(picRow *picture, double *klems, double *
   mins, double *maxs){
        ...
return 0;
}

/* bins the luminance values into klems solid angles */
int lum2klemsIlluminance(picRow *picture, double *klems){
        ...
return 0;
}

double calibrate(picRow * picture, double illum){
        ...

return correction;
}

int main(int argc, char *argv[]){

  // Holds results from function calls to Greg Ward's parsers of .
     hdr files.
  int result;

```

```
// Used to hold the illuminance value taken from the LabJack T7-
    Pro and the correction
// value calculated from illuminance meter and HDRI fisheye
    picture.
double illum, correction;



// Holder for the klems values
double klems[145],*klemsPtr=&klems[0];
double mins[145], *minsPtr=&mins[0];
double maxs[145], *maxsPtr=&maxs[0];

// Declare file objects pointers for input and output files
FILE* inPut, *klemsTotal;

// Resolution structure and pointer
RESOLU resol;
RESOLU* resolpt=&resol;

// Open the .hdr file which is passed as the first argument for
    reading.
inPut = fopen(argv[1],"r");

// Check header - must call this before the 'fgetsresolu(.,.)'
    to get the header out of
// stream. This is a Ward function.
result = checkheader(inPut,CIEFMT,stdout);
printf("\nThe result of checkheader is: %d\n",result);

// Get the resolution string. This is a Ward function
result = fgetsresolu(resolpt,inPut);

// Print xmax, ymax
printf("Value X: %d\nValue Y: %d\n",resolpt->xr,resolpt->yr);

// Initialize xmax and ymax
xmax = resolpt->xr;
ymax = resolpt->yr;

// Get memory for the luminance data for the fisheye-6D picture.
picRow *picture;
picture = (picRow *)malloc(ymax*sizeof(picRow));
```

```
  
  // Calling the actual function which reads in pixel values,
      creates the output file.
  result = ra2csvSLIM(inPut,picture);

  // Error check of the pixel parsing and outfile writing function
  if(result!=0){
  printf("Error writing file...");
  return 1;
  }

  // Correct the luminance data in the picture file to correspond
      with the illuminance
  // number measured using the LabJack T7-Pro.
  printf("The illuminance input: %f\n",atof(argv[2]));
  illum = atof(argv[2]);

  // Using the input value of illuminance calculate the
      illuminance according to the
  // fisheye.hdr (with vignetting correction), then calculate the
      correction factor, then
  // apply the correction factor to the fisheye.hdr image. Return
      the correction factor
  // value in case it is needed.
  correction = calibrate(picture, illum);

  // Loop through the photo and bin the pixels into the klems
      solid angles, note, bins are now illuminance
  result = lum2klemsIlluminance(picture,klemsPtr);


  // Save the Klems totals to a file
  klemsTotal = fopen(argv[3],"w");
  for(int k=0;k<144;k++){
        fprintf(klemsTotal,"%f\n",klems[k]);
  }
  fprintf(klemsTotal,"%f",klems[144]);
  fclose(klemsTotal);

  return 0;
}
```

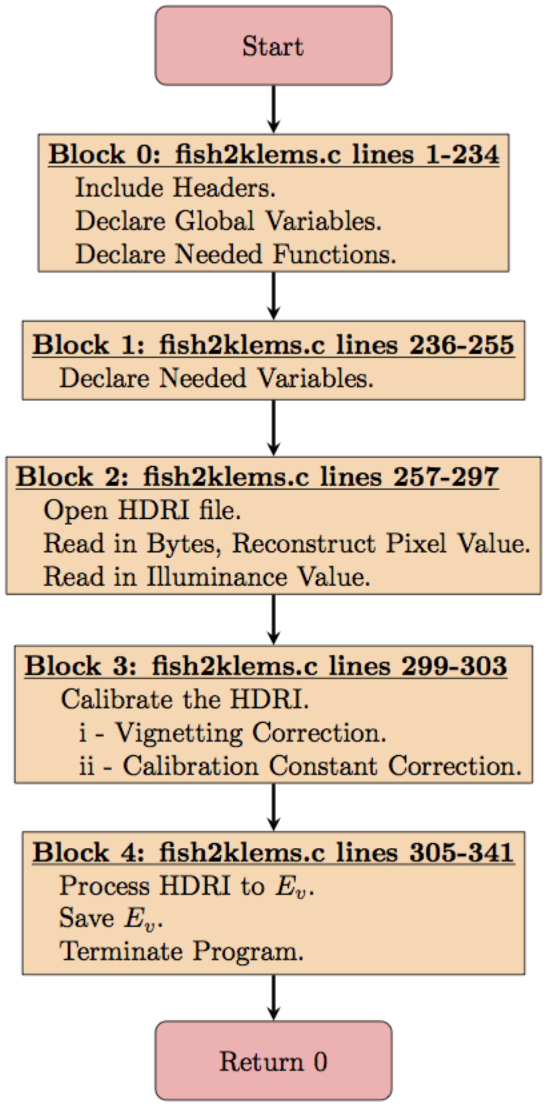

Start
Block 0: fish2klems.c lines 1-234
Include Headers.
Declare Global Variables.
Declare Needed Functions.
Block 1: fish2klems.c lines 236-255
Declare Needed Variables.
Block 2: fish2klems.c lines 257-297
Open HDRI file.
Read in Bytes, Reconstruct Pixel Value.
Read in Illuminance Value.
Block 3: fish2klems.c lines 299-303
Calibrate the HDRI.
i - Vignetting Correction.
ii - Calibration Constant Correction.
Block 4: fish2klems.c lines 305-341
Process HDRI to E_v.
Save E_v.
Terminate Program.
Return 0

Figure D.6: The program fish2klems represented as a graph.

D.7 rpi

```
#!/bin/sh
  # $1 : round number
  cd $1
  sftp -b ../ssh2RPiCM.sh pi@10.0.0.13
  exit
```

D.8 ssh2RPiCM

```
#!/bin/sh
  cd Desktop/realCUBE
  get *.data
  exit
```

D.9 proc

```
#!/bin/sh
  # $1 : round number
  # $2 : klemsOUTx.csv
  cd $1
  # Commented 4-6 for speed increase
  #./../Cprog 1000000.data 100000.data 10000.data 1000.data 100.
     data 10.data $2
  ./../Cprog10F 1000000.data 100000.data 10000.data $2
  cp $2 ../$2
  exit
```

D.10 Cprog

```
// Author: Alex R. Mead
// Date: May 2016
// Description: This code is the C program running on the MBP
    which will process the
// .jpg+RAW photos produced by the Python program.  This program
    is called because it is
// much faster to process the photos in C than it is in Python and
     further to process
// them on the MBP versus the RPiCM.

#include <stdio.h>
```

```
#include <stdbool.h>
#include "OFlocations.h"
#include "scalers.h"

// M matrix for RPiCM-RGB -> CIE-XYZ transformation
double M[3] = {0.0085884,1.1721,-0.25675};

/* Initializing double arrays to hold photo data. */
short pic106[1944][2592];
short pic105[1944][2592];
short pic104[1944][2592];

// Parses the RAW RPiCM pictures and stores them in double arrays
   for further processing.
void picProcess(short* arrayHead,char *fileName ){
   // Parses a .jpg+RAW photo file from the RPiCM into a double
      array for the actual pixel data.

   ...

   // Close the picture input file we read the raw data from.
   fclose(pic);

}

// Saves the array RPiCM exposure as a .csv file for further
   processing and investigation.
void saveArray(short* arrayHead, char *fileName){

  // Variables to use
  FILE *fpOUT;
  int i, j;

  ...

   // Close the file and return to main()
   fclose(fpOUT);

}

// Returns the max value in an array of ints
int maxValue(int myArray[], size_t size) {
    // Get length of array

    ...

    return maxV;
}

// Returns a descending ordered array of ints
void orderedArray(int myArray[], size_t size){

  // Local Variables
  int i, j, temp;

  ...

}

// Takes in array RPiCM pictures of optical fiber ends and returns
    a calibrated luminance value
// for each outgoing klems angle
void processPictureMeasurements(float* storeLum, short* pictures
   []){

   // Values needed in loops
   FILE *klemsData, *OFout;

   ...

     } // End of getting the current OF's pixels from end of fiber
         .

     // 2) Process the exposures to get an HDR measurement. Save
         that measurement to the luminance holder array.
     // Process the HDR measurement of the end of the Optical
         Fibers
     // Check for saturation of first exposure
           if(maxValue(ends[0][0], sizer)==1023||maxValue(ends
              [0][1], sizer)==1023||maxValue(ends[0][2], sizer)
              ==1023){
             // Check for saturation of second exposure
             if(maxValue(ends[1][0], sizer)==1023||maxValue(ends
                [1][1], sizer)==1023||maxValue(ends[1][2], sizer)
                ==1023){
                // Shutter two exposure saturated, thus use
                   shutter three exposure

                // i  - order the luminance by magnitude
                orderedArray(ends[2][0], sizer);
                orderedArray(ends[2][1], sizer);
                orderedArray(ends[2][2], sizer);

                // ii - average the top XXX values
                for(int j=0; j<pixCount; j++){
                  avgLumR = avgLumR + ends[2][0][j];
                  avgLumG = avgLumG + ends[2][1][j];
                  avgLumB = avgLumB + ends[2][2][j];
                }
                avgLumR = (avgLumR / pixCount) - darkSignal;
                avgLumG = (avgLumG / pixCount) - darkSignal;
                avgLumB = (avgLumB / pixCount) - darkSignal;

                // iii   - convert to luminance
                luminance = (M[0]*avgLumR + M[1]*avgLumG + M[2]*
                   avgLumB);

                // iv  - convert to luminance via slope and
                   intercept
                storeLum[optFib] =  scalers[optFib][2]*luminance +
                     offSets[optFib][2];

              }
              else{
              // shutter one saturated, but not shutter two, thus
                 use shutter two

              // i  - order the luminance by magnitude
              orderedArray(ends[1][0], sizer);
              orderedArray(ends[1][1], sizer);
              orderedArray(ends[1][2], sizer);

              // ii - average the top XXX values
              for(int j=0; j<pixCount; j++){
                avgLumR = avgLumR + ends[1][0][j];
                avgLumG = avgLumG + ends[1][1][j];
```

```
                    avgLumB = avgLumB + ends[1][2][j];
                }
                avgLumR = (avgLumR / pixCount) - darkSignal;
                avgLumG = (avgLumG / pixCount) - darkSignal;
                avgLumB = (avgLumB / pixCount) - darkSignal;

                // iii   - convert to luminance
                luminance = (M[0]*avgLumR + M[1]*avgLumG + M[2]*
                    avgLumB);

                // iv  - convert to luminance via slope and
                    intercept
                storeLum[optFib] =  scalers[optFib][1]*luminance +
                    offSets[optFib][1];
            }
        }
        else{

          ...

        }

  } // Looped through each Optical Fiber End

}

int main(int argc, char *argv[]){

  // Housing keeping variables
  int i;

  // Pointers to double arrays to store processed data
  short* p106=&pic106[0][0];
  short* p105=&pic105[0][0];
  short* p104=&pic104[0][0];

  // An array of pointers to the double arrays holding the photos
      data
  short* pics[3] = {&pic106[0][0],&pic105[0][0],&pic104[0][0]};

  // File pointer, array, and array pointer for final KlemsOUT
      measurement
  FILE *klemsData, *OFout;
  float lumMsr[145], *lumMsrPtr=&lumMsr[0];

  // Process the picture measurements - 3 are used for real run as
      6 is too much range.
  // Picture 1
  picProcess(p106,argv[1]);
  // Picture 2
  picProcess(p105,argv[2]);
  // Picture 3
  picProcess(p104,argv[3]);

  // Call function which uses the above processed photos to get
     klems values.
  // Here it reads in the pixel locations of the optical fiber
     ends from a header file
  // created as a result of a Matlab-Pyton script combination.
  processPictureMeasurements(lumMsrPtr,pics);

  // Save the final luminance values which have been constructed
     from the RPiCM exposures
  // of the ends of the optical fibers.
  // Open the file
  OFout = fopen(argv[4],"w");
  for(int k=0;k<145;k++){
     fprintf(OFout,"%f\n",lumMsr[k]);
  }
  // Close the file streams
  fclose(OFout);

  return 0;
}
```

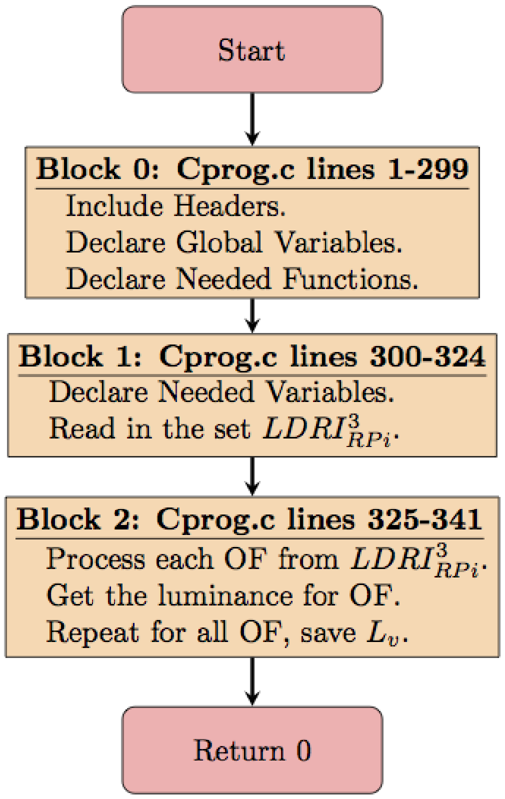

Start
Block 0: Cprog.c lines 1-299
Include Headers.
Declare Global Variables.
Declare Needed Functions.
Block 1: Cprog.c lines 300-324
Declare Needed Variables.
Read in the set $LDRI^3_{RPi}$.
Block 2: Cprog.c lines 325-341
Process each OF from $LDRI^3_{RPi}$.
Get the luminance for OF.
Repeat for all OF, save L_v.
Return 0

Figure D.7: The program Cprog represented as a graph.

D.11 fish2klemsC

```
// Author: Alex R. Mead, some functions taken from Greg Ward work.
// Date: May 2016
// Description: This program reads in an HDR image made by hdrgen
    from the fisheye lens
// equipped 6D.  At first stage it will just read in picture and
    save the Y of the
// CIE-XYZ valus to a csv.  Eventually it will average the values
    with respect ot the
// Klems input solid angles.

#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include "localheaders/color.h"
#include "localheaders/resolu.h"
#include "fisheyeFinal.h"

// Declared needed global variables
int xmax, ymax;
double scale;

// Correct for vignetting and cosine
bool BOOLcorCos = false;
bool SLIM = false;

/* print message and exit */
int quiterr(err)
char  *err;
{
        if (err != NULL) {
                //fprintf(stderr, "%s: %s\n", progname, err);
                printf("\nquiterr␣was␣called!\n");
                exit(1);
        }
        exit(0);
}

/* Reads the hdr photo produced by hdrgen, outputs a .CSV file for
     easy Matlab viewing. */
int ra2csvSLIM(FILE *inPutpt, picRow *picture){
        COLR    *scanin;

        ...

        free((void *)scanin);

        return 0;
}

/* bins the luminance values into klems solid angles */
int lum2klemsCalibrate(picRow *picture, double *klems, double *
   mins, double *maxs){
        int x, y, k;

        ...

return 0;
}

/* bins the luminance values into klems solid angles */
int lum2klemsIlluminance(picRow *picture, double *klems){
        int x, y, k;

        ...

return 0;
}

double calibrate(picRow * picture, double illum){

        // Variables we will need

        ...

        // Return the correction value to main().
return correction;
}

int main(int argc, char *argv[]){

  // Holds results from function calls to Greg Ward's parsers of .
     hdr files.
  int result;

  // Used to hold the illuminance value taken from the LabJack T7-
     Pro and the correction
```

```
// value calculated from illuminance meter and HDRI fisheye
    picture.
double illum, correction;


// Holder for the klems values
double klems[145],*klemsPtr=&klems[0];
        ...

// Declare file objects pointers for input and output files
FILE* inPut, *klemsTotal;

// Resolution structure and pointer
RESOLU resol;
RESOLU* resolpt=&resol;

// Open the .hdr file which is passed as the first argument for
    reading.
inPut = fopen(argv[1],"r");

// Check header - must call this before the 'fgetsresolu(.,.)'
    to get the header out of
// stream. This is a Ward function.
result = checkheader(inPut,CIEFMT,stdout);
printf("\nThe␣result␣of␣checkheader␣is:␣%d\n",result);

// Check to ensure the hdr file is in CIE-XYZ format, else
    terminate.
if(result<0){return 1;}

// Get the resolution string. This is a Ward function
result = fgetsresolu(resolpt,inPut);

// Check to ensure the hdr file resolution returned properly,
    else terminate.
if(result<0){return 1;}

// Print xmax, ymax
printf("Value␣X:␣%d\nValue␣Y:␣%d\n",resolpt->xr,resolpt->yr);

// Initialize xmax and ymax
xmax = resolpt->xr;
```

```
ymax = resolpt->yr;

// Get memory for the luminance data for the fisheye-6D picture.
picRow *picture;
picture = (picRow *)malloc(ymax*sizeof(picRow));

// Calling the actual function which reads in pixel values,
    creates the output file.
result = ra2csvSLIM(inPut,picture);

// Error check of the pixel parsing and outfile writing function
if(result!=0){
printf("Error writing file...");
return 1;
}

// Correct the luminance data in the picture file to correspond
    with the illuminance
// number measured using the LabJack T7-Pro.
printf("The illuminance input: %f\n",atof(argv[2]));
illum = atof(argv[2]);

// Using the input value of illuminance calculate the
    illuminance according to the
// fisheye.hdr (with vignetting correction), then calculate the
    correction factor, then
// apply the correction factor to the fisheye.hdr image. Return
    the correction factor
// value in case it is needed.
correction = calibrate(picture, illum);

// What needs to be done now is get the illuminance per klems
    basis division. This will
// be done by adding for each klems basis division the cosine
    scaled and solid angle scaled
// pixel values.  These will then be saved, and these are the
    input vectors.

//////   For calibration, however, we are not interested in the
     ** illuminance **, but
// rather the ** luminance ** traveling in the device. Hence,
    all that will be used for
```

```
  // calibration is the average luminance (calculated by a solid angle weighted average of
  // the pixels in each klems basis division). This is the luminance passing through the
  // CUBE 2.0 aperture and striking the corresponding exiting solid angle, where it is sampled
  // by the optical fiber end, which assumes an average luminance value for its coresponding
  // solid angle of the exciting luminance of the panel.

  // Loop through the photo and bin the pixels into the klems solid angles
  result = lum2klemsCalibrate(picture,klemsPtr,minsPtr,maxsPtr);

  // Loop through the photo and bin the pixels into the klems solid angles, note, bins are now illuminance
  //result = lum2klemsIlluminance(picture,klemsPtr);

  // Error check of the pixel parsing and outfile writing function
  if(result!=0){
  printf("Error writing file...");
  return 1;
  }

  // Save the Klems totals to a file
  klemsTotal = fopen(argv[3],"w");
  for(int k=0;k<144;k++){
        fprintf(klemsTotal,"%f\n",klems[k]);
  }
  fprintf(klemsTotal,"%f",klems[144]);
  fclose(klemsTotal);

  // Close the input file and exit.
  fclose(inPut);

  return 0;
}
```

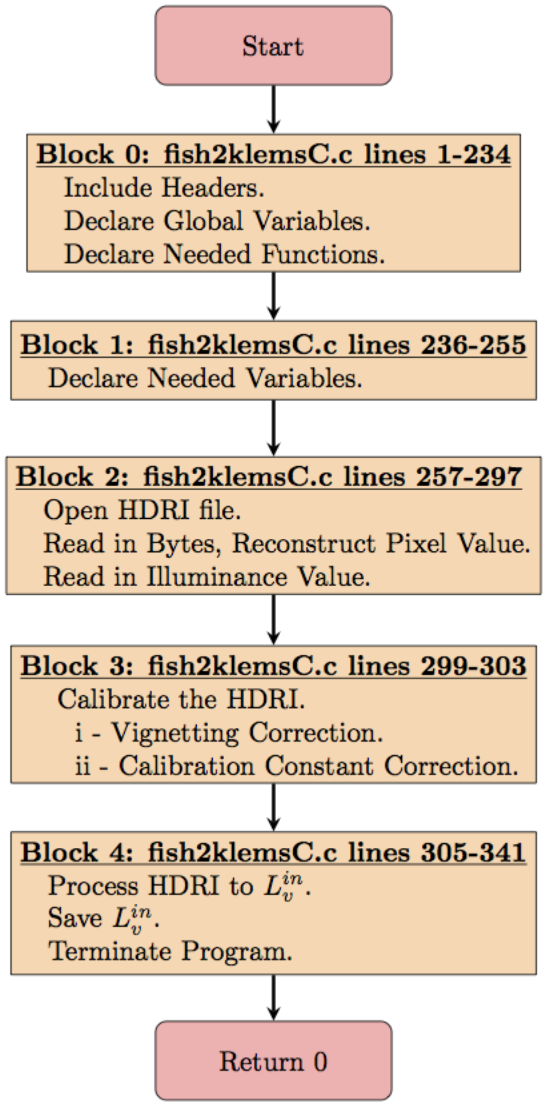

Start
Block 0: fish2klemsC.c lines 1-234
Include Headers.
Declare Global Variables.
Declare Needed Functions.
Block 1: fish2klemsC.c lines 236-255
Declare Needed Variables.
Block 2: fish2klemsC.c lines 257-297
Open HDRI file.
Read in Bytes, Reconstruct Pixel Value.
Read in Illuminance Value.
Block 3: fish2klemsC.c lines 299-303
Calibrate the HDRI.
i - Vignetting Correction.
ii - Calibration Constant Correction.
Block 4: fish2klemsC.c lines 305-341
Process HDRI to L_v^{in}.
Save L_v^{in}.
Terminate Program.
Return 0

Figure D.8: The program fish2klemsC represented as a graph.

D.12 CprogC

```
// Author: Alex R. Mead
// Date: May 2016
// Description: This code is the C program running on the MBP
    which will process the
// .jpg+RAW photos produced by the Python program.  This program
    is called because it is
// much faster to process the photos in C than it is in Python and
     further to process
// them on the MBP versus the RPiCM.


#include <stdio.h>
...
#include "scalers.h"

// Make boolean varible
//typedef int bool;
//#define true 1
//#define false 0

// M matrix for RPiCM-RGB -> CIE-XYZ transformation
double M[3] = {0.0085884,1.1721,-0.25675};

/* Initializing double arrays to hold photo data. I'm not sure
    what "best␣practice" is
    for this type of operation. Perhaps calling malloc() in the
        main()? */
short pic106[1944][2592];
...
short pic101[1944][2592];

void picProcess(short* arrayHead,char *fileName ){
    // Parses a .jpg+RAW photo file from the RPiCM into a double
        array for the actual pixel data.

    // Declare needed variables for parsing the picture .jpg+RAW
        file.
    FILE *pic;

    ...

   }
   // Close the picture input file we read the raw data from.
   fclose(pic);

}

void saveArray(short* arrayHead, char *fileName){

  // Variables to use

  ...

}

void processPictureMeasurements(int* storeLum, short* pictures[],
   char *fileName){

   // Values needed in loops
   FILE *OFout, *klemsData;

         ...

   for(int optFib=0;optFib<145;optFib++){

         ...

            for(i=0;i<((N+1)*(N+1))-1;i++){fprintf(OFout,"%d,",ends[
               SS][0][i]);} fprintf(OFout,"%d\n",ends[SS][0][i]);
            for(i=0;i<((N+1)*(N+1))-1;i++){fprintf(OFout,"%d,",ends[
               SS][1][i]);} fprintf(OFout,"%d\n",ends[SS][1][i]);
            for(i=0;i<((N+1)*(N+1))-1;i++){fprintf(OFout,"%d,",ends[
               SS][2][i]);} fprintf(OFout,"%d\n",ends[SS][2][i]);
            for(i=0;i<((N+1)*(N+1))-1;i++){fprintf(OFout,"%d,",ends[
               SS][3][i]);} fprintf(OFout,"%d\n",ends[SS][3][i]);

         ...

   fclose(OFout);

}
```

```

int main(int argc, char *argv[]){

  // Housing keeping variables
  int i;

  // Pointers to double arrays to store processed data
  short* p106=&pic106[0][0];
        ...
  short* p101=&pic101[0][0];

  // An array of pointers to the double arrays holding the photos
     data
  short* pics[6] = {&pic106[0][0],&pic105[0][0],&pic104[0][0],&
     pic103[0][0],&pic102[0][0],&pic101[0][0]};

  // File pointer, array, and array pointer for final KlemsOUT
     measurement
  FILE *klemsData;
  int lumMsr[145], *lumMsrPtr=&lumMsr[0];

  // Process the picture measurements - could be better with
     dynamic allocation: malloc().
  // Picture 1
  picProcess(p106,argv[1]);

  ...

  picProcess(p101,argv[6]);

  /*  This is not done for the actual production run because we
     don't need these files
      to be looked at to ensure correctness. They are needed for
         the alignment though
  */
  // Saving the pictures to files - This takes a lot of time,
     maybe not do this?
  // For running trials let's still do this so we can examine them
      if need be.
  //saveArray(p106,argv[1]);

  ...

  //saveArray(p101,argv[6]);

  // Call function which uses the above processed photos to get
     klems values.
  // Here it reads in the pixel locations of the optical fiber
     ends from a header file
  // created as a result of a Matlab-Pyton script combination.
     Each time the CUBE2.0
  // RPiCM camera is taken off the back of the physical CUBE2.0 it
      must be realigned to
  // ensure the OF ends are within the boxes. RPiCM now nailed/
     glued on, so not an issue.
  // Used for calibration, will be commented out on real fun.

  processPictureMeasurements(lumMsrPtr,pics,argv[7]);

  return 0;
}
```

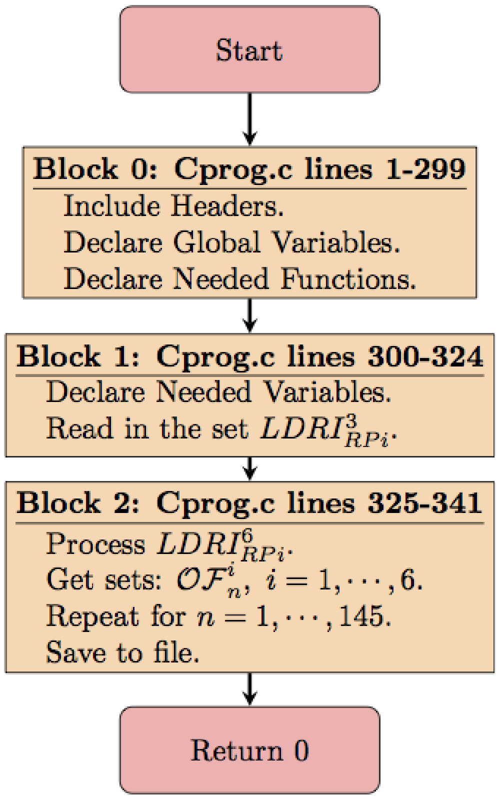

Start
Block 0: Cprog.c lines 1-299
Include Headers.
Declare Global Variables.
Declare Needed Functions.
Block 1: Cprog.c lines 300-324
Declare Needed Variables.
Read in the set $LDRI^3_{RPi}$.
Block 2: Cprog.c lines 325-341
Process $LDRI^6_{RPi}$.
Get sets: $\mathcal{OF}^i_n$, $i = 1, \cdots, 6$.
Repeat for $n = 1, \cdots, 145$.
Save to file.
Return 0

Figure D.9: The program CprogC represented as a graph.

D.13 headerMaker6D

```
% Author: Alex R. Mead
% Date: May 2016
% Description: This file takes the pixel dimension of a fisheye-6D
    HDR
% photo and produces the header files needed to analyze it with a
    C
% program.  The radius of the fisheye section on the picture must
    also be
% entered.
%
% Basically the program assigns each pixel in a photograph a theta
    and rho
% value. These values are then checked for each Klems section to
    determine
% if they sould be added or not. The rho value is also checked for
    both
% cosine and vignetting correction, however, they are precomputed
    here so
% they don't need to be checked and calculated at runtime by the C
    program.
%
% The header files produced are:
% Rinner - 145 radius values for each Klems section, inner radius
% Router - 145 radius values for each Klems section, outer radius
% thetaStart - 145 angles for each Klems section, beginning theta
    (rad)
% thetaEnd - 145 angles for each Klems section, ending theta (rad)

% corVig - matrix(n,m) with vignetting correction
% corCos - matrix(n,m) with cosine correction

% R - matrix(n,m) with the pixel's radius value in polar
    coordinates
% THETA - matrix(n,m) with the pixel's theta value in polar
    coordinates
% klems - matrix(n,m) with the klems basis path number for each
    pixel

clear all; close all; clc;

% Picture height and width:

Y = 480;
X = 720;
```

```
33 rad = 227;
34 fpixel = rad/(2*sind(90/2));
35
36 xShift = X/2;
37 yShift = Y/2;
38
39 % Create x and y matrices for the pixels
40 xBase = [0:1:X-1];
41 x = repmat(xBase,Y,1);
42 yBase = [0:1:Y-1]';
43 y = repmat(yBase,1,X);
44
45 % Shift the x and y matrices to align the origin as the center of the photo
46 % measurement.
47 x = x-xShift;
48 y = y-yShift+3; % Slighlty more shifting down that half way. A product of visual inspection of the picture.
49
50 % Convert from cartesian to polar coordinates. These are now the base theta
51 % and rho values for the associated pixels in the matrices position
52 [theta, rho] = cart2pol(x,y);
53
54 % Process rho and theta for proper orientations
55 % rho: must zero out pixels not in fisheye exposure
56 for i=1:Y
57         ...
58 end
59
60 % Write rho and theta to file
61
62 % Populate the Rinner, Router, thetaStart, thetaEnd
63
64 % Some holder constants
65
66
67 % 'angles' is precisely what needs changing, it is then implemented in lines 101 and 102 in the Rinner and Router calculations.
68
```

```
ang = [0,5,15,25,35,45,55,65,75,90];
angles = (2*fpixel*sind(ang/2))/rad;% This is the corrected value

% Loop through each ring
for j=1:1:9

    ...

end

% Write the Rinner, Router, thetaStart, thetaEnd matrices to files

% Calculate the correction factor with cosine and with vignetting
% Vignetting
corVig = rho;
thetaElevation = rho;
for i=1:Y
    for j=1:X
        if(corVig(i,j)~=-1)
            ...
        else
            ...
        end
    end
end

% Saving the theta elevation and vignetting correction functions
   to csv files
csvwrite('corVig.csv',corVig);
csvwrite('thetaElevation.csv',thetaElevation);

% Cosine: Convert the rho distance to angles, then apply the
   cosine. In
% reality we will want to apply the cosine correction factor for
   the LI210
% sensor, because its profile is not a perfect cosine corrector as
    assumed
% here right now.
corCos = rho;
for i=1:Y
    for j=1:X
        if(corCos(i,j)~=-1)
```

```
                ...
            else
                ...
            end
    end
end

% Saving the cosine weightings functions to csv files
csvwrite('corCos.csv',corCos);

% Preallocate memory for solid angle
solidAngle = zeros(Y,X);
countPixels = 0;
graphPixels = zeros(1,90);
patchAreas = zeros(1,90);
pixelSteradians = zeros(1,90);
% Make the steradian weighted matrix
for i=1:90
    ...

end

csvwrite('solAng.csv',solidAngle);

% Make a klems basis specifier
klems = zeros(Y,X);

% This loop assigns the klems value to each pixel explicitly
weights=zeros(145,1);
for k = 1:145
     ...
end

% Save klems file
csvwrite('klems.csv',klems);
```

```
# Author:  Alex R. Mead
# Date: May 2016
# Description: Used to take Matlab header file outputs and
   translate them to .h header
# files for use in the hdriproc.c program.

```

```
import io

def main():
  ################################################################
  # Rinner
  file = open('Rinner.csv','rU')
  data = file.readlines()
  file.close()

  # Drop the '\n' character
  for i in range(0,len(data)):
    data[i] = data[i][:-1]

  # Create double array string
  head = "float Rinner[145] = {"
  body = ""

  for j in range(0,len(data)-1):
    body = body + data[j]+ ','

  Rinner = head + body + data[-1] + '};'
  ################################################################
  file2 = open('fisheyeFinal.h','w')
  #file2.write(Rinner)
  #file2.write('\n')
  ################################################################

  ...

  ################################################################
  # klems
  file = open('klems.csv','rU')
  data = file.readlines()
  file.close()

  # Drop the '\n' character
  for i in range(0,len(data)):
    data[i] = data[i][:-1]

  # Create double array string
  depth = str(len(data))
  width = str(len(data[0].split(',')))
```

```
    head = "int klemsPic[" + depth + "][" + width + "] = {"
    body = ""

    for j in range(0,len(data)-1):
      body = body + '{' + data[j]+ '},'

    klemsPic = head + body + '{' + data[-1]+'}' '};'
    ####################################################################
    file2.write(klemsPic)
    file2.write('\n')
    file2.write('\n')
    ####################################################################
    # Constants needed in the code
    row = "typedef float picRow["+width+"];"

    ####################################################################
    file2.write(row)
    file2.close()

# Boiler Plate Code
if __name__ == "__main__":
    main()
```

D.14 headerMakerRPiCM

```
% Author: Alex Mead
% Date: May 2016
% Description: This script uses a quick plot to confirm if the
    Cprogram
% which processes the RPiCM photos is in fact getting the bit-wise
% operations correct.

% Clear previous figures
close all;
clear all;
clc

% First Shutter Speed
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% First import the CSV file which is produced.
```

```
%pic = csvread('OlignBlank.data');
%pic = csvread('OireUpRAW.jpg');
pic = csvread('Olign.data');


% General Graphing data
hor = 1:2592;
ver = 1:1944;

%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Define the search boxes for the end of the optical fiber
    locations.
% Original Box definition without reordering.

initialCenterXY = [
            % Ring 1
              1          1232           872;
            % Ring 2

              ...

           145           700          1202; ];
% Sorts the optical fiber number (Klems Patch) so the row_# =
    klems_#
initialCenterXY = sortrows(initialCenterXY,1);

% Preallocate memory for XY matrix
XY = zeros(145,5);

% Populate the XY matrix with boundaries (+/-)10 away from initial
     center
% location put forth above.
for i=1:1:size(initialCenterXY,1)
    XY(i,1) = i;
     ...
    XY(i,5) = initialCenterXY(i,3)+25;

end

% The weigthed average of the optical fiber end value
```

```
cenXY = [[1:1:size(XY,1)]',zeros(size(XY,1),2)];

% Ones matrix for the whole 1944, 2592 array.
%one = pic == 1023;
% Use value less than saturation as some OFs don't get saturated.
    This
% value however needs to large enough that you don't "migrate" the
     average
% location of the OF box too far as this causes bad alignment.
one = pic >= 400;

% Loops to go through each box and figure out the weighted X,Y
    location
% for the center of the optical fiber.
for i = 1:size(XY,1)

    % Holder variables used for the average
    topX = 0;countX = 0;

    ...

    %cenXY(i,3) = 1944 - avgY;

end

% Write the cenXY file so Cprog can read it.
csvwrite('OFlocations.csv',cenXY(:,2:3));

% Plotter to visualize the OF boxes
data = pic;

% Draw boxes into the datafile so we can quickly see if they
    enclose
% the end of the optical fiber.
for i=1:size(XY,1)
    ...
end

% Make the OF average very large so they can be IDed in the plot
    as well.
for l=1:145
    x = 2592-cenXY(l,2);
    %x = cenXY(l,2);
    y = cenXY(l,3);
    %y = 1944-cenXY(l,3);
    data(y,x) = 1500;
end

% Highlight the central horizontal and vertical of the picture.
for i=ver
      ...
end

% Plot the data and boxes - must zoom in, sometimes they don't
    show up.
figgg = figure(2);hold on;
set(figgg, 'units', 'centimeters', 'pos', [1 1 40 30])
```

```
# Author:  Alex R. Mead
# Date: May 2016
# Description: Used to take Matlab optical fiber coordinate end
    points and make them
# into a header file for Cprog.

import io

def main():
  # End of optical fiber locations header file
  ##############################################################
  file = open('OFlocations.csv','rU')
  data = file.readlines()
  file.close()

  # Drop the '\n' character
  for i in range(0,len(data)):
    data[i] = data[i][:-1]

  # Create double array string
  head = "int OFs[145][2] = {"
  body = ""

  for j in range(0,len(data)-1):
    body = body + '{' + data[j]+ '},'

    headBody = head + body + '{' + data[-1]+'}' '};'

    # Write header to file
    file = open('OFlocations.h','w')
    file.write(headBody)
    file.close()

    ############################################################

    # Scaling factor for each photo for each optical fiber,
    # code will be populated once the calibration process for
    # the CUBE2.0 has been executed.
    ############################################################
    # Slopes
    file = open('TotalSlopes.csv','rU')
    data = file.readlines()
    file.close()

    ...

    for j in range(0,len(data)-1):
        body = body + '{' + data[j]+ '},'

    headBody = head + body + '{' + data[-1]+'}' '};'

    # Write header to file
    file = open('scalers.h','a')
    file.write(headBody)
    file.close()

    ############################################################

# Boiler Plate Code
if __name__ == "__main__":
        main()
```

D.15 Alignment

This set of programs is used to align the RPiCM within the CUBE 2.0 measurement cone to take proper exiting luminance measurements.

D.15.1 alignment.py

Python program used to get RPiCM measurement during alignment.

```
# Author: Alex R. Mead
# Date: May 2016
# Description: Used for alignment purposes of the RPiCM inside
    CUBE2.0

# imports
import subprocess
import sys
from time import sleep

# Constants
RPiCM_IP = "10.0.0.13"

def main():

  # Start the loop with a burner
  BURNER = raw_input("Hit 'ENTER' to loop.")

  while (BURNER != "c"):
    # Connect to RPiCM inside CUBE2.0
    connect = True
    while connect:
        try:
            subprocess.call(["./sshAlignment.sh"])
            subprocess.call(['sftp','-b','sshAlignmentGet.sh','
                pi@10.0.0.13'])
            connect = False
        except:
            # Catch the exception if the socket query fails
                because the RPiCM isn't listening.
            sleep(1)
            print("query...")

    # Call the ./Cprog to process the photo
    print("about to call Cprog....\n")
    subprocess.call(["./Cprog", 'align.data', '100000.data', '
        10000.data', '1000.data', '100.data', '10.data','nothing.
        csv'])
    print("...called Cprog\n")
```

```

        BURNER = raw_input("To cylce hit 'ENTER', to exit, hit 'c'\n")

    sys.exit(0)

if __name__ == "__main__":
        main()
```

D.15.2 sshAlignment.sh

Shell script used by alignment.py to execute measurement for RPiCM alignment purposes.

```
#!/bin/sh

    ssh -t -t pi@10.0.0.13 <<EOF
    cd Desktop/realCUBE/align
    python alignShooter.py
    exit
    EOF
```

D.15.3 alignShoot.py

Python scrip running on the RPiCM which is used by sshAlignment.sh for RPiCM alignment purposes.

```
# Author: Alex R. Mead
# Date: May 2016
# Description: Used for alignment purposes, resides on RPiCM
    inside CUBE2.0, and is called externally by the
# alignment.py script from the MBP.

import picamera
import sys
from time import sleep

# Make camera object
camera = picamera.PiCamera()
camera.framerate = 1
sleep(0.2)

# Configure the picture, name the file, take the shot.
fileName = "align.data"
```

```
camera.shutter_speed = 1000000 # Want a long exposure so we
    saturate the pixels associated with the optical fiber ends.
sleep(0.1)
camera.rotation = 180
camera.capture(fileName, format='jpeg', bayer=True)

sys.exit(0)
```

D.15.4 sshAlignmentGet.sh

Shell sript used by alignment.py to download measurement for RPiCM alignment purposes.

```
#!/bin/sh
  cd Desktop/realCUBE
  get align.data
  exit
```

D.16 Select Ptolemy II Model Files

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE entity PUBLIC "-//UC Berkeley//DTD MoML 1//EN"
    "http://ptolemy.eecs.berkeley.edu/xml/dtd/MoML_1.dtd">
<entity name="CUBE2" class="ptolemy.actor.TypedCompositeActor">
    <property name="_createdBy" class="ptolemy.kernel.attributes.
        VersionAttribute" value="11.0.devel">
    </property>
    <property name="_windowProperties" class="ptolemy.actor.gui.
        WindowPropertiesAttribute" value="{bounds={0, 23, 1440,
        873}, maximized=true}">
    </property>
    <property name="_vergilSize" class="ptolemy.actor.gui.
        SizeAttribute" value="[1206, 763]">
    </property>
    <property name="_vergilZoomFactor" class="ptolemy.data.expr.
        ExpertParameter" value="1.1513761467889911">
    </property>
    <property name="_vergilCenter" class="ptolemy.data.expr.
        ExpertParameter" value="{525.5347360557768,
        270.99999999999994}">
    </property>
    <property name="DE␣Director" class="ptolemy.domains.de.kernel.DEDirector">
        <property name="startTime" class="ptolemy.data.expr.Parameter" value="">
        </property>
        <property name="stopTime" class="ptolemy.data.expr.Parameter" value="20000">
        </property>
        <property name="synchronizeToRealTime" class="ptolemy.data.expr.Parameter" value="false">
        </property>
        <property name="_location" class="ptolemy.kernel.util.Location" value="[80.0,␣30.0]">
        </property>
    </property>
    <property name="RPiCM_measureTime" class="ptolemy.data.expr.Parameter" value="20">
        <property name="_hideName" class="ptolemy.kernel.util.SingletonAttribute">
        </property>
        <property name="_icon" class="ptolemy.vergil.icon.ValueIcon">
            <property name="_color" class="ptolemy.actor.gui.ColorAttribute" value="{0.0,␣0.0,␣1.0,␣1.0}">
            </property>
        </property>
        <property name="_smallIconDescription" class="ptolemy.kernel.util.SingletonConfigurableAttribute">
            <configure>
      <svg>
        <text x="20" style="font-size:14;␣font-family:SansSerif;␣fill:blue" y="20">-P-</text>
      </svg>
    </configure>
        </property>
        <property name="_editorFactory" class="ptolemy.vergil.toolbox.VisibleParameterEditorFactory">
        </property>
        <property name="_location" class="ptolemy.kernel.util.Location" value="[490.0,␣-50.0]">
        </property>
    </property>
    <property name="RPiLM_measureTime" class="ptolemy.data.expr.
        Parameter" value="2">
        <property name="_hideName" class="ptolemy.kernel.util.
            SingletonAttribute">
        </property>
        <property name="_icon" class="ptolemy.vergil.icon.
            ValueIcon">
            <property name="_color" class="ptolemy.actor.gui.
                ColorAttribute" value="{0.0,␣0.0,␣1.0,␣1.0}">
            </property>
        </property>
        <property name="_smallIconDescription" class="ptolemy.
            kernel.util.SingletonConfigurableAttribute">
            <configure>
      <svg>
        <text x="20" style="font-size:14;␣font-family:SansSerif;␣
            fill:blue" y="20">-P-</text>
      </svg>
    </configure>
        </property>
        <property name="_editorFactory" class="ptolemy.vergil.
            toolbox.VisibleParameterEditorFactory">
        </property>
        <property name="_location" class="ptolemy.kernel.util.
            Location" value="[490.0,␣-35.0]">
        </property>
    </property>
    <property name="ProcessTime6D" class="ptolemy.data.expr.
        Parameter" value="50">
        <property name="_hideName" class="ptolemy.kernel.util.
            SingletonAttribute">
        </property>
        <property name="_icon" class="ptolemy.vergil.icon.
            ValueIcon">
            <property name="_color" class="ptolemy.actor.gui.
                ColorAttribute" value="{0.0,␣0.0,␣1.0,␣1.0}">
            </property>
        </property>
        <property name="_smallIconDescription" class="ptolemy.
            kernel.util.SingletonConfigurableAttribute">
            <configure>
      <svg>
```

```
        <text x="20" style="font-size:14;␣font-family:SansSerif;␣fill:blue" y="20">-P-</text>
      </svg>
    </configure>
        </property>
        <property name="_editorFactory" class="ptolemy.vergil.toolbox.VisibleParameterEditorFactory">
        </property>
        <property name="_location" class="ptolemy.kernel.util.Location" value="[305.0,␣-50.0]">
        </property>
    </property>
    <property name="RadianceSimulationTime" class="ptolemy.data.expr.Parameter" value="35">
        <property name="_hideName" class="ptolemy.kernel.util.SingletonAttribute">
        </property>
        <property name="_icon" class="ptolemy.vergil.icon.ValueIcon">
            <property name="_color" class="ptolemy.actor.gui.ColorAttribute" value="{0.0,␣0.0,␣1.0,␣1.0}">
            </property>
        </property>
        <property name="_smallIconDescription" class="ptolemy.kernel.util.SingletonConfigurableAttribute">
            <configure>
      <svg>
        <text x="20" style="font-size:14;␣font-family:SansSerif;␣fill:blue" y="20">-P-</text>
      </svg>
    </configure>
        </property>
        <property name="_editorFactory" class="ptolemy.vergil.toolbox.VisibleParameterEditorFactory">
        </property>
        <property name="_location" class="ptolemy.kernel.util.Location" value="[305.0,␣-35.0]">
        </property>
    </property>
    <entity name="MacBookPro" class="ptolemy.domains.modal.modal.ModalModel">
        <property name="_tableauFactory" class="ptolemy.vergil.
            modal.modal.ModalTableauFactory">
        </property>
        <property name="_location" class="ptolemy.kernel.util.
            Location" value="[285.0,␣250.0]">
        </property>
        <port name="Cycle" class="ptolemy.domains.modal.modal.
            ModalPort">
            <property name="input"/>
            <property name="_showName" class="ptolemy.data.expr.
                SingletonParameter" value="true">
            </property>
            <property name="DecoratorAttributesFor_Bus" class="
                ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                <property name="decoratorName" class="ptolemy.
                    kernel.util.StringAttribute" value="Bus">
                </property>
                <property name="enable" class="ptolemy.data.expr.
                    Parameter" value="false">
                </property>
                <property name="sequenceNumber" class="ptolemy.
                    data.expr.Parameter" value="-1">
                </property>
                <property name="messageLength" class="ptolemy.data
                    .expr.Parameter" value="1">
                </property>
            </property>
        </port>
        <port name="RPiCM_OUT" class="ptolemy.domains.modal.modal.
            ModalPort">
            <property name="output"/>
            <property name="_showName" class="ptolemy.data.expr.
                SingletonParameter" value="true">
            </property>
        </port>
        <port name="RPiCM_IN" class="ptolemy.domains.modal.modal.
            ModalPort">
            <property name="input"/>
            <property name="_showName" class="ptolemy.data.expr.
                SingletonParameter" value="true">
            </property>
            <property name="DecoratorAttributesFor_Bus" class="
                ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
```

```
                <property name="decoratorName" class="ptolemy.kernel.util.StringAttribute" value="Bus">
                </property>
                <property name="enable" class="ptolemy.data.expr.Parameter" value="true">
                </property>
                <property name="sequenceNumber" class="ptolemy.data.expr.Parameter" value="1">
                </property>
                <property name="messageLength" class="ptolemy.data.expr.Parameter" value="1">
                </property>
            </property>
            <property name="_showInfo" class="ptolemy.kernel.util.StringAttribute" value="Aspects: Bus">
            </property>
        </port>
        <port name="CycleDone" class="ptolemy.domains.modal.modal.ModalPort">
            <property name="output"/>
            <property name="_showName" class="ptolemy.data.expr.SingletonParameter" value="true">
            </property>
        </port>
        <port name="RPiLM_OUT" class="ptolemy.domains.modal.modal.ModalPort">
            <property name="output"/>
            <property name="_showName" class="ptolemy.data.expr.SingletonParameter" value="true">
            </property>
        </port>
        <port name="RPiLM_IN" class="ptolemy.domains.modal.modal.ModalPort">
            <property name="input"/>
            <property name="_showName" class="ptolemy.data.expr.SingletonParameter" value="true">
            </property>
            <property name="DecoratorAttributesFor_Bus" class="ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                <property name="decoratorName" class="ptolemy.kernel.util.StringAttribute" value="Bus">
                </property>
                <property name="enable" class="ptolemy.data.expr.Parameter" value="true">
                </property>
                <property name="sequenceNumber" class="ptolemy.data.expr.Parameter" value="1">
                </property>
                <property name="messageLength" class="ptolemy.data.expr.Parameter" value="1">
                </property>
            </property>
            <property name="_showInfo" class="ptolemy.kernel.util.StringAttribute" value="Aspects:␣Bus">
            </property>
        </port>
        <entity name="_Controller" class="ptolemy.domains.modal.modal.ModalController">
            <property name="_library" class="ptolemy.moml.LibraryAttribute">
                <configure>
      <entity name="state␣library" class="ptolemy.kernel.CompositeEntity"><input source="ptolemy/configs/basicUtilities.xml"></input><entity name="state" class="ptolemy.domains.modal.kernel.State"><property name="_centerName" class="ptolemy.kernel.util.Attribute"></property><property name="_controllerFactory" class="ptolemy.vergil.modal.modal.HierarchicalStateControllerFactory"></property></entity></entity>
    </configure>
            </property>
            <property name="initializedRPiCM" class="ptolemy.data.expr.Parameter" value="false">
                <property name="_hideName" class="ptolemy.kernel.util.SingletonAttribute">
                </property>
                <property name="_icon" class="ptolemy.vergil.icon.ValueIcon">
                    <property name="_color" class="ptolemy.actor.gui.ColorAttribute" value="{0.0,␣0.0,␣1.0,␣1.0}">
                    </property>
                </property>
```

```
              <property name="_smallIconDescription" class="ptolemy.kernel.util.SingletonConfigurableAttribute">
                  <configure>
    <svg>
      <text x="20" style="font-size:14; font-family:SansSerif; fill:blue" y="20">-P-</text>
    </svg>
  </configure>
              </property>
              <property name="_editorFactory" class="ptolemy.vergil.toolbox.VisibleParameterEditorFactory">
              </property>
              <property name="_location" class="ptolemy.kernel.util.Location" value="[475.0, 30.0]">
              </property>
          </property>
          <property name="retrievedRPiCM" class="ptolemy.data.expr.Parameter" value="false">
              <property name="_hideName" class="ptolemy.kernel.util.SingletonAttribute">
              </property>
              <property name="_icon" class="ptolemy.vergil.icon.ValueIcon">
                  <property name="_color" class="ptolemy.actor.gui.ColorAttribute" value="{0.0, 0.0, 1.0, 1.0}">
                  </property>
              </property>
              <property name="_smallIconDescription" class="ptolemy.kernel.util.SingletonConfigurableAttribute">
                  <configure>
    <svg>
      <text x="20" style="font-size:14; font-family:SansSerif; fill:blue" y="20">-P-</text>
    </svg>
  </configure>
              </property>
              <property name="_editorFactory" class="ptolemy.vergil.toolbox.VisibleParameterEditorFactory">
              </property>
```

```
                <property name="_location" class="ptolemy.kernel.util.Location" value="[475.0, 45.0]">
                </property>
            </property>
            <property name="_windowProperties" class="ptolemy.actor.gui.WindowPropertiesAttribute" value="{bounds={0, 23, 1440, 873}, maximized=true}">
            </property>
            <property name="_vergilSize" class="ptolemy.actor.gui.SizeAttribute" value="[1686, 946]">
            </property>
            <property name="_vergilZoomFactor" class="ptolemy.data.expr.ExpertParameter" value="1.1268464045371822">
            </property>
            <property name="_vergilCenter" class="ptolemy.data.expr.ExpertParameter" value="{780.6683443509617, 438.91992187500006}">
            </property>
            <property name="initializedRPiLM" class="ptolemy.data.expr.Parameter" value="false">
                <property name="_hideName" class="ptolemy.kernel.util.SingletonAttribute">
                </property>
                <property name="_icon" class="ptolemy.vergil.icon.ValueIcon">
                    <property name="_color" class="ptolemy.actor.gui.ColorAttribute" value="{0.0, 0.0, 1.0, 1.0}">
                    </property>
                </property>
                <property name="_smallIconDescription" class="ptolemy.kernel.util.SingletonConfigurableAttribute">
                    <configure>
      <svg>
        <text x="20" style="font-size:14; font-family:SansSerif; fill:blue" y="20">-P-</text>
      </svg>
    </configure>
                </property>
                <property name="_editorFactory" class="ptolemy.vergil.toolbox.VisibleParameterEditorFactory">
```

```
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[635.0, 30.0]">
                    </property>
                </property>
                <property name="retrievedRPiLM" class="ptolemy.data.
                    expr.Parameter" value="false">
                    <property name="_hideName" class="ptolemy.kernel.
                        util.SingletonAttribute">
                    </property>
                    <property name="_icon" class="ptolemy.vergil.icon.
                        ValueIcon">
                        <property name="_color" class="ptolemy.actor.
                            gui.ColorAttribute" value="{0.0, 0.0, 1.0, 
                            1.0}">
                        </property>
                    </property>
                    <property name="_smallIconDescription" class="
                        ptolemy.kernel.util.
                        SingletonConfigurableAttribute">
                        <configure>
      <svg>
        <text x="20" style="font-size:14; font-family:SansSerif; 
            fill:blue" y="20">-P-</text>
      </svg>
    </configure>
                    </property>
                    <property name="_editorFactory" class="ptolemy.
                        vergil.toolbox.VisibleParameterEditorFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[635.0, 45.0]">
                    </property>
                </property>
                <property name="RPiCM_refreshRate" class="ptolemy.data
                    .expr.Parameter" value="0.5">
                    <property name="_hideName" class="ptolemy.kernel.
                        util.SingletonAttribute">
                    </property>
                    <property name="_icon" class="ptolemy.vergil.icon.
                        ValueIcon">
                        <property name="_color" class="ptolemy.actor.
                        gui.ColorAttribute" value="{0.0, 0.0, 1.0, 
                        1.0}">
                    </property>
                </property>
                <property name="_smallIconDescription" class="
                    ptolemy.kernel.util.
                    SingletonConfigurableAttribute">
                    <configure>
        <svg>
          <text x="20" style="font-size:14; font-family:SansSerif; 
              fill:blue" y="20">-P-</text>
        </svg>
      </configure>
                </property>
                <property name="_editorFactory" class="ptolemy.
                    vergil.toolbox.VisibleParameterEditorFactory">
                </property>
                <property name="_location" class="ptolemy.kernel.
                    util.Location" value="[810.0, 30.0]">
                </property>
            </property>
            <property name="RPiLM_refreshRate" class="ptolemy.data
                .expr.Parameter" value="0.5">
                <property name="_hideName" class="ptolemy.kernel.
                    util.SingletonAttribute">
                </property>
                <property name="_icon" class="ptolemy.vergil.icon.
                    ValueIcon">
                    <property name="_color" class="ptolemy.actor.
                        gui.ColorAttribute" value="{0.0, 0.0, 1.0, 
                        1.0}">
                    </property>
                </property>
                <property name="_smallIconDescription" class="
                    ptolemy.kernel.util.
                    SingletonConfigurableAttribute">
                    <configure>
        <svg>
          <text x="20" style="font-size:14; font-family:SansSerif; 
              fill:blue" y="20">-P-</text>
        </svg>
      </configure>
```

```
                    </property>
                    <property name="_editorFactory" class="ptolemy.
                        vergil.toolbox.VisibleParameterEditorFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[810.0, 50.0]">
                    </property>
                </property>
                <port name="Cycle" class="ptolemy.domains.modal.modal.
                    RefinementPort">
                    <property name="input"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[50.0, 380.0]">
                    </property>
                </port>
                <port name="RPiCM_OUT" class="ptolemy.domains.modal.
                    modal.RefinementPort">
                    <property name="input"/>
                    <property name="output"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[1045.0, 245.0]">
                    </property>
                </port>
                <port name="RPiCM_IN" class="ptolemy.domains.modal.
                    modal.RefinementPort">
                    <property name="input"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[50.0, 250.0]">
                    </property>
                </port>
                <port name="CycleDone" class="ptolemy.domains.modal.
                    modal.RefinementPort">
                    <property name="input"/>
                    <property name="output"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[1045.0, 380.0]">
                    </property>
                    <property name="DecoratorAttributesFor_Bus" class=
                        "ptolemy.domains.de.lib.aspect.Bus$
                        BusAttributes">
                        <property name="decoratorName" class="ptolemy.
                            kernel.util.StringAttribute" value="Bus">
                        </property>
                        <property name="enable" class="ptolemy.data.
                            expr.Parameter" value="false">
                        </property>
                        <property name="sequenceNumber" class="ptolemy
                            .data.expr.Parameter" value="-1">
                        </property>
                        <property name="messageLength" class="ptolemy.
                            data.expr.Parameter" value="1">
                        </property>
                    </property>
                </port>
                <port name="RPiLM_OUT" class="ptolemy.domains.modal.
                    modal.RefinementPort">
                    <property name="input"/>
                    <property name="output"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[1045.0, 285.0]">
                    </property>
                </port>
                <port name="RPiLM_IN" class="ptolemy.domains.modal.
                    modal.RefinementPort">
                    <property name="input"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[50.0, 295.0]">
                    </property>
                </port>
                <entity name="idle" class="ptolemy.domains.modal.
                    kernel.State">
                    <property name="isInitialState" class="ptolemy.
                        data.expr.Parameter" value="true">
                    </property>
                    <property name="_hideName" class="ptolemy.data.
                        expr.SingletonParameter" value="true">
                    </property>
                    <property name="_controllerFactory" class="ptolemy
                        .vergil.modal.modal.
                        HierarchicalStateControllerFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[340.0, 230.0]">
                    </property>
                </entity>
                <entity name="NotifyRPiCM" class="ptolemy.domains.
                    modal.kernel.State">
                    <display name="NotifyRPiCM&#10;"/>
                    <property name="refinementName" class="ptolemy.
                        kernel.util.StringAttribute" value="
                        NotifyRPiCMLM">
                    </property>
                    <property name="_hideName" class="ptolemy.data.
                        expr.SingletonParameter" value="true">
                    </property>
                    <property name="_controllerFactory" class="ptolemy
                        .vergil.modal.modal.
                        HierarchicalStateControllerFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[300.0, 380.0]">
                    </property>
                </entity>
                <entity name="6D" class="ptolemy.domains.modal.kernel.
                    State">
                    <property name="_hideName" class="ptolemy.data.
                        expr.SingletonParameter" value="true">
                    </property>
```

```
          <property name="_controllerFactory" class="ptolemy
              .vergil.modal.modal.
              HierarchicalStateControllerFactory">
          </property>
          <property name="_location" class="ptolemy.kernel.
              util.Location" value="[695.0,␣530.0]">
          </property>
      </entity>
      <entity name="RetrieveRPiCM" class="ptolemy.domains.
          modal.kernel.State">
          <property name="refinementName" class="ptolemy.
              kernel.util.StringAttribute" value="
              RetrieveRPiCMLM">
          </property>
          <property name="_hideName" class="ptolemy.data.
              expr.SingletonParameter" value="true">
          </property>
          <property name="_controllerFactory" class="ptolemy
              .vergil.modal.modal.
              HierarchicalStateControllerFactory">
          </property>
          <property name="_location" class="ptolemy.kernel.
              util.Location" value="[780.0,␣390.0]">
          </property>
      </entity>
      <entity name="Radiance" class="ptolemy.domains.modal.
          kernel.State">
          <property name="_hideName" class="ptolemy.data.
              expr.SingletonParameter" value="true">
          </property>
          <property name="_controllerFactory" class="ptolemy
              .vergil.modal.modal.
              HierarchicalStateControllerFactory">
          </property>
          <property name="_location" class="ptolemy.kernel.
              util.Location" value="[520.0,␣110.0]">
          </property>
      </entity>
      <entity name="NotifyRPiLM" class="ptolemy.domains.
          modal.kernel.State">
          <property name="refinementName" class="ptolemy.
              kernel.util.StringAttribute" value="NotifyRPiLM
```

```
                       ">
                    </property>
                    <property name="_hideName" class="ptolemy.data.
                       expr.SingletonParameter" value="true">
                    </property>
                    <property name="_controllerFactory" class="ptolemy
                       .vergil.modal.modal.
                       HierarchicalStateControllerFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                       util.Location" value="[395.0,␣535.0]">
                    </property>
                </entity>
                <entity name="RetrieveRPiLM" class="ptolemy.domains.
                   modal.kernel.State">
                    <property name="refinementName" class="ptolemy.
                       kernel.util.StringAttribute" value="
                       RetrieveRPiLM">
                    </property>
                    <property name="_hideName" class="ptolemy.data.
                       expr.SingletonParameter" value="true">
                    </property>
                    <property name="_controllerFactory" class="ptolemy
                       .vergil.modal.modal.
                       HierarchicalStateControllerFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                       util.Location" value="[755.0,␣215.0]">
                    </property>
                </entity>
                <relation name="relation" class="ptolemy.domains.modal
                   .kernel.Transition">
                    <property name="guardExpression" class="ptolemy.
                       kernel.util.StringAttribute" value="
                       Cycle_isPresent">
                    </property>
                    <property name="exitAngle" class="ptolemy.data.
                       expr.Parameter" value="0.4462441122563543">
                    </property>
                    <property name="gamma" class="ptolemy.data.expr.
                       Parameter" value="1.8174650378624067">
                    </property>
```

```
        </relation>
        <relation name="relation3" class="ptolemy.domains.
            modal.kernel.Transition">
            <property name="guardExpression" class="ptolemy.
                kernel.util.StringAttribute" value="timeout(
                ProcessTime6D)">
            </property>
            <property name="exitAngle" class="ptolemy.data.
                expr.Parameter" value="0.32531064491484435">
            </property>
            <property name="gamma" class="ptolemy.data.expr.
                Parameter" value="-1.1252165533306393">
            </property>
        </relation>
        <relation name="relation5" class="ptolemy.domains.
            modal.kernel.Transition">
            <property name="guardExpression" class="ptolemy.
                kernel.util.StringAttribute" value="timeout(
                RadianceSimulationTime)">
            </property>
            <property name="outputActions" class="ptolemy.
                domains.modal.kernel.OutputActionsAttribute"
                value="CycleDone=&quot;Done&quot;">
            </property>
            <property name="exitAngle" class="ptolemy.data.
                expr.Parameter" value="0.28834054522264757">
            </property>
            <property name="gamma" class="ptolemy.data.expr.
                Parameter" value="3.064852174861029">
            </property>
        </relation>
        <relation name="relation2" class="ptolemy.domains.
            modal.kernel.Transition">
            <property name="guardExpression" class="ptolemy.
                kernel.util.StringAttribute" value="
                initializedRPiCM==true">
            </property>
            <property name="setActions" class="ptolemy.domains
                .modal.kernel.CommitActionsAttribute" value="
                initializedRPiCM=false">
            </property>
            <property name="exitAngle" class="ptolemy.data.
```

```
                expr.Parameter" value="0.45759752500425954">
            </property>
            <property name="gamma" class="ptolemy.data.expr.
                Parameter" value="0.9621207336686424">
            </property>
            <property name="immediate" class="ptolemy.data.
                expr.Parameter" value="false">
            </property>
            <property name="preemptive" class="ptolemy.data.
                expr.Parameter" value="true">
            </property>
        </relation>
        <relation name="relation6" class="ptolemy.domains.
            modal.kernel.Transition">
            <property name="guardExpression" class="ptolemy.
                kernel.util.StringAttribute" value="
                initializedRPiLM==true">
            </property>
            <property name="setActions" class="ptolemy.domains
                .modal.kernel.CommitActionsAttribute" value="
                initializedRPiLM=false">
            </property>
            <property name="exitAngle" class="ptolemy.data.
                expr.Parameter" value="0.41395737592128434">
            </property>
            <property name="gamma" class="ptolemy.data.expr.
                Parameter" value="0.03371074730779905 6">
            </property>
            <property name="preemptive" class="ptolemy.data.
                expr.Parameter" value="true">
            </property>
        </relation>
        <relation name="relation4" class="ptolemy.domains.
            modal.kernel.Transition">
            <property name="guardExpression" class="ptolemy.
                kernel.util.StringAttribute" value="
                retrievedRPiCM==true">
            </property>
            <property name="setActions" class="ptolemy.domains
                .modal.kernel.CommitActionsAttribute" value="
                retrievedRPiCM=false">
            </property>
            </property>
            <property name="exitAngle" class="ptolemy.data.
                expr.Parameter" value="0.313503629732943">
            </property>
            <property name="gamma" class="ptolemy.data.expr.
                Parameter" value="-2.5671047394685025">
            </property>
            <property name="preemptive" class="ptolemy.data.
                expr.Parameter" value="true">
            </property>
        </relation>
        <relation name="relation7" class="ptolemy.domains.
            modal.kernel.Transition">
            <property name="guardExpression" class="ptolemy.
                kernel.util.StringAttribute" value="
                retrievedRPiLM==true">
            </property>
            <property name="setActions" class="ptolemy.domains
                .modal.kernel.CommitActionsAttribute" value="
                retrievedRPiLM=false">
            </property>
            <property name="exitAngle" class="ptolemy.data.
                expr.Parameter" value="0.6283185307179586">
            </property>
            <property name="gamma" class="ptolemy.data.expr.
                Parameter" value="-2.8448816474767624">
            </property>
            <property name="preemptive" class="ptolemy.data.
                expr.Parameter" value="true">
            </property>
        </relation>
        <link port="idle.incomingPort" relation="relation5"/>
        <link port="idle.outgoingPort" relation="relation"/>
        <link port="NotifyRPiCM.incomingPort" relation="
            relation"/>
        <link port="NotifyRPiCM.outgoingPort" relation="
            relation2"/>
        <link port="6D.incomingPort" relation="relation6"/>
        <link port="6D.outgoingPort" relation="relation3"/>
        <link port="RetrieveRPiCM.incomingPort" relation="
            relation3"/>
        <link port="RetrieveRPiCM.outgoingPort" relation="
            relation4"/>
        <link port="Radiance.incomingPort" relation="relation7"/>
        <link port="Radiance.outgoingPort" relation="relation5"/>
        <link port="NotifyRPiLM.incomingPort" relation="relation2"/>
        <link port="NotifyRPiLM.outgoingPort" relation="relation6"/>
        <link port="RetrieveRPiLM.incomingPort" relation="relation4"/>
        <link port="RetrieveRPiLM.outgoingPort" relation="relation7"/>
    </entity>
    <entity name="NotifyRPiCMLM" class="ptolemy.domains.modal.modal.ModalRefinement">
        <property name="_tableauFactory" class="ptolemy.vergil.modal.modal.ModalTableauFactory">
        </property>
        <port name="Cycle" class="ptolemy.domains.modal.modal.ModalRefinementPort">
            <property name="input"/>
            <property name="defaultValue" class="ptolemy.data.expr.Parameter" value="">
            </property>
            <property name="_location" class="ptolemy.kernel.util.Location" value="[155.0, 40.0]">
            </property>
        </port>
        <port name="RPiCM_OUT" class="ptolemy.domains.modal.modal.ModalRefinementPort">
            <property name="output"/>
            <property name="defaultValue" class="ptolemy.data.expr.Parameter" value="">
            </property>
        </port>
        <port name="RPiCM_IN" class="ptolemy.domains.modal.modal.ModalRefinementPort">
            <property name="input"/>
            <property name="defaultValue" class="ptolemy.data.expr.Parameter" value="">
            </property>
        </port>
```

```
            <port name="CycleDone" class="ptolemy.domains.modal.
               modal.ModalRefinementPort">
                <property name="output"/>
                <property name="defaultValue" class="ptolemy.data.
                   expr.Parameter" value="">
                </property>
            </port>
            <port name="RPiLM_OUT" class="ptolemy.domains.modal.
               modal.ModalRefinementPort">
                <property name="output"/>
                <property name="defaultValue" class="ptolemy.data.
                   expr.Parameter" value="">
                </property>
            </port>
            <port name="RPiLM_IN" class="ptolemy.domains.modal.
               modal.ModalRefinementPort">
                <property name="input"/>
                <property name="defaultValue" class="ptolemy.data.
                   expr.Parameter" value="">
                </property>
            </port>
            <entity name="_Controller" class="ptolemy.domains.
               modal.modal.ModalController">
                <property name="_library" class="ptolemy.moml.
                   LibraryAttribute">
                    <configure>
            <entity name="state␣library" class="ptolemy.kernel.
               CompositeEntity"><input source="ptolemy/configs/
               basicUtilities.xml"></input><entity name="state"
               class="ptolemy.domains.modal.kernel.State"><
               property name="_controllerFactory" class="ptolemy.
               vergil.modal.modal.
               HierarchicalStateControllerFactory"></property></
               entity></entity>
          </configure>
                </property>
                <property name="_windowProperties" class="ptolemy.
                   actor.gui.WindowPropertiesAttribute" value="{
                   bounds={240,␣23,␣960,␣873},␣maximized=false}">
                </property>
                <property name="_vergilSize" class="ptolemy.actor.
                   gui.SizeAttribute" value="[726,␣946]">
          </property>
          <property name="_vergilZoomFactor" class="ptolemy.
             data.expr.ExpertParameter" value="1.0">
          </property>
          <property name="_vergilCenter" class="ptolemy.data
             .expr.ExpertParameter" value="{360.0, 
             638.43359375}">
          </property>
          <port name="Cycle" class="ptolemy.domains.modal.
             modal.RefinementPort">
              <property name="input"/>
              <property name="_location" class="ptolemy.
                 kernel.util.Location" value="[35.0, 570.0]"
                 >
              </property>
          </port>
          <port name="RPiCM_OUT" class="ptolemy.domains.
             modal.modal.RefinementPort">
              <property name="input"/>
              <property name="output"/>
              <property name="_location" class="ptolemy.
                 kernel.util.Location" value="[605.0, 380.0]
                 ">
              </property>
          </port>
          <port name="RPiCM_IN" class="ptolemy.domains.modal
             .modal.RefinementPort">
              <property name="input"/>
              <property name="_location" class="ptolemy.
                 kernel.util.Location" value="[40.0, 395.0]"
                 >
              </property>
          </port>
          <port name="CycleDone" class="ptolemy.domains.
             modal.modal.RefinementPort">
              <property name="input"/>
              <property name="output"/>
              <property name="_location" class="ptolemy.
                 kernel.util.Location" value="[605.0, 505.0]
                 ">
              </property>
          </port>
```

```
                    <port name="RPiLM_OUT" class="ptolemy.domains.
                       modal.modal.RefinementPort">
                        <property name="input"/>
                        <property name="output"/>
                        <property name="_location" class="ptolemy.
                           kernel.util.Location" value="[605.0, 425.0]
                           ">
                        </property>
                    </port>
                    <port name="RPiLM_IN" class="ptolemy.domains.modal
                       .modal.RefinementPort">
                        <property name="input"/>
                        <property name="_location" class="ptolemy.
                           kernel.util.Location" value="[40.0, 445.0]"
                           >
                        </property>
                    </port>
                    <entity name="initiate" class="ptolemy.domains.
                       modal.kernel.State">
                        <property name="isInitialState" class="ptolemy
                           .data.expr.Parameter" value="true">
                        </property>
                        <property name="_hideName" class="ptolemy.data
                           .expr.SingletonParameter" value="true">
                        </property>
                        <property name="_controllerFactory" class="
                           ptolemy.vergil.modal.modal.
                           HierarchicalStateControllerFactory">
                        </property>
                        <property name="_location" class="ptolemy.
                           kernel.util.Location" value="[137.0,
                           361.87042235951213]">
                        </property>
                    </entity>
                    <entity name="wait" class="ptolemy.domains.modal.
                       kernel.State">
                        <property name="_hideName" class="ptolemy.data
                           .expr.SingletonParameter" value="true">
                        </property>
                        <property name="_controllerFactory" class="
                           ptolemy.vergil.modal.modal.
                           HierarchicalStateControllerFactory">
```

```
                </property>
                <property name="_location" class="ptolemy.
                    kernel.util.Location" value="[367.0,␣
                    361.87042235951213]">
                </property>
            </entity>
            <relation name="relation" class="ptolemy.domains.
                modal.kernel.Transition">
                <property name="guardExpression" class="
                    ptolemy.kernel.util.StringAttribute" value=
                    "true">
                </property>
                <property name="outputActions" class="ptolemy.
                    domains.modal.kernel.OutputActionsAttribute
                    " value="RPiCM_OUT=true">
                </property>
                <property name="setActions" class="ptolemy.
                    domains.modal.kernel.CommitActionsAttribute
                    " value="initializedRPiCM=false">
                </property>
                <property name="exitAngle" class="ptolemy.data
                    .expr.Parameter" value="0.6283185307179586"
                    >
                </property>
                <property name="gamma" class="ptolemy.data.
                    expr.Parameter" value="0.0">
                </property>
            </relation>
            <relation name="relation2" class="ptolemy.domains.
                modal.kernel.Transition">
                <property name="guardExpression" class="
                    ptolemy.kernel.util.StringAttribute" value=
                    "RPiCM_IN==true">
                </property>
                <property name="setActions" class="ptolemy.
                    domains.modal.kernel.CommitActionsAttribute
                    " value="initializedRPiCM=true">
                </property>
                <property name="exitAngle" class="ptolemy.data
                    .expr.Parameter" value="0.6283185307179586"
                    >
                </property>
```

```
                    <property name="gamma" class="ptolemy.data.expr.Parameter" value="-3.1050180242444787">
                    </property>
                    <property name="immediate" class="ptolemy.data.expr.Parameter" value="false">
                    </property>
                </relation>
                <link port="initiate.incomingPort" relation="relation2"/>
                <link port="initiate.outgoingPort" relation="relation"/>
                <link port="wait.incomingPort" relation="relation"/>
                <link port="wait.outgoingPort" relation="relation2"/>
            </entity>
            <relation name="CycleRelation" class="ptolemy.actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
                </property>
            </relation>
            <relation name="RPiCM_OUTRelation" class="ptolemy.actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
                </property>
            </relation>
            <relation name="RPiCM_INRelation" class="ptolemy.actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
                </property>
            </relation>
            <relation name="CycleDoneRelation" class="ptolemy.actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
                </property>
            </relation>
            <relation name="RPiLM_OUTRelation" class="ptolemy.
```

```
                actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <relation name="RPiLM_INRelation" class="ptolemy.actor
                .TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <link port="Cycle" relation="CycleRelation"/>
            <link port="RPiCM_OUT" relation="RPiCM_OUTRelation"/>
            <link port="RPiCM_IN" relation="RPiCM_INRelation"/>
            <link port="CycleDone" relation="CycleDoneRelation"/>
            <link port="RPiLM_OUT" relation="RPiLM_OUTRelation"/>
            <link port="RPiLM_IN" relation="RPiLM_INRelation"/>
            <link port="_Controller.Cycle" relation="CycleRelation
                "/>
            <link port="_Controller.RPiCM_OUT" relation="
                RPiCM_OUTRelation"/>
            <link port="_Controller.RPiCM_IN" relation="
                RPiCM_INRelation"/>
            <link port="_Controller.CycleDone" relation="
                CycleDoneRelation"/>
            <link port="_Controller.RPiLM_OUT" relation="
                RPiLM_OUTRelation"/>
            <link port="_Controller.RPiLM_IN" relation="
                RPiLM_INRelation"/>
        </entity>
        <entity name="RetrieveRPiCMLM" class="ptolemy.domains.
            modal.modal.ModalRefinement">
            <property name="_tableauFactory" class="ptolemy.vergil
                .modal.modal.ModalTableauFactory">
            </property>
            <port name="Cycle" class="ptolemy.domains.modal.modal.
                ModalRefinementPort">
                <property name="input"/>
                <property name="defaultValue" class="ptolemy.data.
                    expr.Parameter" value="">
                </property>
                <property name="_location" class="ptolemy.kernel.
```

```
               util.Location" value="[155.0,␣40.0]">
          </property>
      </port>
      <port name="RPiCM_OUT" class="ptolemy.domains.modal.
          modal.ModalRefinementPort">
          <property name="output"/>
          <property name="defaultValue" class="ptolemy.data.
              expr.Parameter" value="">
          </property>
          <property name="_location" class="ptolemy.kernel.
              util.Location" value="[475.0,␣775.0]">
          </property>
      </port>
      <port name="RPiCM_IN" class="ptolemy.domains.modal.
          modal.ModalRefinementPort">
          <property name="input"/>
          <property name="defaultValue" class="ptolemy.data.
              expr.Parameter" value="">
          </property>
          <property name="_location" class="ptolemy.kernel.
              util.Location" value="[155.0,␣80.0]">
          </property>
      </port>
      <port name="CycleDone" class="ptolemy.domains.modal.
          modal.ModalRefinementPort">
          <property name="output"/>
          <property name="defaultValue" class="ptolemy.data.
              expr.Parameter" value="">
          </property>
      </port>
      <port name="RPiLM_OUT" class="ptolemy.domains.modal.
          modal.ModalRefinementPort">
          <property name="output"/>
          <property name="defaultValue" class="ptolemy.data.
              expr.Parameter" value="">
          </property>
      </port>
      <port name="RPiLM_IN" class="ptolemy.domains.modal.
          modal.ModalRefinementPort">
          <property name="input"/>
          <property name="defaultValue" class="ptolemy.data.
              expr.Parameter" value="">
```

```
                  </property>
              </port>
              <entity name="_Controller" class="ptolemy.domains.modal.modal.ModalController">
                  <property name="_library" class="ptolemy.moml.LibraryAttribute">
                      <configure>
              <entity name="state␣library" class="ptolemy.kernel.CompositeEntity"><input source="ptolemy/configs/basicUtilities.xml"></input><entity name="state" class="ptolemy.domains.modal.kernel.State"><property name="_controllerFactory" class="ptolemy.vergil.modal.modal.HierarchicalStateControllerFactory"></property></entity></entity>
            </configure>
                  </property>
                  <property name="_windowProperties" class="ptolemy.actor.gui.WindowPropertiesAttribute" value="{bounds={3360,␣23,␣960,␣1057},␣maximized=false}">
                  </property>
                  <property name="_vergilSize" class="ptolemy.actor.gui.SizeAttribute" value="[726,␣946]">
                  </property>
                  <property name="_vergilZoomFactor" class="ptolemy.data.expr.ExpertParameter" value="1.0">
                  </property>
                  <property name="_vergilCenter" class="ptolemy.data.expr.ExpertParameter" value="{360.0,␣638.43359375}">
                  </property>
                  <port name="Cycle" class="ptolemy.domains.modal.modal.RefinementPort">
                      <property name="input"/>
                      <property name="_location" class="ptolemy.kernel.util.Location" value="{20.0,␣200.0}">
                      </property>
                  </port>
                  <port name="RPiCM_OUT" class="ptolemy.domains.modal.modal.RefinementPort">
```

```
                <property name="input"/>
                <property name="output"/>
                <property name="_location" class="ptolemy.
                    kernel.util.Location" value="[580.0, 235.0]
                    ">
                </property>
            </port>
            <port name="RPiCM_IN" class="ptolemy.domains.modal
                .modal.RefinementPort">
                <property name="input"/>
                <property name="_location" class="ptolemy.
                    kernel.util.Location" value="{20.0, 240.0}"
                    >
                </property>
            </port>
            <port name="CycleDone" class="ptolemy.domains.
                modal.modal.RefinementPort">
                <property name="input"/>
                <property name="output"/>
                <property name="_location" class="ptolemy.
                    kernel.util.Location" value="[590.0, 405.0]
                    ">
                </property>
            </port>
            <port name="RPiLM_OUT" class="ptolemy.domains.
                modal.modal.RefinementPort">
                <property name="input"/>
                <property name="output"/>
                <property name="_location" class="ptolemy.
                    kernel.util.Location" value="[575.0, 295.0]
                    ">
                </property>
                <property name="DecoratorAttributesFor_Bus"
                    class="ptolemy.domains.de.lib.aspect.Bus$
                    BusAttributes">
                    <property name="decoratorName" class="
                        ptolemy.kernel.util.StringAttribute"
                        value="Bus">
                    </property>
                    <property name="enable" class="ptolemy.
                        data.expr.Parameter" value="false">
                    </property>
                    <property name="sequenceNumber" class="
                        ptolemy.data.expr.Parameter" value="-1"
                        >
                    </property>
                    <property name="messageLength" class="
                        ptolemy.data.expr.Parameter" value="1">
                    </property>
                </property>
            </port>
            <port name="RPiLM_IN" class="ptolemy.domains.modal
                .modal.RefinementPort">
                <property name="input"/>
                <property name="_location" class="ptolemy.
                    kernel.util.Location" value="[20.0, 285.0]"
                    >
                </property>
            </port>
            <entity name="retrieve" class="ptolemy.domains.
                modal.kernel.State">
                <property name="isInitialState" class="ptolemy
                    .data.expr.Parameter" value="true">
                </property>
                <property name="_hideName" class="ptolemy.data
                    .expr.SingletonParameter" value="true">
                </property>
                <property name="_controllerFactory" class="
                    ptolemy.vergil.modal.modal.
                    HierarchicalStateControllerFactory">
                </property>
                <property name="_location" class="ptolemy.
                    kernel.util.Location" value="[145.0, 525.0]
                    ">
                </property>
            </entity>
            <entity name="wait" class="ptolemy.domains.modal.
                kernel.State">
                <property name="_hideName" class="ptolemy.data
                    .expr.SingletonParameter" value="true">
                </property>
                <property name="_controllerFactory" class="
                    ptolemy.vergil.modal.modal.
                    HierarchicalStateControllerFactory">
                </property>
                <property name="_location" class="ptolemy.kernel.util.Location" value="[385.0, 520.0]">
                </property>
            </entity>
            <relation name="relation" class="ptolemy.domains.modal.kernel.Transition">
                <property name="guardExpression" class="ptolemy.kernel.util.StringAttribute" value="true">
                </property>
                <property name="outputActions" class="ptolemy.domains.modal.kernel.OutputActionsAttribute" value="RPiCM_OUT=true">
                </property>
                <property name="setActions" class="ptolemy.domains.modal.kernel.CommitActionsAttribute" value="retrievedRPiCM=false">
                </property>
                <property name="exitAngle" class="ptolemy.data.expr.Parameter" value="0.6283185307179586">
                </property>
                <property name="gamma" class="ptolemy.data.expr.Parameter" value="0.0">
                </property>
            </relation>
            <relation name="relation2" class="ptolemy.domains.modal.kernel.Transition">
                <property name="guardExpression" class="ptolemy.kernel.util.StringAttribute" value="RPiCM_IN==true">
                </property>
                <property name="setActions" class="ptolemy.domains.modal.kernel.CommitActionsAttribute" value="retrievedRPiCM=true">
                </property>
                <property name="exitAngle" class="ptolemy.data.expr.Parameter" value="0.6283185307179586">
                </property>
```

```
                    <property name="gamma" class="ptolemy.data.
                        expr.Parameter" value="-3.1304825078331295"
                        >
                    </property>
                </relation>
                <relation name="relation3" class="ptolemy.domains.
                    modal.kernel.Transition">
                    <property name="guardExpression" class="
                        ptolemy.kernel.util.StringAttribute" value=
                        "timeout(RPiCM_refreshRate)">
                    </property>
                    <property name="outputActions" class="ptolemy.
                        domains.modal.kernel.OutputActionsAttribute
                        " value="RPiCM_OUT=true">
                    </property>
                    <property name="exitAngle" class="ptolemy.data
                        .expr.Parameter" value="2.692100088512782">
                    </property>
                    <property name="gamma" class="ptolemy.data.
                        expr.Parameter" value="-1.5540005524232863"
                        >
                    </property>
                    <property name="defaultTransition" class="
                        ptolemy.data.expr.Parameter" value="false">
                         <display name="default"/>
                    </property>
                </relation>
                <link port="retrieve.incomingPort" relation="
                    relation2"/>
                <link port="retrieve.outgoingPort" relation="
                    relation"/>
                <link port="wait.incomingPort" relation="relation"
                    />
                <link port="wait.incomingPort" relation="relation3
                    "/>
                <link port="wait.outgoingPort" relation="relation2
                    "/>
                <link port="wait.outgoingPort" relation="relation3
                    "/>
            </entity>
            <relation name="CycleRelation" class="ptolemy.actor.
                TypedIORelation">
```

```
                    <property name="width" class="ptolemy.data.expr.
                        Parameter" value="Auto">
                    </property>
                </relation>
                <relation name="RPiCM_OUTRelation" class="ptolemy.
                    actor.TypedIORelation">
                    <property name="width" class="ptolemy.data.expr.
                        Parameter" value="Auto">
                    </property>
                </relation>
                <relation name="RPiCM_INRelation" class="ptolemy.actor
                    .TypedIORelation">
                    <property name="width" class="ptolemy.data.expr.
                        Parameter" value="Auto">
                    </property>
                </relation>
                <relation name="CycleDoneRelation" class="ptolemy.
                    actor.TypedIORelation">
                    <property name="width" class="ptolemy.data.expr.
                        Parameter" value="Auto">
                    </property>
                </relation>
                <relation name="RPiLM_OUTRelation" class="ptolemy.
                    actor.TypedIORelation">
                    <property name="width" class="ptolemy.data.expr.
                        Parameter" value="Auto">
                    </property>
                </relation>
                <relation name="RPiLM_INRelation" class="ptolemy.actor
                    .TypedIORelation">
                    <property name="width" class="ptolemy.data.expr.
                        Parameter" value="Auto">
                    </property>
                </relation>
                <link port="Cycle" relation="CycleRelation"/>
                <link port="RPiCM_OUT" relation="RPiCM_OUTRelation"/>
                <link port="RPiCM_IN" relation="RPiCM_INRelation"/>
                <link port="CycleDone" relation="CycleDoneRelation"/>
                <link port="RPiLM_OUT" relation="RPiLM_OUTRelation"/>
                <link port="RPiLM_IN" relation="RPiLM_INRelation"/>
                <link port="_Controller.Cycle" relation="CycleRelation
                    "/>
```

```
            <link port="_Controller.RPiCM_OUT" relation="
                RPiCM_OUTRelation"/>
            <link port="_Controller.RPiCM_IN" relation="
                RPiCM_INRelation"/>
            <link port="_Controller.CycleDone" relation="
                CycleDoneRelation"/>
            <link port="_Controller.RPiLM_OUT" relation="
                RPiLM_OUTRelation"/>
            <link port="_Controller.RPiLM_IN" relation="
                RPiLM_INRelation"/>
        </entity>
        <entity name="NotifyRPiLM" class="ptolemy.domains.modal.
            modal.ModalRefinement">
            <property name="_tableauFactory" class="ptolemy.vergil
                .modal.modal.ModalTableauFactory">
            </property>
            <port name="Cycle" class="ptolemy.domains.modal.modal.
                ModalRefinementPort">
                <property name="input"/>
                <property name="defaultValue" class="ptolemy.data.
                    expr.Parameter" value="">
                </property>
                <property name="_location" class="ptolemy.kernel.
                    util.Location" value="[155.0, 40.0]">
                </property>
            </port>
            <port name="RPiCM_OUT" class="ptolemy.domains.modal.
                modal.ModalRefinementPort">
                <property name="output"/>
                <property name="defaultValue" class="ptolemy.data.
                    expr.Parameter" value="">
                </property>
                <property name="_location" class="ptolemy.kernel.
                    util.Location" value="[475.0, 775.0]">
                </property>
            </port>
            <port name="RPiCM_IN" class="ptolemy.domains.modal.
                modal.ModalRefinementPort">
                <property name="input"/>
                <property name="defaultValue" class="ptolemy.data.
                    expr.Parameter" value="">
                </property>
```

```
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[155.0, 80.0]">
                    </property>
                </port>
                <port name="CycleDone" class="ptolemy.domains.modal.
                    modal.ModalRefinementPort">
                    <property name="output"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[590.0, 770.0]">
                    </property>
                </port>
                <port name="RPiLM_OUT" class="ptolemy.domains.modal.
                    modal.ModalRefinementPort">
                    <property name="output"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                </port>
                <port name="RPiLM_IN" class="ptolemy.domains.modal.
                    modal.ModalRefinementPort">
                    <property name="input"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                </port>
                <entity name="_Controller" class="ptolemy.domains.
                    modal.modal.ModalController">
                    <property name="_library" class="ptolemy.moml.
                        LibraryAttribute">
                        <configure>
                <entity name="state library" class="ptolemy.kernel.
                    CompositeEntity"><input source="ptolemy/configs/
                    basicUtilities.xml"></input><entity name="state"
                    class="ptolemy.domains.modal.kernel.State"><
                    property name="_controllerFactory" class="ptolemy.
                    vergil.modal.modal.
                    HierarchicalStateControllerFactory"></property></
                    entity></entity>
            </configure>
```

```
                    </property>
                    <property name="_windowProperties" class="ptolemy.
                       actor.gui.WindowPropertiesAttribute" value="{
                       bounds={1930, 107, 960, 873}, maximized=false}"
                       >
                    </property>
                    <property name="_vergilSize" class="ptolemy.actor.
                       gui.SizeAttribute" value="[726, 762]">
                    </property>
                    <property name="_vergilZoomFactor" class="ptolemy.
                       data.expr.ExpertParameter" value="1.25">
                    </property>
                    <property name="_vergilCenter" class="ptolemy.data
                       .expr.ExpertParameter" value="{360.0, 
                       546.4335937499999}">
                    </property>
                    <port name="Cycle" class="ptolemy.domains.modal.
                       modal.RefinementPort">
                        <property name="input"/>
                        <property name="_location" class="ptolemy.
                           kernel.util.Location" value="[35.0, 610.0]"
                           >
                        </property>
                    </port>
                    <port name="RPiCM_OUT" class="ptolemy.domains.
                       modal.modal.RefinementPort">
                        <property name="input"/>
                        <property name="output"/>
                        <property name="_location" class="ptolemy.
                           kernel.util.Location" value="[535.0, 400.0]
                           ">
                        </property>
                    </port>
                    <port name="RPiCM_IN" class="ptolemy.domains.modal
                       .modal.RefinementPort">
                        <property name="input"/>
                        <property name="_location" class="ptolemy.
                           kernel.util.Location" value="[20.0, 480.0]"
                           >
                        </property>
                    </port>
                    <port name="CycleDone" class="ptolemy.domains.
```

```
                      modal.modal.RefinementPort">
                        <property name="input"/>
                        <property name="output"/>
                        <property name="_location" class="ptolemy.
                            kernel.util.Location" value="[540.0, 540.0]
                            ">
                        </property>
                    </port>
                    <port name="RPiLM_OUT" class="ptolemy.domains.
                      modal.modal.RefinementPort">
                        <property name="input"/>
                        <property name="output"/>
                        <property name="_location" class="ptolemy.
                            kernel.util.Location" value="[535.0, 440.0]
                            ">
                        </property>
                    </port>
                    <port name="RPiLM_IN" class="ptolemy.domains.modal
                      .modal.RefinementPort">
                        <property name="input"/>
                        <property name="_location" class="ptolemy.
                            kernel.util.Location" value="[20.0, 525.0]"
                            >
                        </property>
                    </port>
                    <entity name="initiate" class="ptolemy.domains.
                      modal.kernel.State">
                        <property name="isInitialState" class="ptolemy
                            .data.expr.Parameter" value="true">
                        </property>
                        <property name="_hideName" class="ptolemy.data
                            .expr.SingletonParameter" value="true">
                        </property>
                        <property name="_controllerFactory" class="
                            ptolemy.vergil.modal.modal.
                            HierarchicalStateControllerFactory">
                        </property>
                        <property name="_location" class="ptolemy.
                            kernel.util.Location" value="[145.0, 530.0]
                            ">
                        </property>
                    </entity>
```

```
            <entity name="wait" class="ptolemy.domains.modal.
               kernel.State">
                <property name="_hideName" class="ptolemy.data
                   .expr.SingletonParameter" value="true">
                </property>
                <property name="_controllerFactory" class="
                   ptolemy.vergil.modal.modal.
                   HierarchicalStateControllerFactory">
                </property>
                <property name="_location" class="ptolemy.
                   kernel.util.Location" value="[375.0, 530.0]
                   ">
                </property>
            </entity>
            <relation name="relation" class="ptolemy.domains.
               modal.kernel.Transition">
                <property name="guardExpression" class="
                   ptolemy.kernel.util.StringAttribute" value=
                   "true">
                </property>
                <property name="outputActions" class="ptolemy.
                   domains.modal.kernel.OutputActionsAttribute
                   " value="RPiLM_OUT=true">
                </property>
                <property name="setActions" class="ptolemy.
                   domains.modal.kernel.CommitActionsAttribute
                   " value="initializedRPiLM=false">
                </property>
                <property name="exitAngle" class="ptolemy.data
                   .expr.Parameter" value="0.6283185307179586"
                   >
                </property>
                <property name="gamma" class="ptolemy.data.
                   expr.Parameter" value="0.0">
                </property>
            </relation>
            <relation name="relation2" class="ptolemy.domains.
               modal.kernel.Transition">
                <property name="guardExpression" class="
                   ptolemy.kernel.util.StringAttribute" value=
                   "RPiLM_IN==true">
                </property>
```

```
                    <property name="setActions" class="ptolemy.domains.modal.kernel.CommitActionsAttribute" value="initializedRPiLM=true">
                    </property>
                    <property name="exitAngle" class="ptolemy.data.expr.Parameter" value="0.6283185307179586">
                    </property>
                    <property name="gamma" class="ptolemy.data.expr.Parameter" value="-3.1050180242444787">
                    </property>
                    <property name="immediate" class="ptolemy.data.expr.Parameter" value="false">
                    </property>
                </relation>
                <link port="initiate.incomingPort" relation="relation2"/>
                <link port="initiate.outgoingPort" relation="relation"/>
                <link port="wait.incomingPort" relation="relation"/>
                <link port="wait.outgoingPort" relation="relation2"/>
            </entity>
            <relation name="CycleRelation" class="ptolemy.actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
                </property>
            </relation>
            <relation name="RPiCM_OUTRelation" class="ptolemy.actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
                </property>
            </relation>
            <relation name="RPiCM_INRelation" class="ptolemy.actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
                </property>
            </relation>
            <relation name="CycleDoneRelation" class="ptolemy.
                actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <relation name="RPiLM_OUTRelation" class="ptolemy.
                actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <relation name="RPiLM_INRelation" class="ptolemy.actor
                .TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <link port="Cycle" relation="CycleRelation"/>
            <link port="RPiCM_OUT" relation="RPiCM_OUTRelation"/>
            <link port="RPiCM_IN" relation="RPiCM_INRelation"/>
            <link port="CycleDone" relation="CycleDoneRelation"/>
            <link port="RPiLM_OUT" relation="RPiLM_OUTRelation"/>
            <link port="RPiLM_IN" relation="RPiLM_INRelation"/>
            <link port="_Controller.Cycle" relation="CycleRelation
                "/>
            <link port="_Controller.RPiCM_OUT" relation="
                RPiCM_OUTRelation"/>
            <link port="_Controller.RPiCM_IN" relation="
                RPiCM_INRelation"/>
            <link port="_Controller.CycleDone" relation="
                CycleDoneRelation"/>
            <link port="_Controller.RPiLM_OUT" relation="
                RPiLM_OUTRelation"/>
            <link port="_Controller.RPiLM_IN" relation="
                RPiLM_INRelation"/>
        </entity>
        <entity name="RetrieveRPiLM" class="ptolemy.domains.modal.
            modal.ModalRefinement">
            <property name="_tableauFactory" class="ptolemy.vergil
                .modal.modal.ModalTableauFactory">
            </property>
            <port name="Cycle" class="ptolemy.domains.modal.modal.
                ModalRefinementPort">
                <property name="input"/>
                <property name="defaultValue" class="ptolemy.data.
                    expr.Parameter" value="">
                </property>
                <property name="_location" class="ptolemy.kernel.
                    util.Location" value="[50.0, 380.0]">
                </property>
            </port>
            <port name="RPiCM_OUT" class="ptolemy.domains.modal.
                modal.ModalRefinementPort">
                <property name="output"/>
                <property name="defaultValue" class="ptolemy.data.
                    expr.Parameter" value="">
                </property>
                <property name="_location" class="ptolemy.kernel.
                    util.Location" value="[1045.0, 245.0]">
                </property>
            </port>
            <port name="RPiCM_IN" class="ptolemy.domains.modal.
                modal.ModalRefinementPort">
                <property name="input"/>
                <property name="defaultValue" class="ptolemy.data.
                    expr.Parameter" value="">
                </property>
                <property name="_location" class="ptolemy.kernel.
                    util.Location" value="[50.0, 250.0]">
                </property>
            </port>
            <port name="CycleDone" class="ptolemy.domains.modal.
                modal.ModalRefinementPort">
                <property name="output"/>
                <property name="defaultValue" class="ptolemy.data.
                    expr.Parameter" value="">
                </property>
                <property name="_location" class="ptolemy.kernel.
                    util.Location" value="[1045.0, 380.0]">
                </property>
            </port>
            <port name="RPiLM_OUT" class="ptolemy.domains.modal.
```

```
                modal.ModalRefinementPort">
                <property name="output"/>
                <property name="defaultValue" class="ptolemy.data.
                    expr.Parameter" value="">
                </property>
                <property name="_location" class="ptolemy.kernel.
                    util.Location" value="[1045.0, 285.0]">
                </property>
            </port>
            <port name="RPiLM_IN" class="ptolemy.domains.modal.
                modal.ModalRefinementPort">
                <property name="input"/>
                <property name="defaultValue" class="ptolemy.data.
                    expr.Parameter" value="">
                </property>
                <property name="_location" class="ptolemy.kernel.
                    util.Location" value="[50.0, 295.0]">
                </property>
            </port>
            <entity name="_Controller" class="ptolemy.domains.
                modal.modal.ModalController">
                <property name="_library" class="ptolemy.moml.
                    LibraryAttribute">
                    <configure>
            <entity name="state_library" class="ptolemy.kernel.
                CompositeEntity"><input source="ptolemy/configs/
                basicUtilities.xml"></input><entity name="state"
                class="ptolemy.domains.modal.kernel.State"><
                property name="_controllerFactory" class="ptolemy.
                vergil.modal.modal.
                HierarchicalStateControllerFactory"></property></
                entity></entity>
        </configure>
                </property>
                <property name="_windowProperties" class="ptolemy.
                    actor.gui.WindowPropertiesAttribute" value="{
                    bounds={4320, 23, 960, 1057}, maximized=false}"
                    >
                </property>
                <property name="_vergilSize" class="ptolemy.actor.
                    gui.SizeAttribute" value="[726, 946]">
                </property>
```

```
                    <property name="_vergilZoomFactor" class="ptolemy.
                       data.expr.ExpertParameter" value="
                       0.9685997470118667">
                    </property>
                    <property name="_vergilCenter" class="ptolemy.data
                       .expr.ExpertParameter" value="
                       {302.68457031250006, 561.9709649965084}">
                    </property>
                    <port name="Cycle" class="ptolemy.domains.modal.
                       modal.RefinementPort">
                        <property name="input"/>
                        <property name="_location" class="ptolemy.
                           kernel.util.Location" value="{20.0, 200.0}"
                           >
                        </property>
                    </port>
                    <port name="RPiCM_OUT" class="ptolemy.domains.
                       modal.modal.RefinementPort">
                        <property name="input"/>
                        <property name="output"/>
                        <property name="_location" class="ptolemy.
                           kernel.util.Location" value="[545.0, 280.0]
                           ">
                        </property>
                    </port>
                    <port name="RPiCM_IN" class="ptolemy.domains.modal
                       .modal.RefinementPort">
                        <property name="input"/>
                        <property name="_location" class="ptolemy.
                           kernel.util.Location" value="{20.0, 240.0}"
                           >
                        </property>
                    </port>
                    <port name="CycleDone" class="ptolemy.domains.
                       modal.modal.RefinementPort">
                        <property name="input"/>
                        <property name="output"/>
                        <property name="_location" class="ptolemy.
                           kernel.util.Location" value="[545.0, 315.0]
                           ">
                        </property>
                    </port>
```

```
            <port name="RPiLM_OUT" class="ptolemy.domains.
               modal.modal.RefinementPort">
                <property name="input"/>
                <property name="output"/>
                <property name="_location" class="ptolemy.
                    kernel.util.Location" value="[540.0,␣365.0]
                    ">
                </property>
            </port>
            <port name="RPiLM_IN" class="ptolemy.domains.modal
               .modal.RefinementPort">
                <property name="input"/>
                <property name="_location" class="ptolemy.
                    kernel.util.Location" value="{20.0,␣280.0}"
                    >
                </property>
            </port>
            <entity name="retrieve" class="ptolemy.domains.
               modal.kernel.State">
                <property name="isInitialState" class="ptolemy
                    .data.expr.Parameter" value="true">
                </property>
                <property name="_hideName" class="ptolemy.data
                    .expr.SingletonParameter" value="true">
                </property>
                <property name="_controllerFactory" class="
                    ptolemy.vergil.modal.modal.
                    HierarchicalStateControllerFactory">
                </property>
                <property name="_location" class="ptolemy.
                    kernel.util.Location" value="[119.5,␣
                    393.10351562500006]">
                </property>
            </entity>
            <entity name="wait" class="ptolemy.domains.modal.
               kernel.State">
                <property name="_hideName" class="ptolemy.data
                    .expr.SingletonParameter" value="true">
                </property>
                <property name="_controllerFactory" class="
                    ptolemy.vergil.modal.modal.
                    HierarchicalStateControllerFactory">
                </property>
                <property name="_location" class="ptolemy.
                    kernel.util.Location" value="[334.5,␣
                    393.10351562500006]">
                </property>
            </entity>
            <relation name="relation" class="ptolemy.domains.
               modal.kernel.Transition">
                <property name="guardExpression" class="
                    ptolemy.kernel.util.StringAttribute" value=
                    "true">
                </property>
                <property name="outputActions" class="ptolemy.
                    domains.modal.kernel.OutputActionsAttribute
                    " value="RPiLM_OUT=true">
                </property>
                <property name="setActions" class="ptolemy.
                    domains.modal.kernel.CommitActionsAttribute
                    " value="retrievedRPiLM=false">
                </property>
                <property name="exitAngle" class="ptolemy.data
                    .expr.Parameter" value="0.6283185307179586"
                    >
                </property>
                <property name="gamma" class="ptolemy.data.
                    expr.Parameter" value="0.0">
                </property>
            </relation>
            <relation name="relation2" class="ptolemy.domains.
               modal.kernel.Transition">
                <property name="guardExpression" class="
                    ptolemy.kernel.util.StringAttribute" value=
                    "RPiLM_IN==true">
                </property>
                <property name="setActions" class="ptolemy.
                    domains.modal.kernel.CommitActionsAttribute
                    " value="retrievedRPiLM=true">
                </property>
                <property name="exitAngle" class="ptolemy.data
                    .expr.Parameter" value="0.6283185307179586"
                    >
                </property>
                <property name="gamma" class="ptolemy.data.
                   expr.Parameter" value="3.141592653589793">
                </property>
            </relation>
            <relation name="relation3" class="ptolemy.domains.
               modal.kernel.Transition">
                <property name="guardExpression" class="
                   ptolemy.kernel.util.StringAttribute" value=
                   "timeout(RPiLM_refreshRate)">
                </property>
                <property name="outputActions" class="ptolemy.
                   domains.modal.kernel.OutputActionsAttribute
                   " value="RPiLM_OUT=true">
                </property>
                <property name="exitAngle" class="ptolemy.data
                   .expr.Parameter" value="2.6764962611560787"
                   >
                </property>
                <property name="gamma" class="ptolemy.data.
                   expr.Parameter" value="-1.6057655076229849"
                   >
                </property>
            </relation>
            <link port="retrieve.incomingPort" relation="
               relation2"/>
            <link port="retrieve.outgoingPort" relation="
               relation"/>
            <link port="wait.incomingPort" relation="relation"
               />
            <link port="wait.incomingPort" relation="relation3
               "/>
            <link port="wait.outgoingPort" relation="relation2
               "/>
            <link port="wait.outgoingPort" relation="relation3
               "/>
        </entity>
        <relation name="CycleRelation" class="ptolemy.actor.
           TypedIORelation">
            <property name="width" class="ptolemy.data.expr.
               Parameter" value="Auto">
            </property>
        </relation>
```

```
            <relation name="RPiCM_OUTRelation" class="ptolemy.
                actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <relation name="RPiCM_INRelation" class="ptolemy.actor
                .TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <relation name="CycleDoneRelation" class="ptolemy.
                actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <relation name="RPiLM_OUTRelation" class="ptolemy.
                actor.TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <relation name="RPiLM_INRelation" class="ptolemy.actor
                .TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <link port="Cycle" relation="CycleRelation"/>
            <link port="RPiCM_OUT" relation="RPiCM_OUTRelation"/>
            <link port="RPiCM_IN" relation="RPiCM_INRelation"/>
            <link port="CycleDone" relation="CycleDoneRelation"/>
            <link port="RPiLM_OUT" relation="RPiLM_OUTRelation"/>
            <link port="RPiLM_IN" relation="RPiLM_INRelation"/>
            <link port="_Controller.Cycle" relation="CycleRelation
                "/>
            <link port="_Controller.RPiCM_OUT" relation="
                RPiCM_OUTRelation"/>
            <link port="_Controller.RPiCM_IN" relation="
                RPiCM_INRelation"/>
            <link port="_Controller.CycleDone" relation="CycleDoneRelation"/>
            <link port="_Controller.RPiLM_OUT" relation="RPiLM_OUTRelation"/>
            <link port="_Controller.RPiLM_IN" relation="RPiLM_INRelation"/>
        </entity>
        <relation name="CycleRelation" class="ptolemy.actor.TypedIORelation">
            <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
            </property>
        </relation>
        <relation name="RPiCM_OUTRelation" class="ptolemy.actor.TypedIORelation">
            <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
            </property>
        </relation>
        <relation name="RPiCM_INRelation" class="ptolemy.actor.TypedIORelation">
            <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
            </property>
        </relation>
        <relation name="CycleDoneRelation" class="ptolemy.actor.TypedIORelation">
            <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
            </property>
        </relation>
        <relation name="RPiLM_OUTRelation" class="ptolemy.actor.TypedIORelation">
            <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
            </property>
        </relation>
        <relation name="RPiLM_INRelation" class="ptolemy.actor.TypedIORelation">
            <property name="width" class="ptolemy.data.expr.Parameter" value="Auto">
            </property>
        </relation>
        <link port=”Cycle” relation=”CycleRelation”/>
        <link port=”RPiCM_OUT” relation=”RPiCM_OUTRelation”/>
        <link port=”RPiCM_IN” relation=”RPiCM_INRelation”/>
        <link port=”CycleDone” relation=”CycleDoneRelation”/>
        <link port=”RPiLM_OUT” relation=”RPiLM_OUTRelation”/>
        <link port=”RPiLM_IN” relation=”RPiLM_INRelation”/>
        <link port=”_Controller.Cycle” relation=”CycleRelation”/>
        <link port=”_Controller.RPiCM_OUT” relation=”
           RPiCM_OUTRelation”/>
        <link port=”_Controller.RPiCM_IN” relation=”
           RPiCM_INRelation”/>
        <link port=”_Controller.CycleDone” relation=”
           CycleDoneRelation”/>
        <link port=”_Controller.RPiLM_OUT” relation=”
           RPiLM_OUTRelation”/>
        <link port=”_Controller.RPiLM_IN” relation=”
           RPiLM_INRelation”/>
        <link port=”NotifyRPiCMLM.Cycle” relation=”CycleRelation”/
           >
        <link port=”NotifyRPiCMLM.RPiCM_OUT” relation=”
           RPiCM_OUTRelation”/>
        <link port=”NotifyRPiCMLM.RPiCM_IN” relation=”
           RPiCM_INRelation”/>
        <link port=”NotifyRPiCMLM.CycleDone” relation=”
           CycleDoneRelation”/>
        <link port=”NotifyRPiCMLM.RPiLM_OUT” relation=”
           RPiLM_OUTRelation”/>
        <link port=”NotifyRPiCMLM.RPiLM_IN” relation=”
           RPiLM_INRelation”/>
        <link port=”RetrieveRPiCMLM.Cycle” relation=”CycleRelation
           ”/>
        <link port=”RetrieveRPiCMLM.RPiCM_OUT” relation=”
           RPiCM_OUTRelation”/>
        <link port=”RetrieveRPiCMLM.RPiCM_IN” relation=”
           RPiCM_INRelation”/>
        <link port=”RetrieveRPiCMLM.CycleDone” relation=”
           CycleDoneRelation”/>
        <link port=”RetrieveRPiCMLM.RPiLM_OUT” relation=”
           RPiLM_OUTRelation”/>
        <link port=”RetrieveRPiCMLM.RPiLM_IN” relation=”
           RPiLM_INRelation”/>
        <link port="NotifyRPiLM.Cycle" relation="CycleRelation"/>
        <link port="NotifyRPiLM.RPiCM_OUT" relation="
            RPiCM_OUTRelation"/>
        <link port="NotifyRPiLM.RPiCM_IN" relation="
            RPiCM_INRelation"/>
        <link port="NotifyRPiLM.CycleDone" relation="
            CycleDoneRelation"/>
        <link port="NotifyRPiLM.RPiLM_OUT" relation="
            RPiLM_OUTRelation"/>
        <link port="NotifyRPiLM.RPiLM_IN" relation="
            RPiLM_INRelation"/>
        <link port="RetrieveRPiLM.Cycle" relation="CycleRelation"/
            >
        <link port="RetrieveRPiLM.RPiCM_OUT" relation="
            RPiCM_OUTRelation"/>
        <link port="RetrieveRPiLM.RPiCM_IN" relation="
            RPiCM_INRelation"/>
        <link port="RetrieveRPiLM.CycleDone" relation="
            CycleDoneRelation"/>
        <link port="RetrieveRPiLM.RPiLM_OUT" relation="
            RPiLM_OUTRelation"/>
        <link port="RetrieveRPiLM.RPiLM_IN" relation="
            RPiLM_INRelation"/>
    </entity>
    <entity name="DiscreteClock" class="ptolemy.actor.lib.
        DiscreteClock">
        <property name="period" class="ptolemy.actor.parameters.
            PortParameter" value="900.0">
        </property>
        <property name="values" class="ptolemy.data.expr.Parameter
            " value="{true}">
        </property>
        <doc>Create periodic timed events.</doc>
        <property name="_location" class="ptolemy.kernel.util.
            Location" value="{60.0,␣105.0}">
        </property>
        <port name="trigger" class="ptolemy.actor.TypedIOPort">
            <property name="input"/>
            <property name="multiport"/>
            <property name="DecoratorAttributesFor_Bus" class="
                ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                <property name="decoratorName" class="ptolemy.
```

```
                        kernel.util.StringAttribute" value="Bus">
                    </property>
                    <property name="enable" class="ptolemy.data.expr.
                        Parameter" value="false">
                    </property>
                    <property name="sequenceNumber" class="ptolemy.
                        data.expr.Parameter" value="-1">
                    </property>
                    <property name="messageLength" class="ptolemy.data
                        .expr.Parameter" value="1">
                    </property>
                </property>
            </port>
            <port name="period" class="ptolemy.actor.parameters.
                ParameterPort">
                <property name="input"/>
                <property name="DecoratorAttributesFor_Bus" class="
                    ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                    <property name="decoratorName" class="ptolemy.
                        kernel.util.StringAttribute" value="Bus">
                    </property>
                    <property name="enable" class="ptolemy.data.expr.
                        Parameter" value="false">
                    </property>
                    <property name="sequenceNumber" class="ptolemy.
                        data.expr.Parameter" value="-1">
                    </property>
                    <property name="messageLength" class="ptolemy.data
                        .expr.Parameter" value="1">
                    </property>
                </property>
            </port>
            <port name="start" class="ptolemy.actor.TypedIOPort">
                <property name="input"/>
                <property name="DecoratorAttributesFor_Bus" class="
                    ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                    <property name="decoratorName" class="ptolemy.
                        kernel.util.StringAttribute" value="Bus">
                    </property>
                    <property name="enable" class="ptolemy.data.expr.
                        Parameter" value="false">
                    </property>
                    <property name="sequenceNumber" class="ptolemy.
                        data.expr.Parameter" value="-1">
                    </property>
                    <property name="messageLength" class="ptolemy.data
                        .expr.Parameter" value="1">
                    </property>
                </property>
            </port>
            <port name="stop" class="ptolemy.actor.TypedIOPort">
                <property name="input"/>
                <property name="DecoratorAttributesFor_Bus" class="
                    ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                    <property name="decoratorName" class="ptolemy.
                        kernel.util.StringAttribute" value="Bus">
                    </property>
                    <property name="enable" class="ptolemy.data.expr.
                        Parameter" value="false">
                    </property>
                    <property name="sequenceNumber" class="ptolemy.
                        data.expr.Parameter" value="-1">
                    </property>
                    <property name="messageLength" class="ptolemy.data
                        .expr.Parameter" value="1">
                    </property>
                </property>
            </port>
        </entity>
        <entity name="RPiCM" class="ptolemy.domains.modal.modal.
            ModalModel">
            <property name="_tableauFactory" class="ptolemy.vergil.
                modal.modal.ModalTableauFactory">
            </property>
            <property name="_location" class="ptolemy.kernel.util.
                Location" value="[820.0, 240.0]">
            </property>
            <port name="MBP_IN" class="ptolemy.domains.modal.modal.
                ModalPort">
                <property name="input"/>
                <property name="_showName" class="ptolemy.data.expr.
                    SingletonParameter" value="true">
                </property>
                <property name="DecoratorAttributesFor_Bus" class="
```

```
                    ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                    <property name="decoratorName" class="ptolemy.
                        kernel.util.StringAttribute" value="Bus">
                    </property>
                    <property name="enable" class="ptolemy.data.expr.
                        Parameter" value="true">
                    </property>
                    <property name="sequenceNumber" class="ptolemy.
                        data.expr.Parameter" value="1">
                    </property>
                    <property name="messageLength" class="ptolemy.data
                        .expr.Parameter" value="1">
                    </property>
                </property>
                <property name="_showInfo" class="ptolemy.kernel.util.
                    StringAttribute" value="Aspects:_Bus">
                </property>
            </port>
            <port name="MBP_OUT" class="ptolemy.domains.modal.modal.
                ModalPort">
                <property name="output"/>
                <property name="_showName" class="ptolemy.data.expr.
                    SingletonParameter" value="true">
                </property>
            </port>
            <entity name="_Controller" class="ptolemy.domains.modal.
                modal.ModalController">
                <property name="_library" class="ptolemy.moml.
                    LibraryAttribute">
                    <configure>
          <entity name="state_library" class="ptolemy.kernel.
              CompositeEntity"><input source="ptolemy/configs/
              basicUtilities.xml"></input><entity name="state" class="
              ptolemy.domains.modal.kernel.State"><property name="
              _centerName" class="ptolemy.kernel.util.Attribute"></
              property><property name="_controllerFactory" class="
              ptolemy.vergil.modal.modal.
              HierarchicalStateControllerFactory"></property></entity><
              /entity>
        </configure>
                </property>
                <property name="_windowProperties" class="ptolemy.
                    actor.gui.WindowPropertiesAttribute" value="{bounds
                    ={240, 23, 960, 873}, maximized=false}">
                </property>
                <property name="_vergilSize" class="ptolemy.actor.gui.
                    SizeAttribute" value="[726, 946]">
                </property>
                <property name="_vergilZoomFactor" class="ptolemy.data
                    .expr.ExpertParameter" value="0.8">
                </property>
                <property name="_vergilCenter" class="ptolemy.data.
                    expr.ExpertParameter" value="{360.0, 477.03515625}"
                    >
                </property>
                <port name="MBP_IN" class="ptolemy.domains.modal.modal
                    .RefinementPort">
                    <property name="input"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="{20.0, 200.0}">
                    </property>
                </port>
                <port name="MBP_OUT" class="ptolemy.domains.modal.
                    modal.RefinementPort">
                    <property name="input"/>
                    <property name="output"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[605.0, 195.0]">
                    </property>
                </port>
                <entity name="waitToMeasure" class="ptolemy.domains.
                    modal.kernel.State">
                    <property name="isInitialState" class="ptolemy.
                        data.expr.Parameter" value="true">
                    </property>
                    <property name="_hideName" class="ptolemy.data.
                        expr.SingletonParameter" value="true">
                    </property>
                    <property name="_controllerFactory" class="ptolemy
                        .vergil.modal.modal.
                        HierarchicalStateControllerFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[75.0,␣265.0]">
                    </property>
                </entity>
                <entity name="initialized" class="ptolemy.domains.
                    modal.kernel.State">
                    <property name="_hideName" class="ptolemy.data.
                        expr.SingletonParameter" value="true">
                    </property>
                    <property name="_controllerFactory" class="ptolemy
                        .vergil.modal.modal.
                        HierarchicalStateControllerFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[220.0,␣350.0]">
                    </property>
                </entity>
                <entity name="processMeasurement" class="ptolemy.
                    domains.modal.kernel.State">
                    <property name="_hideName" class="ptolemy.data.
                        expr.SingletonParameter" value="true">
                    </property>
                    <property name="_controllerFactory" class="ptolemy
                        .vergil.modal.modal.
                        HierarchicalStateControllerFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[375.0,␣265.0]">
                    </property>
                </entity>
                <entity name="waitToSend" class="ptolemy.domains.modal
                    .kernel.State">
                    <property name="_hideName" class="ptolemy.data.
                        expr.SingletonParameter" value="true">
                    </property>
                    <property name="_controllerFactory" class="ptolemy
                        .vergil.modal.modal.
                        HierarchicalStateControllerFactory">
```

```
          </property>
          <property name="_location" class="ptolemy.kernel.
              util.Location" value="[350.0, 130.0]">
          </property>
      </entity>
      <entity name="sendingMeasurement" class="ptolemy.
          domains.modal.kernel.State">
          <property name="_hideName" class="ptolemy.data.
              expr.SingletonParameter" value="true">
          </property>
          <property name="_controllerFactory" class="ptolemy
              .vergil.modal.modal.
              HierarchicalStateControllerFactory">
          </property>
          <property name="_location" class="ptolemy.kernel.
              util.Location" value="[165.0, 85.0]">
          </property>
      </entity>
      <relation name="relation" class="ptolemy.domains.modal
          .kernel.Transition">
          <property name="guardExpression" class="ptolemy.
              kernel.util.StringAttribute" value="MBP_IN==
              true">
          </property>
          <property name="exitAngle" class="ptolemy.data.
              expr.Parameter" value="0.6283185307179586">
          </property>
          <property name="gamma" class="ptolemy.data.expr.
              Parameter" value="0.615545092401963">
          </property>
      </relation>
      <relation name="relation2" class="ptolemy.domains.
          modal.kernel.Transition">
          <property name="guardExpression" class="ptolemy.
              kernel.util.StringAttribute" value="true">
          </property>
          <property name="outputActions" class="ptolemy.
              domains.modal.kernel.OutputActionsAttribute"
              value="MBP_OUT=true">
          </property>
          <property name="exitAngle" class="ptolemy.data.
              expr.Parameter" value="0.6283185307179586">
```

```
            </property>
            <property name="gamma" class="ptolemy.data.expr.
                Parameter" value="-0.7493455110079961">
            </property>
            <property name="immediate" class="ptolemy.data.
                expr.Parameter" value="false">
            </property>
        </relation>
        <relation name="relation3" class="ptolemy.domains.
            modal.kernel.Transition">
            <property name="guardExpression" class="ptolemy.
                kernel.util.StringAttribute" value="timeout(
                RPiCM_measureTime)">
            </property>
            <property name="exitAngle" class="ptolemy.data.
                expr.Parameter" value="0.6283185307179586">
            </property>
            <property name="gamma" class="ptolemy.data.expr.
                Parameter" value="-2.495415464049102">
            </property>
        </relation>
        <relation name="relation4" class="ptolemy.domains.
            modal.kernel.Transition">
            <property name="guardExpression" class="ptolemy.
                kernel.util.StringAttribute" value="MBP_IN==
                true">
            </property>
            <property name="exitAngle" class="ptolemy.data.
                expr.Parameter" value="0.6283185307179586">
            </property>
            <property name="gamma" class="ptolemy.data.expr.
                Parameter" value="-2.4839244446789226">
            </property>
        </relation>
        <relation name="relation5" class="ptolemy.domains.
            modal.kernel.Transition">
            <property name="guardExpression" class="ptolemy.
                kernel.util.StringAttribute" value="true">
            </property>
            <property name="outputActions" class="ptolemy.
                domains.modal.kernel.OutputActionsAttribute"
                value="MBP_OUT=true">
            </property>
            <property name="exitAngle" class="ptolemy.data.
                expr.Parameter" value="0.6283185307179586">
            </property>
            <property name="gamma" class="ptolemy.data.expr.
                Parameter" value="2.171099330207916">
            </property>
            <property name="immediate" class="ptolemy.data.
                expr.Parameter" value="true">
            </property>
        </relation>
        <link port="waitToMeasure.incomingPort" relation="
            relation5"/>
        <link port="waitToMeasure.outgoingPort" relation="
            relation"/>
        <link port="initialized.incomingPort" relation="
            relation"/>
        <link port="initialized.outgoingPort" relation="
            relation2"/>
        <link port="processMeasurement.incomingPort" relation=
            "relation2"/>
        <link port="processMeasurement.outgoingPort" relation=
            "relation3"/>
        <link port="waitToSend.incomingPort" relation="
            relation3"/>
        <link port="waitToSend.outgoingPort" relation="
            relation4"/>
        <link port="sendingMeasurement.incomingPort" relation=
            "relation4"/>
        <link port="sendingMeasurement.outgoingPort" relation=
            "relation5"/>
    </entity>
    <relation name="MBP_INRelation" class="ptolemy.actor.
        TypedIORelation">
        <property name="width" class="ptolemy.data.expr.
            Parameter" value="Auto">
        </property>
    </relation>
    <relation name="MBP_OUTRelation" class="ptolemy.actor.
        TypedIORelation">
        <property name="width" class="ptolemy.data.expr.
            Parameter" value="Auto">
        </property>
    </relation>
    <link port="MBP_IN" relation="MBP_INRelation"/>
    <link port="MBP_OUT" relation="MBP_OUTRelation"/>
    <link port="_Controller.MBP_IN" relation="MBP_INRelation"/>
    <link port="_Controller.MBP_OUT" relation="MBP_OUTRelation"/>
</entity>
<entity name="cycles" class="ptolemy.actor.lib.gui.TimedDisplay">
    <property name="_windowProperties" class="ptolemy.actor.gui.WindowPropertiesAttribute" value="{bounds={344, 46, 499, 208}, maximized=false}">
    </property>
    <property name="_paneSize" class="ptolemy.actor.gui.SizeAttribute" value="[499, 164]">
    </property>
    <property name="_location" class="ptolemy.kernel.util.Location" value="[435.0, 110.0]">
    </property>
    <port name="input" class="ptolemy.actor.TypedIOPort">
        <property name="input"/>
        <property name="multiport"/>
        <property name="DecoratorAttributesFor_Bus" class="ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
            <property name="decoratorName" class="ptolemy.kernel.util.StringAttribute" value="Bus">
            </property>
            <property name="enable" class="ptolemy.data.expr.Parameter" value="false">
            </property>
            <property name="sequenceNumber" class="ptolemy.data.expr.Parameter" value="-1">
            </property>
            <property name="messageLength" class="ptolemy.data.expr.Parameter" value="1">
            </property>
        </property>
    </port>
</entity>
<entity name="TO-RPiCM" class="ptolemy.actor.lib.gui.
```

```
        TimedDisplay">
         <property name="_windowProperties" class="ptolemy.actor.
             gui.WindowPropertiesAttribute" value="{bounds={581,␣
             395,␣499,␣208},␣maximized=false}">
         </property>
         <property name="_paneSize" class="ptolemy.actor.gui.
             SizeAttribute" value="[499,␣164]">
         </property>
         <property name="_location" class="ptolemy.kernel.util.
             Location" value="[460.00860595703125,␣
             159.45136260986328]">
         </property>
         <port name="input" class="ptolemy.actor.TypedIOPort">
             <property name="input"/>
             <property name="multiport"/>
             <property name="DecoratorAttributesFor_Bus" class="
                 ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                 <property name="decoratorName" class="ptolemy.
                     kernel.util.StringAttribute" value="Bus">
                 </property>
                 <property name="enable" class="ptolemy.data.expr.
                     Parameter" value="false">
                 </property>
                 <property name="sequenceNumber" class="ptolemy.
                     data.expr.Parameter" value="-1">
                 </property>
                 <property name="messageLength" class="ptolemy.data
                     .expr.Parameter" value="1">
                 </property>
             </property>
         </port>
     </entity>
     <entity name="TO-MBP" class="ptolemy.actor.lib.gui.
        TimedDisplay">
         <property name="_windowProperties" class="ptolemy.actor.
             gui.WindowPropertiesAttribute" value="{bounds={880,␣75,
             ␣499,␣208},␣maximized=false}">
         </property>
         <property name="_paneSize" class="ptolemy.actor.gui.
             SizeAttribute" value="[499,␣164]">
         </property>
         <property name="_location" class="ptolemy.kernel.util.
```

```
            Location" value="[820.0, 580.0]">
        </property>
        <property name="_flipPortsHorizontal" class="ptolemy.data.
            expr.Parameter" value="true">
        </property>
        <port name="input" class="ptolemy.actor.TypedIOPort">
            <property name="input"/>
            <property name="multiport"/>
            <property name="DecoratorAttributesFor_Bus" class="
                ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                <property name="decoratorName" class="ptolemy.
                    kernel.util.StringAttribute" value="Bus">
                </property>
                <property name="enable" class="ptolemy.data.expr.
                    Parameter" value="false">
                </property>
                <property name="sequenceNumber" class="ptolemy.
                    data.expr.Parameter" value="-1">
                </property>
                <property name="messageLength" class="ptolemy.data
                    .expr.Parameter" value="1">
                </property>
            </property>
        </port>
    </entity>
    <entity name="Bus" class="ptolemy.domains.de.lib.aspect.Bus">
        <property name="serviceTimeMultiplicationFactor" class="
            ptolemy.data.expr.Parameter" value="0.1">
        </property>
        <property name="_location" class="ptolemy.kernel.util.
            Location" value="[710.0, 35.0]">
        </property>
    </entity>
    <entity name="RPiLM" class="ptolemy.domains.modal.modal.
        ModalModel">
        <display name="LJ_LiCor"/>
        <property name="_tableauFactory" class="ptolemy.vergil.
            modal.modal.ModalTableauFactory">
        </property>
        <property name="_location" class="ptolemy.kernel.util.
            Location" value="[750.0, 295.0]">
        </property>
```

```
          <port name="MBP_IN" class="ptolemy.domains.modal.modal.ModalPort">
              <property name="input"/>
              <property name="_showName" class="ptolemy.data.expr.SingletonParameter" value="true">
              </property>
              <property name="DecoratorAttributesFor_Bus" class="ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                  <property name="decoratorName" class="ptolemy.kernel.util.StringAttribute" value="Bus">
                  </property>
                  <property name="enable" class="ptolemy.data.expr.Parameter" value="true">
                  </property>
                  <property name="sequenceNumber" class="ptolemy.data.expr.Parameter" value="1">
                  </property>
                  <property name="messageLength" class="ptolemy.data.expr.Parameter" value="1">
                  </property>
              </property>
              <property name="_showInfo" class="ptolemy.kernel.util.StringAttribute" value="Aspects:␣Bus">
              </property>
          </port>
          <port name="MBP_OUT" class="ptolemy.domains.modal.modal.ModalPort">
              <property name="output"/>
              <property name="_showName" class="ptolemy.data.expr.SingletonParameter" value="true">
              </property>
          </port>
          <entity name="_Controller" class="ptolemy.domains.modal.modal.ModalController">
              <property name="_library" class="ptolemy.moml.LibraryAttribute">
                 <configure>
        <entity name="state␣library" class="ptolemy.kernel.CompositeEntity"><input source="ptolemy/configs/basicUtilities.xml"></input><entity name="state" class="ptolemy.domains.modal.kernel.State"><property name="_centerName" class="ptolemy.kernel.util.Attribute"></
```

```
            property><property name="_controllerFactory" class="
            ptolemy.vergil.modal.modal.
            HierarchicalStateControllerFactory"></property></entity><
            /entity>
        </configure>
                </property>
                <property name="_windowProperties" class="ptolemy.
                    actor.gui.WindowPropertiesAttribute" value="{bounds
                    ={240, 23, 960, 873}, maximized=false}">
                </property>
                <property name="_vergilSize" class="ptolemy.actor.gui.
                    SizeAttribute" value="[726, 762]">
                </property>
                <property name="_vergilZoomFactor" class="ptolemy.data
                    .expr.ExpertParameter" value="1.048699506336833">
                </property>
                <property name="_vergilCenter" class="ptolemy.data.
                    expr.ExpertParameter" value="{304.9939441680908, 
                    387.5393124798822}">
                </property>
                <port name="MBP_IN" class="ptolemy.domains.modal.modal
                    .RefinementPort">
                    <property name="input"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="{20.0, 200.0}">
                    </property>
                </port>
                <port name="MBP_OUT" class="ptolemy.domains.modal.
                    modal.RefinementPort">
                    <property name="input"/>
                    <property name="output"/>
                    <property name="defaultValue" class="ptolemy.data.
                        expr.Parameter" value="">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[605.0, 195.0]">
                    </property>
                </port>
                <entity name="waitToMeasure" class="ptolemy.domains.
                    modal.kernel.State">
                    <property name="isInitialState" class="ptolemy.
                        data.expr.Parameter" value="true">
                    </property>
                    <property name="_hideName" class="ptolemy.data.
                        expr.SingletonParameter" value="true">
                    </property>
                    <property name="_controllerFactory" class="ptolemy
                        .vergil.modal.modal.
                        HierarchicalStateControllerFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[75.0,␣265.0]">
                    </property>
                </entity>
                <entity name="initialized" class="ptolemy.domains.
                    modal.kernel.State">
                    <property name="_hideName" class="ptolemy.data.
                        expr.SingletonParameter" value="true">
                    </property>
                    <property name="_controllerFactory" class="ptolemy
                        .vergil.modal.modal.
                        HierarchicalStateControllerFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[220.0,␣350.0]">
                    </property>
                </entity>
                <entity name="processMeasurement" class="ptolemy.
                    domains.modal.kernel.State">
                    <property name="_hideName" class="ptolemy.data.
                        expr.SingletonParameter" value="true">
                    </property>
                    <property name="_controllerFactory" class="ptolemy
                        .vergil.modal.modal.
                        HierarchicalStateControllerFactory">
                    </property>
                    <property name="_location" class="ptolemy.kernel.
                        util.Location" value="[375.0,␣265.0]">
                    </property>
                </entity>
                <entity name="waitToSend" class="ptolemy.domains.modal
```

```
                .kernel.State">
                <property name="_hideName" class="ptolemy.data.
                    expr.SingletonParameter" value="true">
                </property>
                <property name="_controllerFactory" class="ptolemy
                    .vergil.modal.modal.
                    HierarchicalStateControllerFactory">
                </property>
                <property name="_location" class="ptolemy.kernel.
                    util.Location" value="[350.0, 130.0]">
                </property>
            </entity>
            <entity name="sendingMeasurement" class="ptolemy.
                domains.modal.kernel.State">
                <property name="_hideName" class="ptolemy.data.
                    expr.SingletonParameter" value="true">
                </property>
                <property name="_controllerFactory" class="ptolemy
                    .vergil.modal.modal.
                    HierarchicalStateControllerFactory">
                </property>
                <property name="_location" class="ptolemy.kernel.
                    util.Location" value="[165.0, 85.0]">
                </property>
            </entity>
            <relation name="relation" class="ptolemy.domains.modal
                .kernel.Transition">
                <property name="guardExpression" class="ptolemy.
                    kernel.util.StringAttribute" value="MBP_IN==
                    true">
                </property>
                <property name="exitAngle" class="ptolemy.data.
                    expr.Parameter" value="0.6283185307179586">
                </property>
                <property name="gamma" class="ptolemy.data.expr.
                    Parameter" value="0.615545092401963">
                </property>
            </relation>
            <relation name="relation2" class="ptolemy.domains.
                modal.kernel.Transition">
                <property name="guardExpression" class="ptolemy.
                    kernel.util.StringAttribute" value="true">
                </property>
                <property name="outputActions" class="ptolemy.
                    domains.modal.kernel.OutputActionsAttribute"
                    value="MBP_OUT=true">
                </property>
                <property name="exitAngle" class="ptolemy.data.
                    expr.Parameter" value="0.6283185307179586">
                </property>
                <property name="gamma" class="ptolemy.data.expr.
                    Parameter" value="-0.7493455110079961">
                </property>
                <property name="immediate" class="ptolemy.data.
                    expr.Parameter" value="false">
                </property>
            </relation>
            <relation name="relation3" class="ptolemy.domains.
                modal.kernel.Transition">
                <property name="guardExpression" class="ptolemy.
                    kernel.util.StringAttribute" value="timeout(
                    RPiLM_measureTime)">
                </property>
                <property name="exitAngle" class="ptolemy.data.
                    expr.Parameter" value="0.6283185307179586">
                </property>
                <property name="gamma" class="ptolemy.data.expr.
                    Parameter" value="-2.495415464049102">
                </property>
            </relation>
            <relation name="relation4" class="ptolemy.domains.
                modal.kernel.Transition">
                <property name="guardExpression" class="ptolemy.
                    kernel.util.StringAttribute" value="MBP_IN==
                    true">
                </property>
                <property name="exitAngle" class="ptolemy.data.
                    expr.Parameter" value="0.6283185307179586">
                </property>
                <property name="gamma" class="ptolemy.data.expr.
                    Parameter" value="-2.4839244446789226">
                </property>
            </relation>
            <relation name="relation5" class="ptolemy.domains.
```

```
                    modal.kernel.Transition">
                     <property name="guardExpression" class="ptolemy.
                         kernel.util.StringAttribute" value="true">
                     </property>
                     <property name="outputActions" class="ptolemy.
                         domains.modal.kernel.OutputActionsAttribute"
                         value="MBP_OUT=true">
                     </property>
                     <property name="exitAngle" class="ptolemy.data.
                         expr.Parameter" value="0.6283185307179586">
                     </property>
                     <property name="gamma" class="ptolemy.data.expr.
                         Parameter" value="2.171099330207916">
                     </property>
                     <property name="immediate" class="ptolemy.data.
                         expr.Parameter" value="true">
                     </property>
                </relation>
                <link port="waitToMeasure.incomingPort" relation="
                    relation5"/>
                <link port="waitToMeasure.outgoingPort" relation="
                    relation"/>
                <link port="initialized.incomingPort" relation="
                    relation"/>
                <link port="initialized.outgoingPort" relation="
                    relation2"/>
                <link port="processMeasurement.incomingPort" relation=
                    "relation2"/>
                <link port="processMeasurement.outgoingPort" relation=
                    "relation3"/>
                <link port="waitToSend.incomingPort" relation="
                    relation3"/>
                <link port="waitToSend.outgoingPort" relation="
                    relation4"/>
                <link port="sendingMeasurement.incomingPort" relation=
                    "relation4"/>
                <link port="sendingMeasurement.outgoingPort" relation=
                    "relation5"/>
            </entity>
            <relation name="MBP_INRelation" class="ptolemy.actor.
                TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <relation name="MBP_OUTRelation" class="ptolemy.actor.
                TypedIORelation">
                <property name="width" class="ptolemy.data.expr.
                    Parameter" value="Auto">
                </property>
            </relation>
            <link port="MBP_IN" relation="MBP_INRelation"/>
            <link port="MBP_OUT" relation="MBP_OUTRelation"/>
            <link port="_Controller.MBP_IN" relation="MBP_INRelation"/
                >
            <link port="_Controller.MBP_OUT" relation="MBP_OUTRelation
                "/>
        </entity>
        <entity name="TO-RPiLM" class="ptolemy.actor.lib.gui.
            TimedDisplay">
            <property name="_windowProperties" class="ptolemy.actor.
                gui.WindowPropertiesAttribute" value="{bounds={279,
                586, 499, 208}, maximized=false}">
            </property>
            <property name="_paneSize" class="ptolemy.actor.gui.
                SizeAttribute" value="[499, 164]">
            </property>
            <property name="_location" class="ptolemy.kernel.util.
                Location" value="[460.0, 325.0]">
            </property>
            <port name="input" class="ptolemy.actor.TypedIOPort">
                <property name="input"/>
                <property name="multiport"/>
                <property name="DecoratorAttributesFor_Bus" class="
                    ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                     <property name="decoratorName" class="ptolemy.
                         kernel.util.StringAttribute" value="Bus">
                     </property>
                     <property name="enable" class="ptolemy.data.expr.
                         Parameter" value="false">
                     </property>
                     <property name="sequenceNumber" class="ptolemy.
                         data.expr.Parameter" value="-1">
                     </property>
                     <property name="messageLength" class="ptolemy.data
                         .expr.Parameter" value="1">
                     </property>
                </property>
            </port>
        </entity>
        <entity name="TO-MBP_LM" class="ptolemy.actor.lib.gui.
            TimedDisplay">
            <property name="_windowProperties" class="ptolemy.actor.
                gui.WindowPropertiesAttribute" value="{bounds={38, 348,
                 499, 208}, maximized=false}">
            </property>
            <property name="_paneSize" class="ptolemy.actor.gui.
                SizeAttribute" value="[499, 164]">
            </property>
            <property name="_location" class="ptolemy.kernel.util.
                Location" value="[770.0, 490.0]">
            </property>
            <property name="_flipPortsHorizontal" class="ptolemy.data.
                expr.Parameter" value="true">
            </property>
            <port name="input" class="ptolemy.actor.TypedIOPort">
                <property name="input"/>
                <property name="multiport"/>
                <property name="DecoratorAttributesFor_Bus" class="
                    ptolemy.domains.de.lib.aspect.Bus$BusAttributes">
                     <property name="decoratorName" class="ptolemy.
                         kernel.util.StringAttribute" value="Bus">
                     </property>
                     <property name="enable" class="ptolemy.data.expr.
                         Parameter" value="false">
                     </property>
                     <property name="sequenceNumber" class="ptolemy.
                         data.expr.Parameter" value="-1">
                     </property>
                     <property name="messageLength" class="ptolemy.data
                         .expr.Parameter" value="1">
                     </property>
                </property>
            </port>
        </entity>
        <relation name="relation3" class="ptolemy.actor.
```

```
            TypedIORelation">
            <vertex name="vertex1" value="[905.0, 415.0]">
            </vertex>
        </relation>
        <relation name="relation5" class="ptolemy.actor.
            TypedIORelation">
            <vertex name="vertex1" value="{190.0, 105.0}">
            </vertex>
        </relation>
        <relation name="relation6" class="ptolemy.actor.
            TypedIORelation">
            <property name="width" class="ptolemy.data.expr.Parameter"
                value="-1">
            </property>
            <vertex name="vertex1" value="{400.0, 240.0}">
            </vertex>
        </relation>
        <relation name="relation" class="ptolemy.actor.TypedIORelation
            ">
        </relation>
        <relation name="relation2" class="ptolemy.actor.
            TypedIORelation">
            <vertex name="vertex1" value="[90.0, 415.0]">
            </vertex>
        </relation>
        <relation name="relation7" class="ptolemy.actor.
            TypedIORelation">
            <vertex name="vertex1" value="[850.0, 385.0]">
            </vertex>
        </relation>
        <relation name="relation8" class="ptolemy.actor.
            TypedIORelation">
            <vertex name="vertex1" value="{120.0, 385.0}">
            </vertex>
        </relation>
        <relation name="relation9" class="ptolemy.actor.
            TypedIORelation">
            <property name="width" class="ptolemy.data.expr.Parameter"
                value="-1">
            </property>
            <vertex name="vertex1" value="[400.0, 260.0]">
            </vertex>
```

```
    </relation>
    <link port="MacBookPro.Cycle" relation="relation5"/>
    <link port="MacBookPro.RPiCM_OUT" relation="relation6"/>
    <link port="MacBookPro.RPiCM_IN" relation="relation2"/>
    <link port="MacBookPro.CycleDone" relation="relation"/>
    <link port="MacBookPro.RPiLM_OUT" relation="relation9"/>
    <link port="MacBookPro.RPiLM_IN" relation="relation8"/>
    <link port="DiscreteClock.output" relation="relation5"/>
    <link port="RPiCM.MBP_IN" relation="relation6"/>
    <link port="RPiCM.MBP_OUT" relation="relation3"/>
    <link port="cycles.input" relation="relation5"/>
    <link port="cycles.input" relation="relation"/>
    <link port="TO-RPiCM.input" relation="relation6"/>
    <link port="TO-MBP.input" relation="relation3"/>
    <link port="RPiLM.MBP_IN" relation="relation9"/>
    <link port="RPiLM.MBP_OUT" relation="relation7"/>
    <link port="TO-RPiLM.input" relation="relation9"/>
    <link port="TO-MBP_LM.input" relation="relation7"/>
    <link relation1="relation3" relation2="relation2"/>
    <link relation1="relation7" relation2="relation8"/>
</entity>
```

Appendix E

Construction Details of the CUBE 2.0 System

E.1 8020 Frame Materials

E.1.1 1.50”×1.50” T-Slotted Profile - Four Open T-Slots (Part Number 1515)

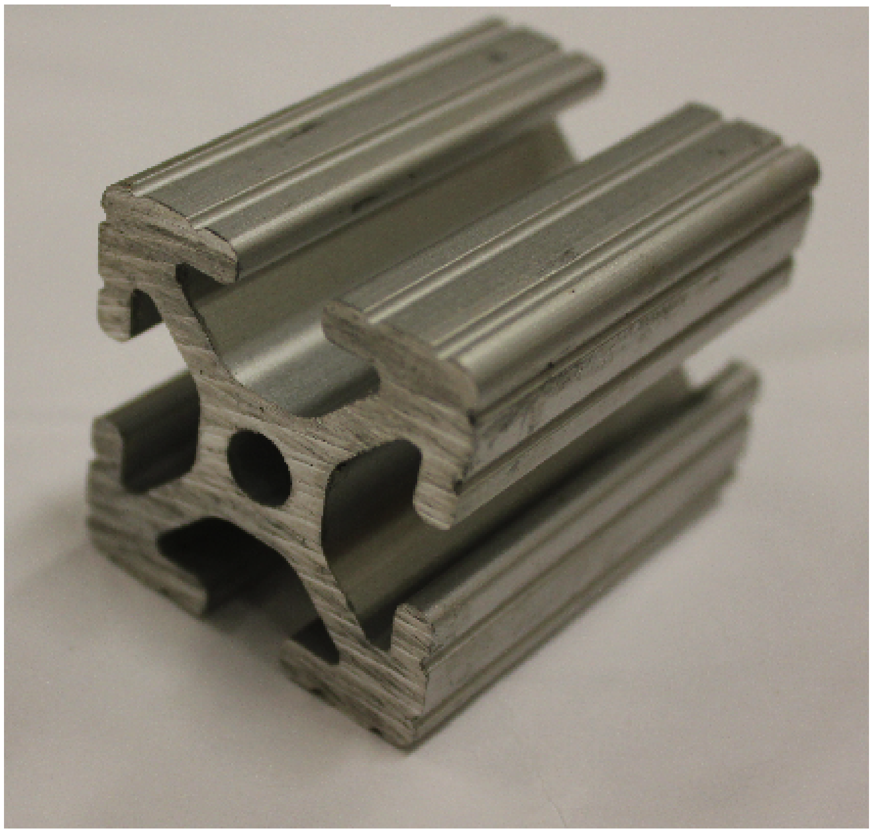


Figure E.1: 8020 aluminum member example

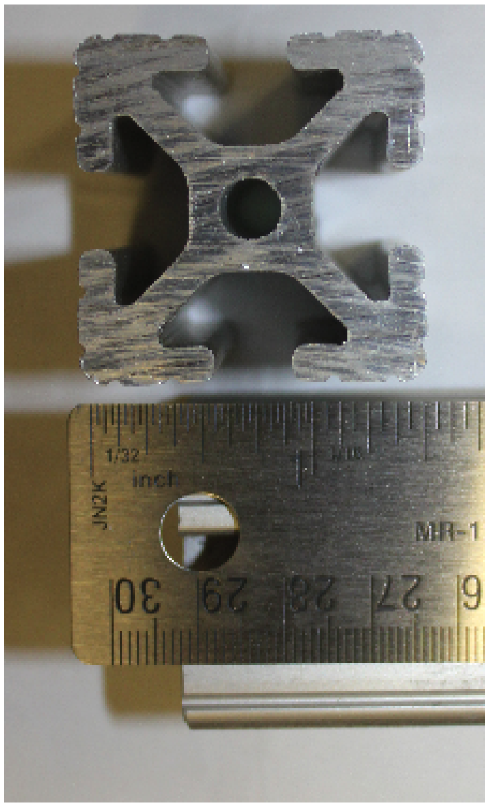


Figure E.2: 8020 aluminum section example

E.1.2 Connectors

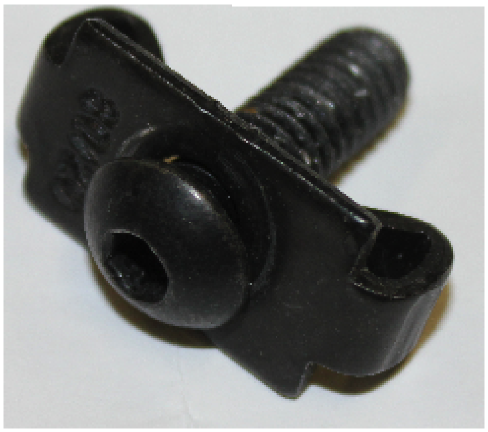


Figure E.3: Standard End Fastener, $\frac{1}{4}'' - 20$ (Part Number: 3381)

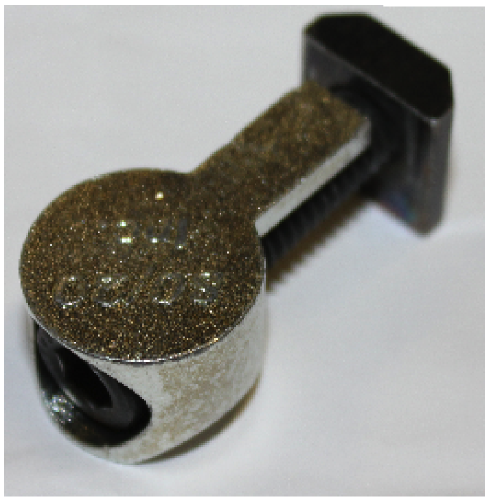


Figure E.4: 15 Series $\frac{1}{4}''-20$ Standard Slide-In T-Nut (Part Number: 3202)

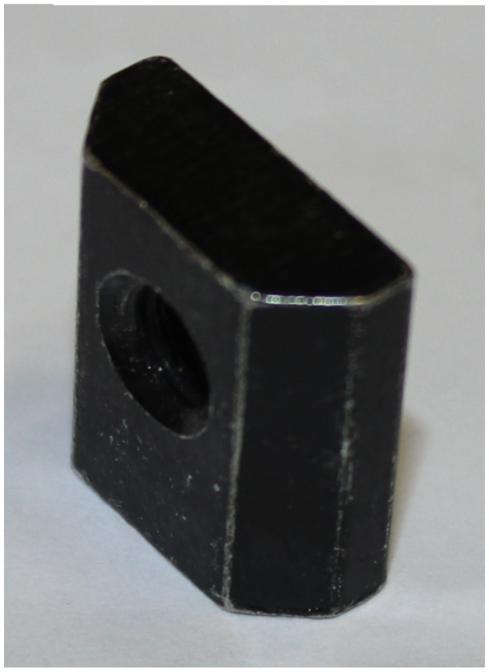


Figure E.5: 15 Series $\frac{5}{16}''-18$ Standard Anchor Fastener Assembly (Part Number: 3360)

E.2 CUBE 2.0 Frame constructed from 8020 Materials

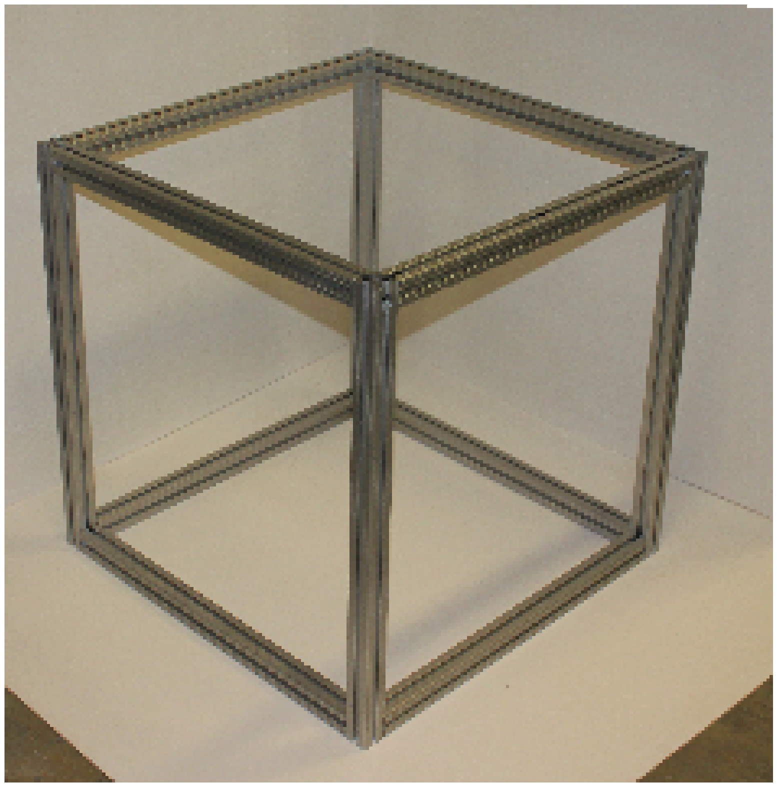


Figure E.6: The assembled 8020 frame for the CUBE 2.0 physical structure.

E.3 Plywood Enclosure

E.3.1 8020 Frame Covering

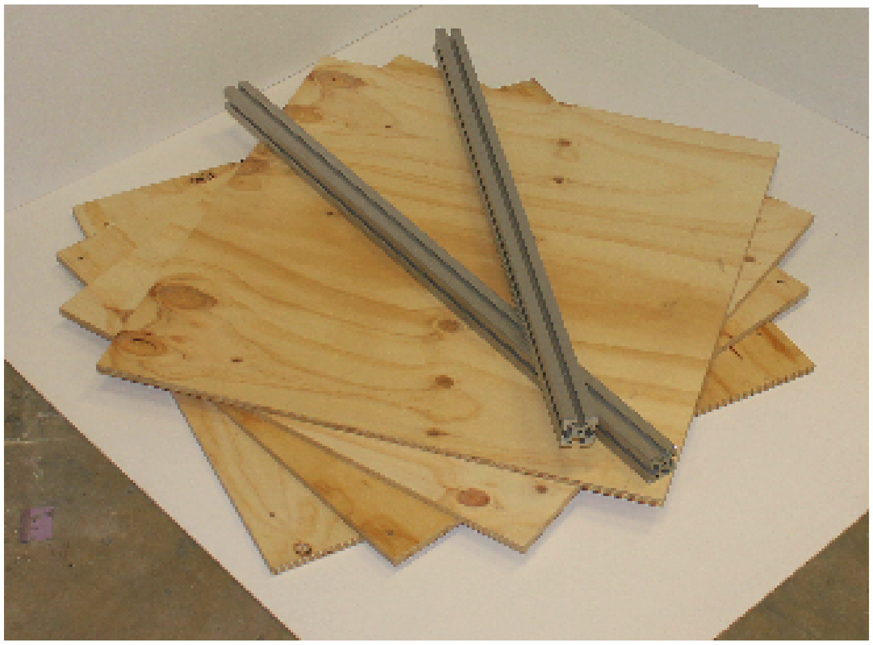


Figure E.7: These are pictures of the square sections which cover the 8020 frame in a standard manner.

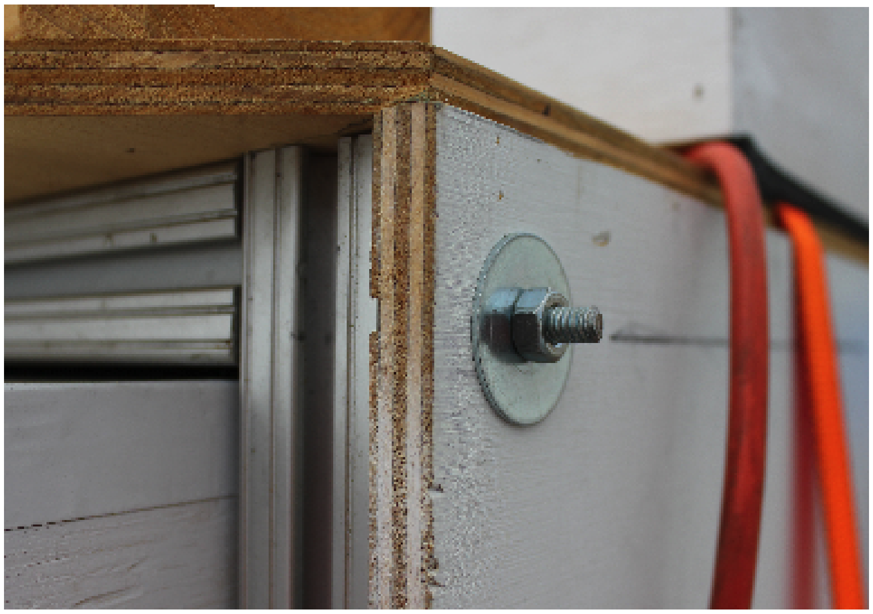


Figure E.8: Threaded rod for connecting the enclosure panel to the 8020 frame.

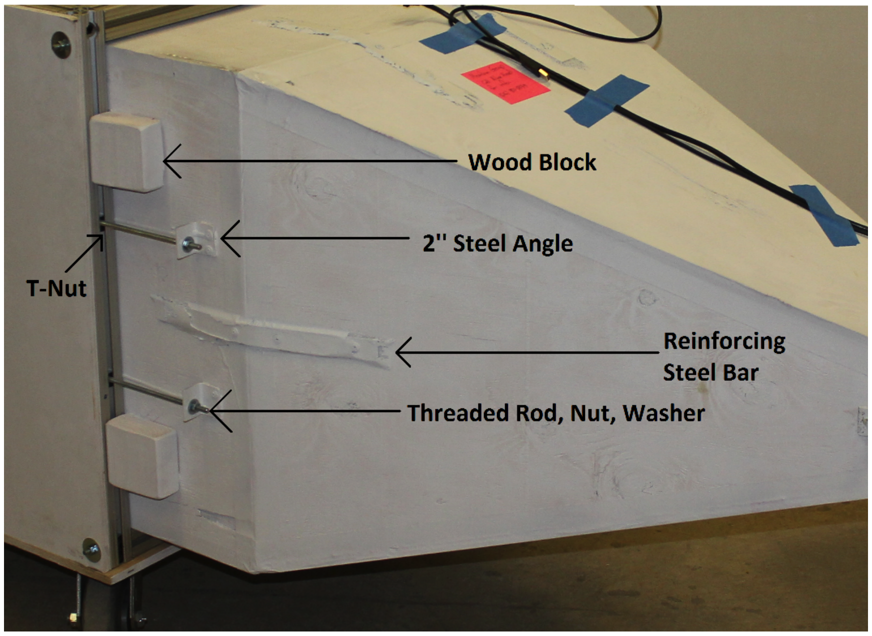


Figure E.9: Close up of reinforcing steel bars and connecting steel angles, threaded rods, nuts, washers and wooden blocks.

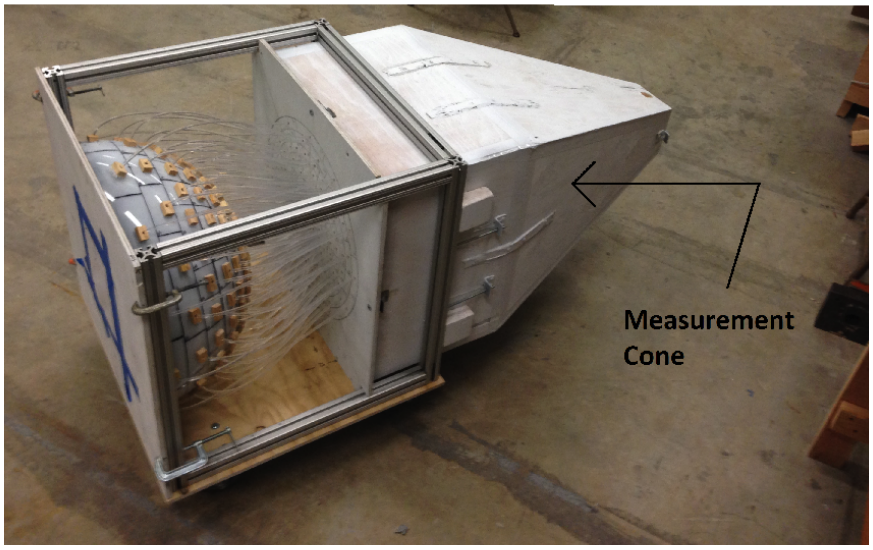


Figure E.10: The completed measurement cone attached to the CUBE 2.0 8020 frame. Note the Optical Fiber Array is also in place in this photogrpah, which is discussed below.

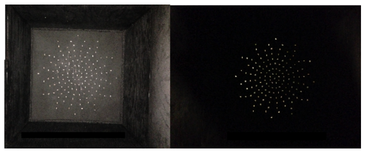


Figure E.11: The inside of the measurement cone showing the optical fiber ends which are captured by the RPiCM.

E.4 Total CUBE with Side Panels and Measurement Cone

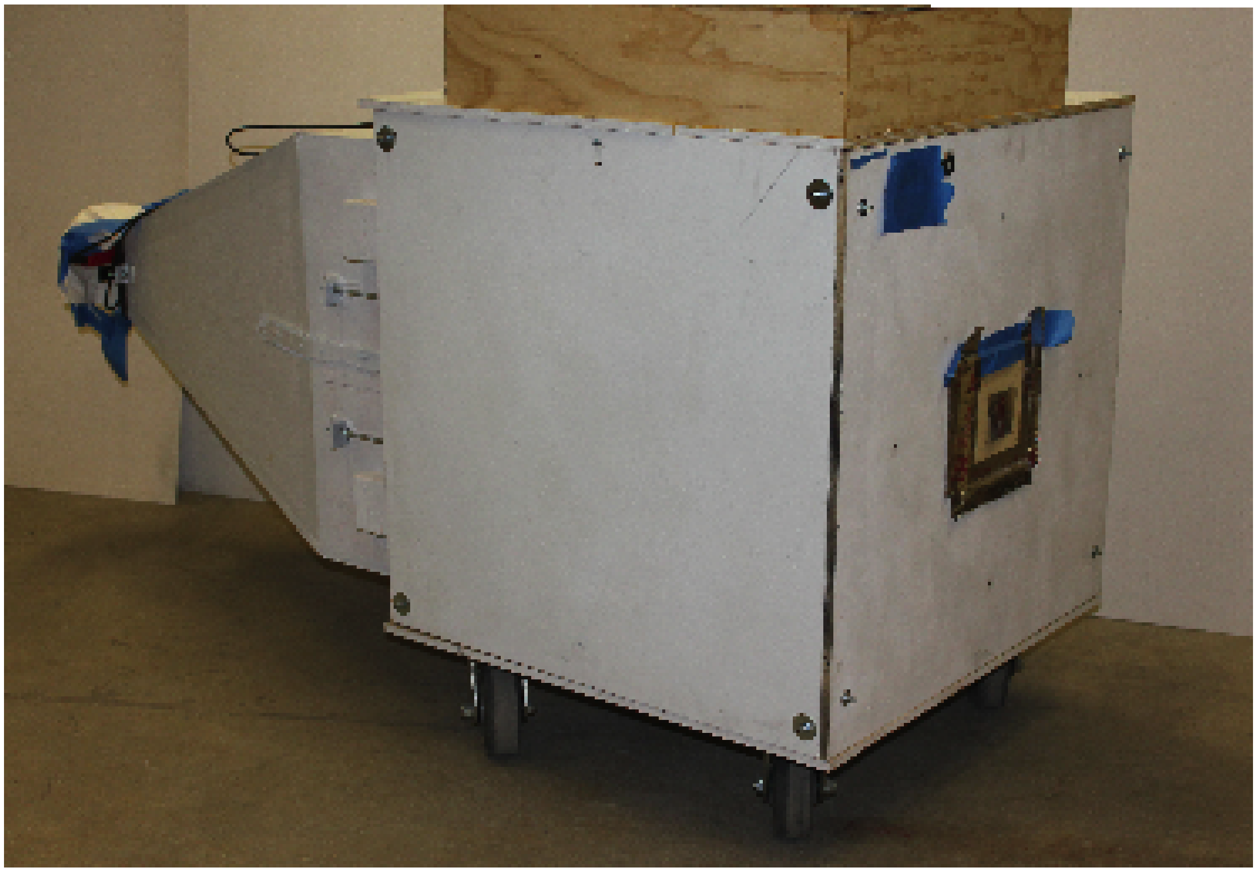


Figure E.12: The completed CUBE.

E.5 Optical Fiber Array

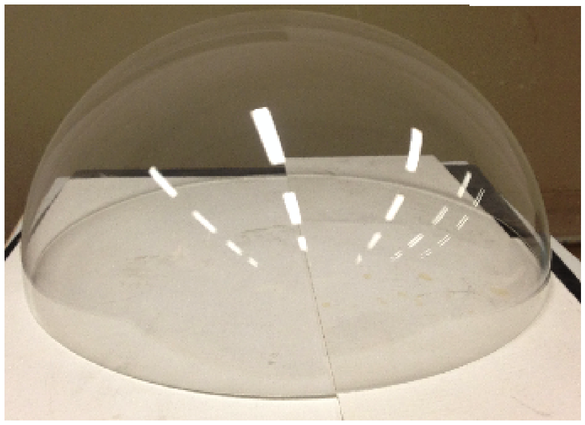


Figure E.13: 24 inch diameter acrylic hemisphere.

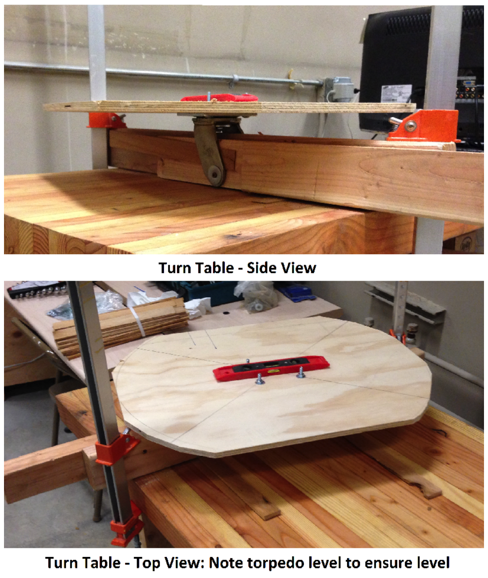


Figure E.14: Turn table used to mark Klems Basis Patches on the acrylic hemisphere.

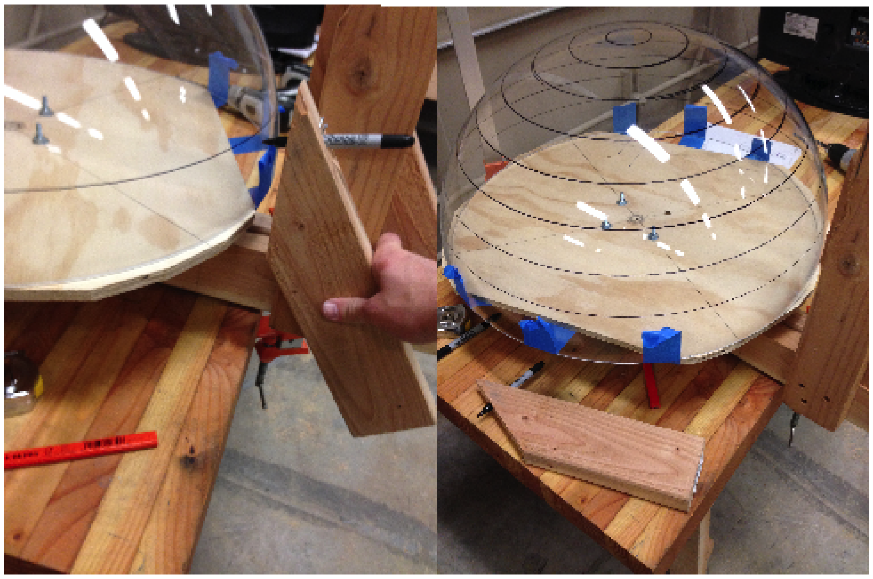


Figure E.15: Marking of the theta bands of the Klems Basis Patches on the acrylic hemisphere.

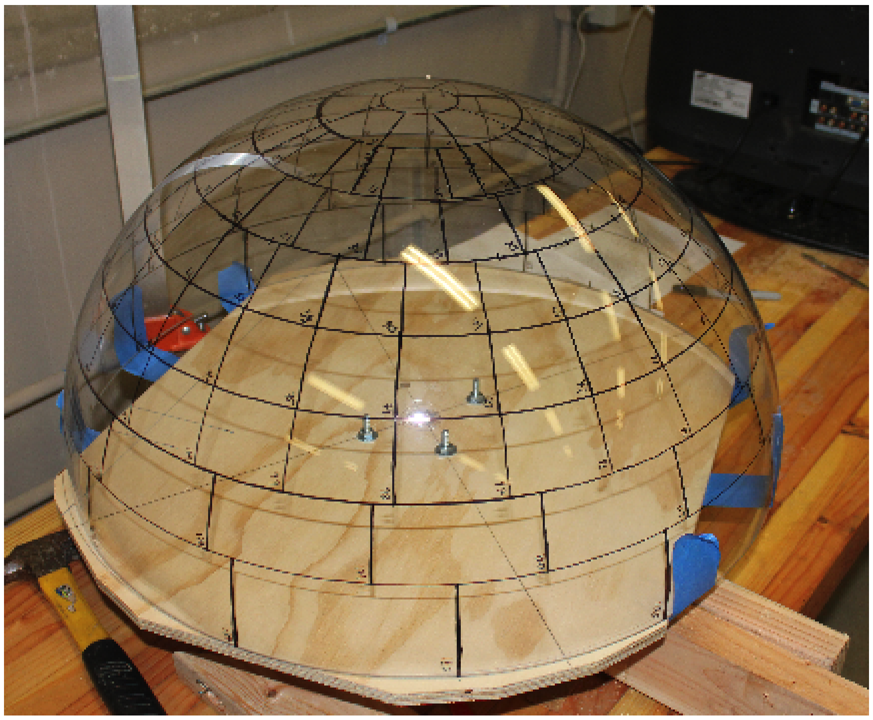


Figure E.16: The total Klems Basis Patches marked on the acrylic hemisphere.

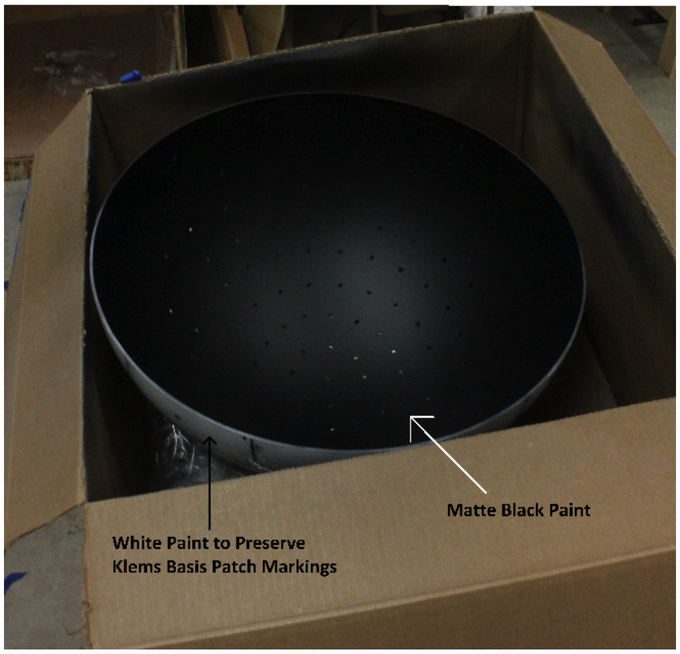


Figure E.17: White then matte black paint coats the inside of the acrylic hemisphere.

s

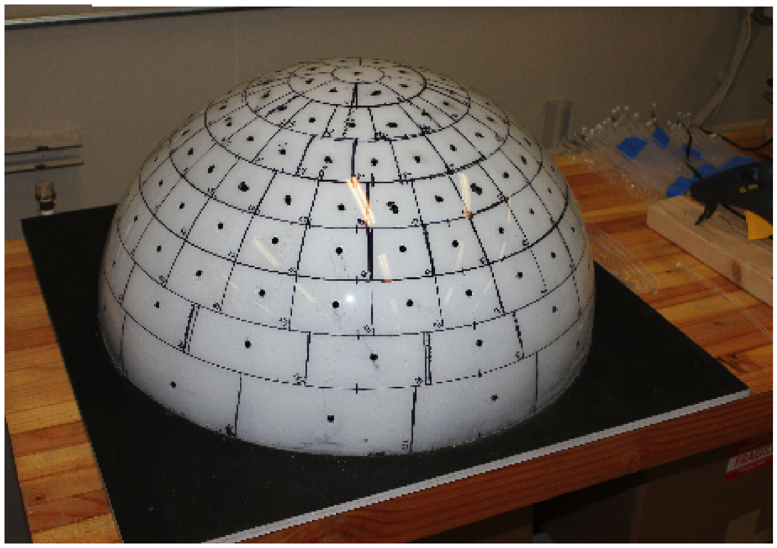


Figure E.18: The marked, drilled, and painted acrylic hemisphere is now mounted on the corresponding CUBE 2.0 enclosure plywood sheet.

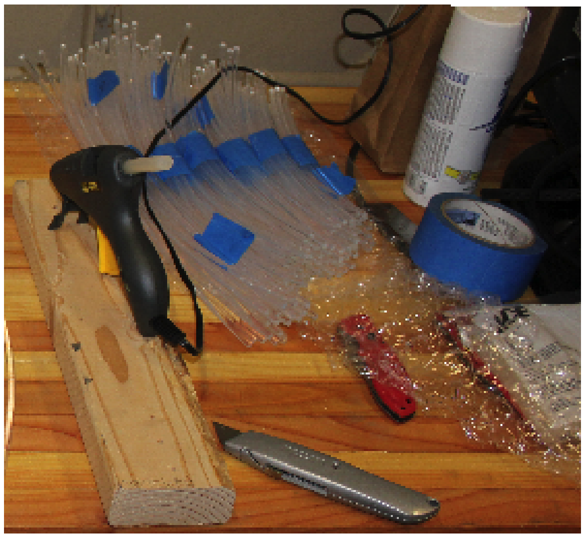


Figure E.19: The uncut $\frac{1}{4}''$ diameter, poly(methyl methacrylate) (PMMA) optical fibers.

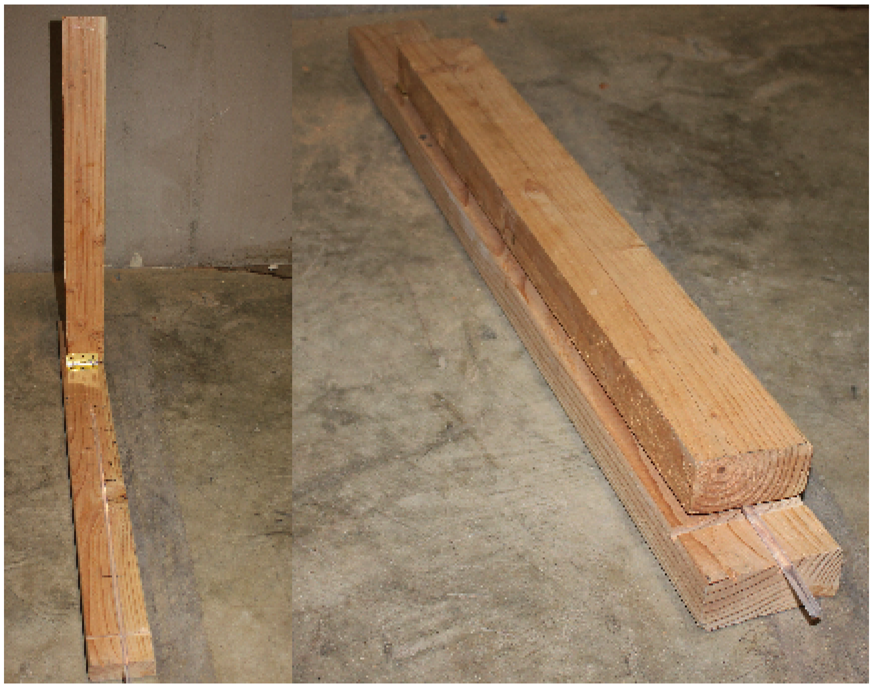


Figure E.20: A simple 2×4 cutting device used to ensure consistent optical fiber length.

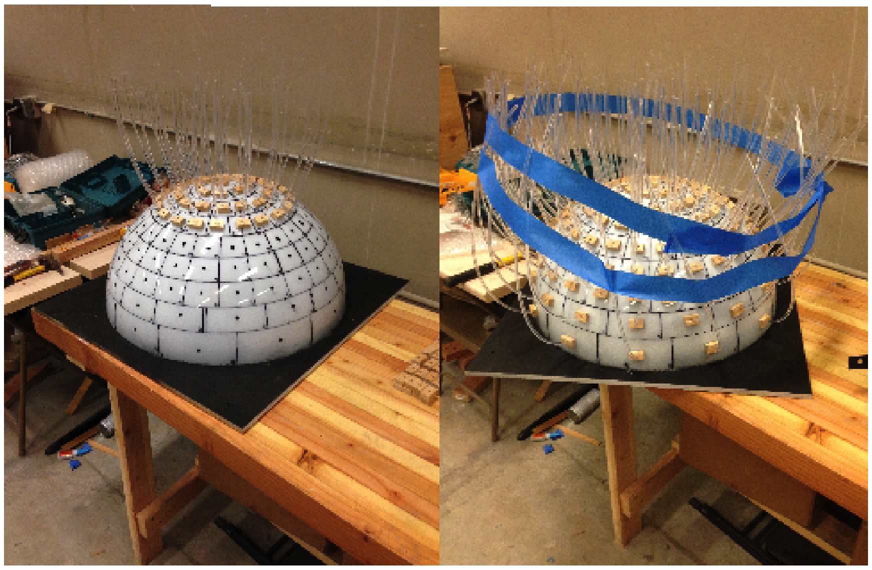


Figure E.21: “Sampler” ends of the optical fibers mounted in the hemisphere using small wood blocks.

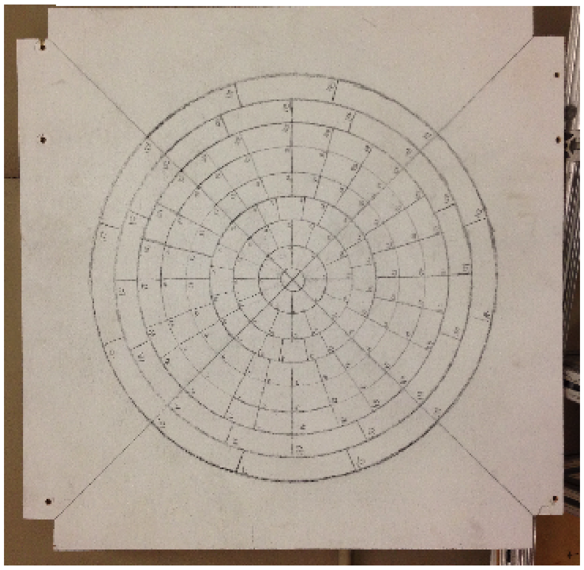


Figure E.22: Plywood sheet which holds the “measurement” end of the optical fibers.

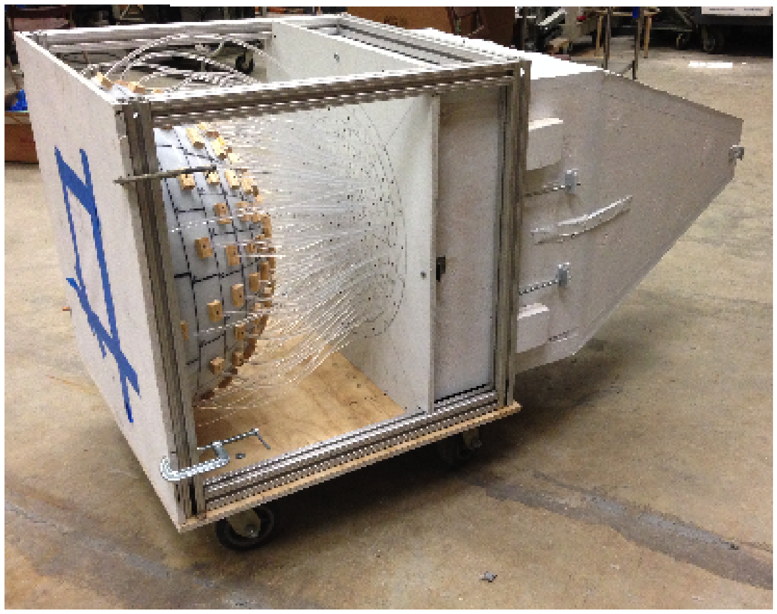


Figure E.23: The total optical fiber sampling system from exiting luminance measurement hemisphere holding the “sampling” optical fiber end, through optical fiber “measurement” end terminating at the *measurement cone*.

E.6 Raspiberry Pi Camera Module Bracket

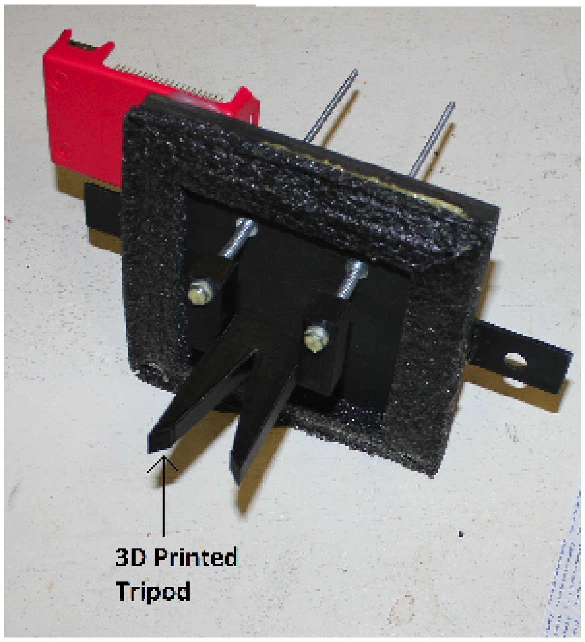


Figure E.24: 3D printed tripod camera holder for the Raspberry Pi Camera Module inside the measurement cone.

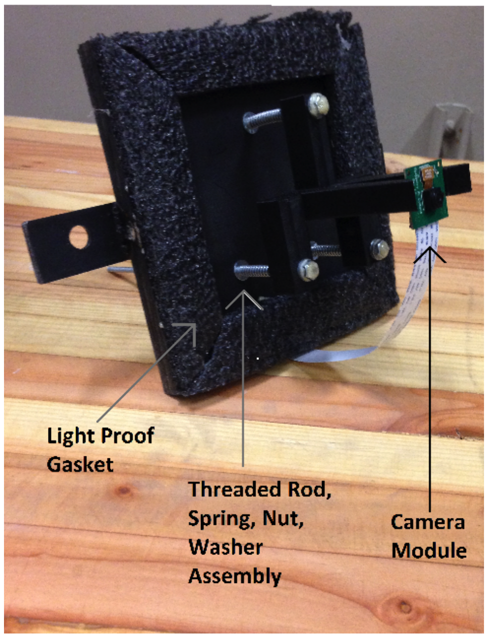


Figure E.25: RPiCM camera holder mounted on a plywood sheet, ready to be placed within the measurement cone.

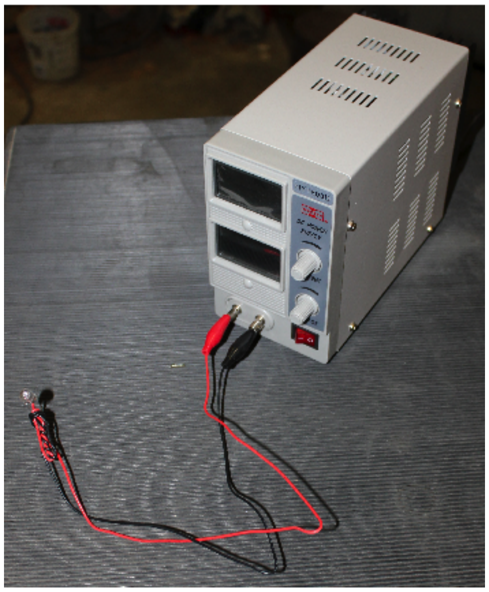


Figure E.26: 50 W Halogen lamp used for “sampler” optical fiber end excitation.

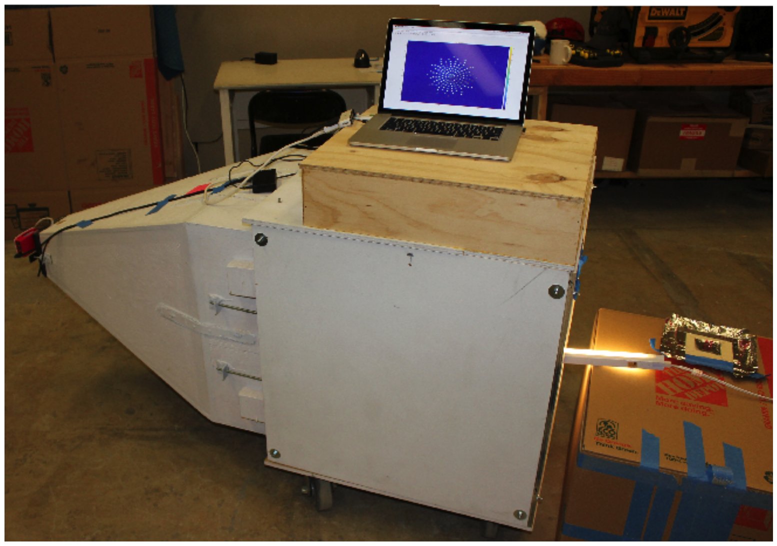


Figure E.27: Small fluorescent light placed in the CFS test aperture. Note, 50 W Halogen Lamp is now used as it emits light in all directions.

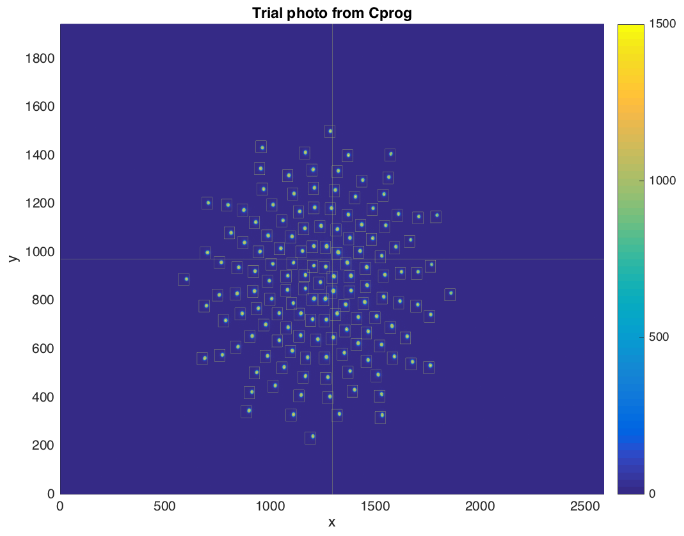


Figure E.28: Matlab Surf plot showing aligned RPiCM. Notice the yellow "boxes" surrounding the optical fiber ends.

E.7 Complex Fenestration System Specimen Holder

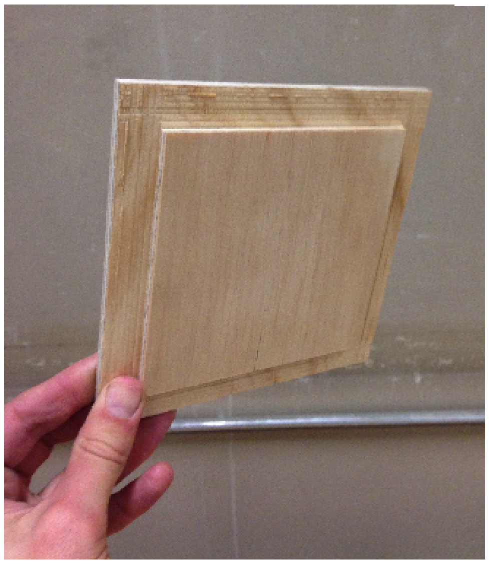


Figure E.29: Precision cut plywood insert before CFS apertur drilling.

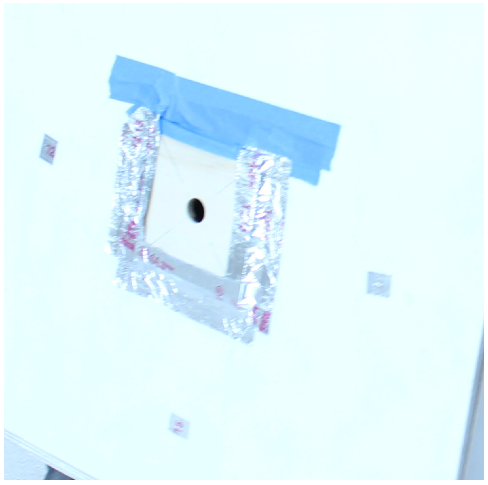


Figure E.30: Original plywood CFS holder in place on CUBE 2.0.

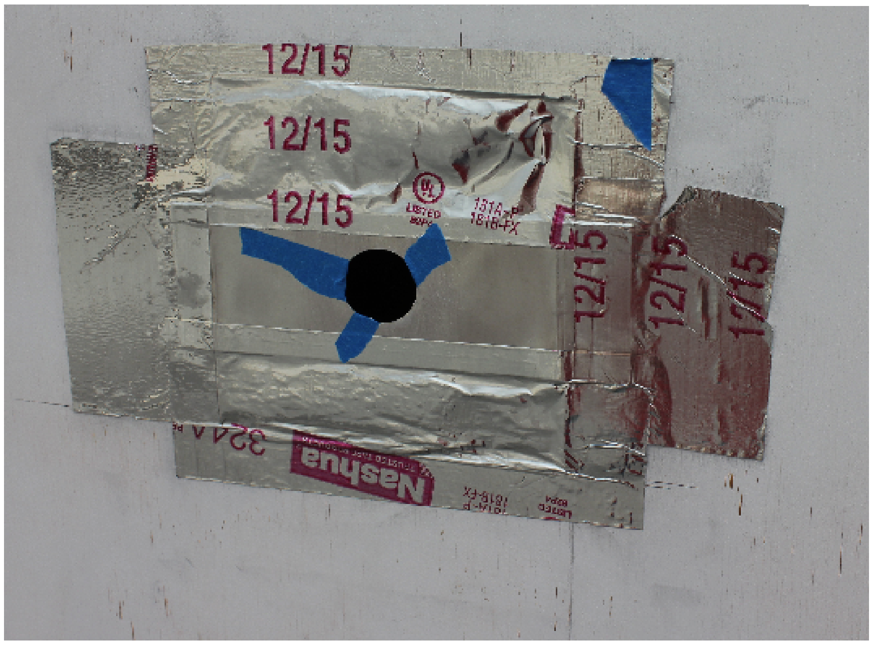


Figure E.31: Brass CFS holder in place on CUBE 2.0. The 1 inch hole is the CFS test aperture.

E.8 Canon 6D, Sigma Fisheye Lens, LiCor LI-210

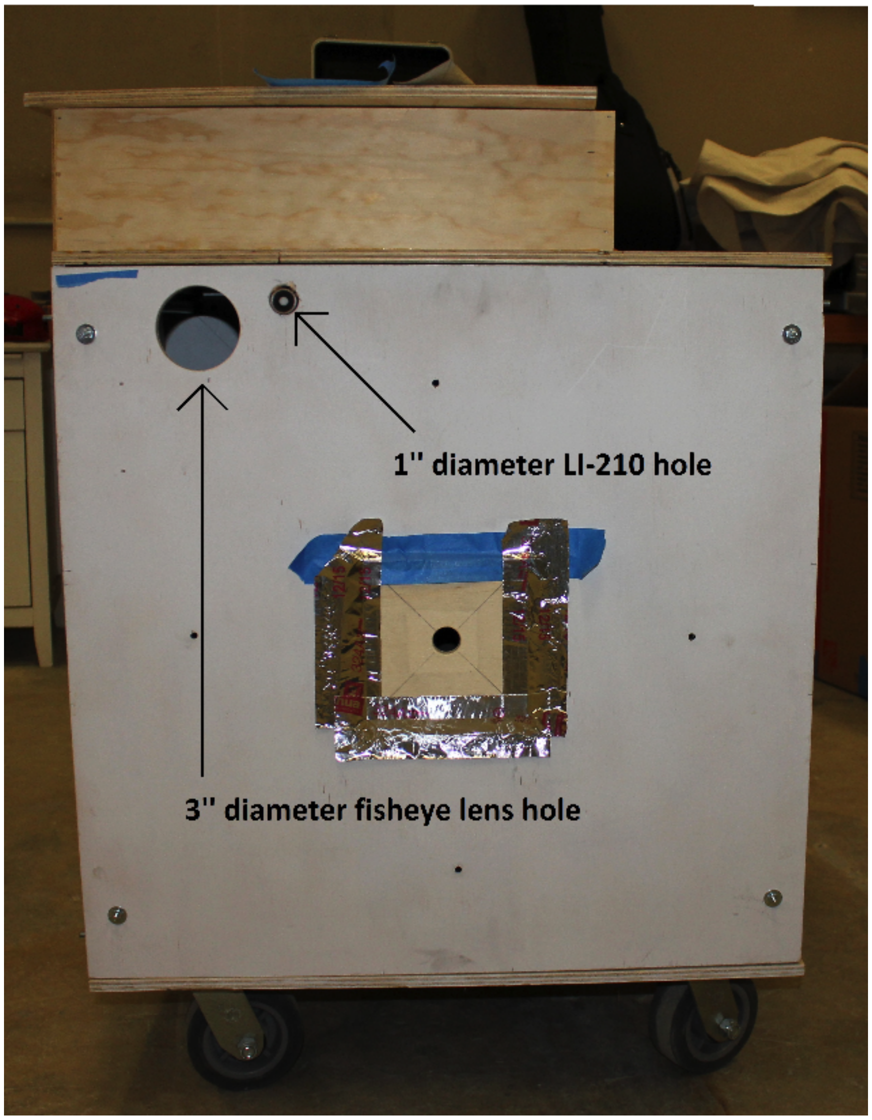


Figure E.32: Fisheye lens and LI-210 sensor holes. Note, the LI-210 hole is being test fit in the picture.

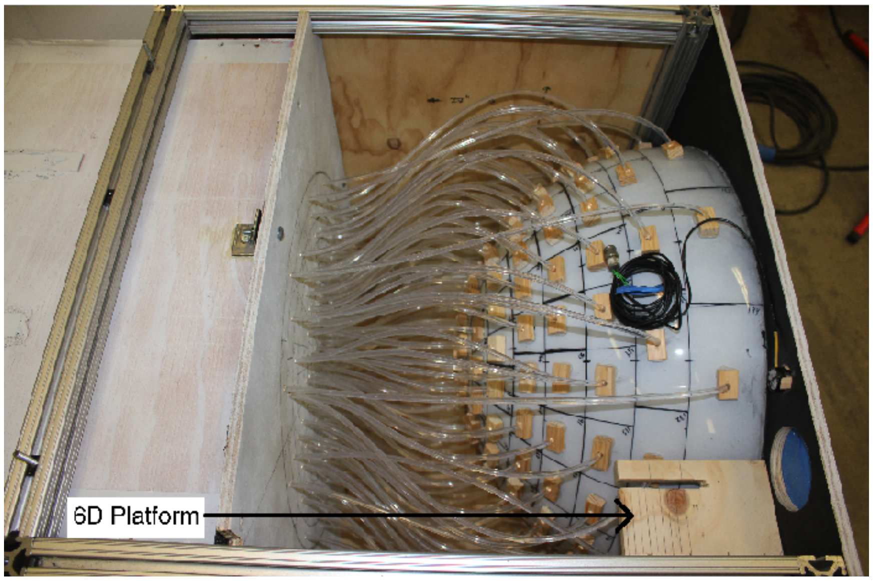


Figure E.33: Inside of CUBE 2.0 view of Canon 6D camera platform. Note, the LI-210 is fit into its hole above the 3" diameter fisheye lens hole.

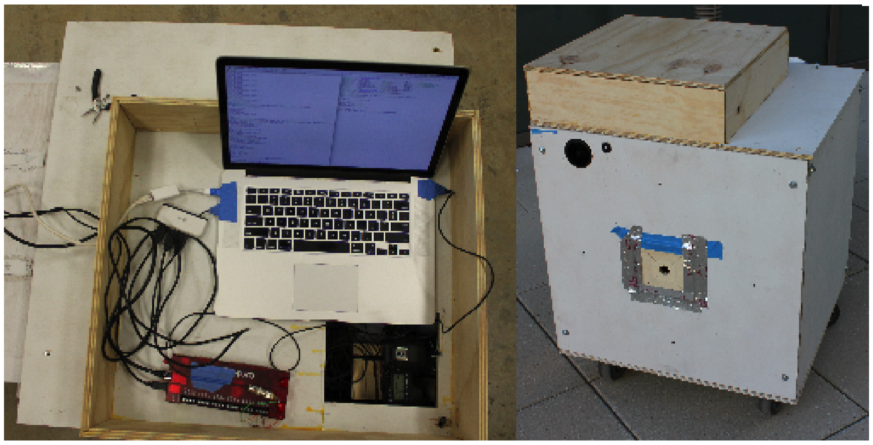


Figure E.34: Left: Inside view of CUBE 2.0 with all components present. Right: Outside view of CUBE 2.0 with fisheye lens and LI-210 visible. Note, the older plywood CFS specimen holder is present, not the final brass one.

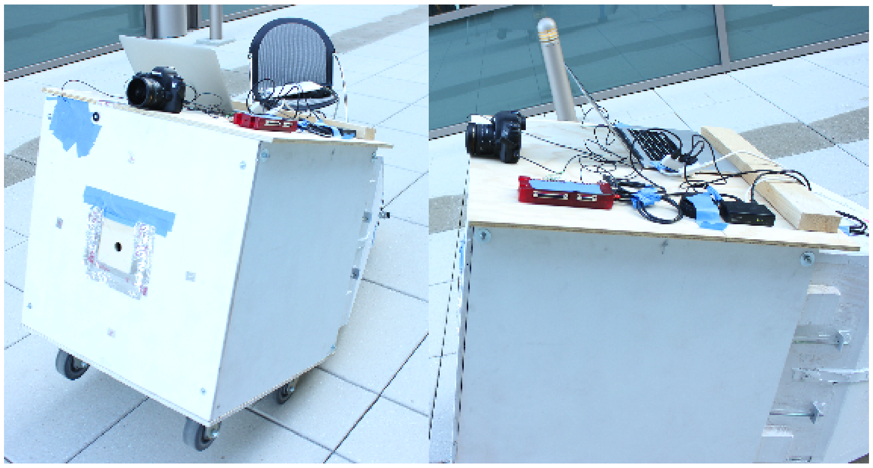


Figure E.35: Two views of the second camera mounting scheme, both head on (left) and angled (right).

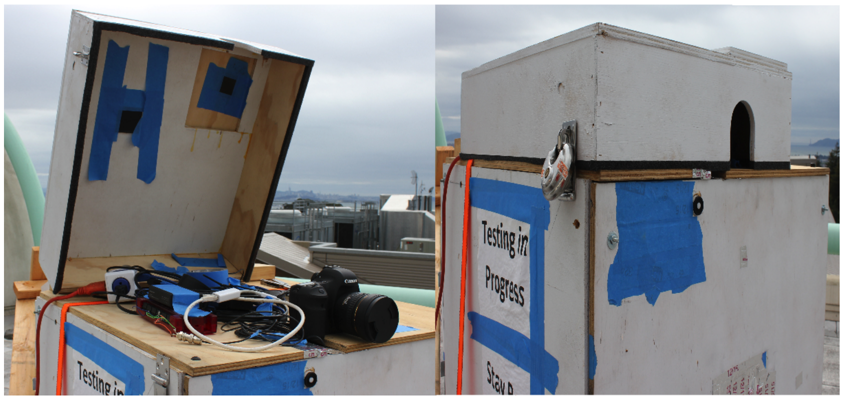


Figure E.36: The second enclosure for the Canon 6D, MacBook Pro, Ethernet Router, USB connector, Power Hub.

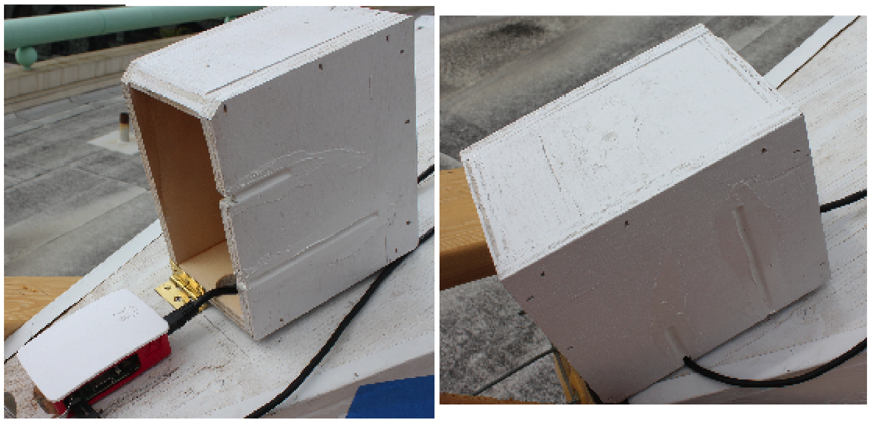


Figure E.37: Enclosure for the RPiCM as to protect it from the elements.

E.9 Transportation and Support

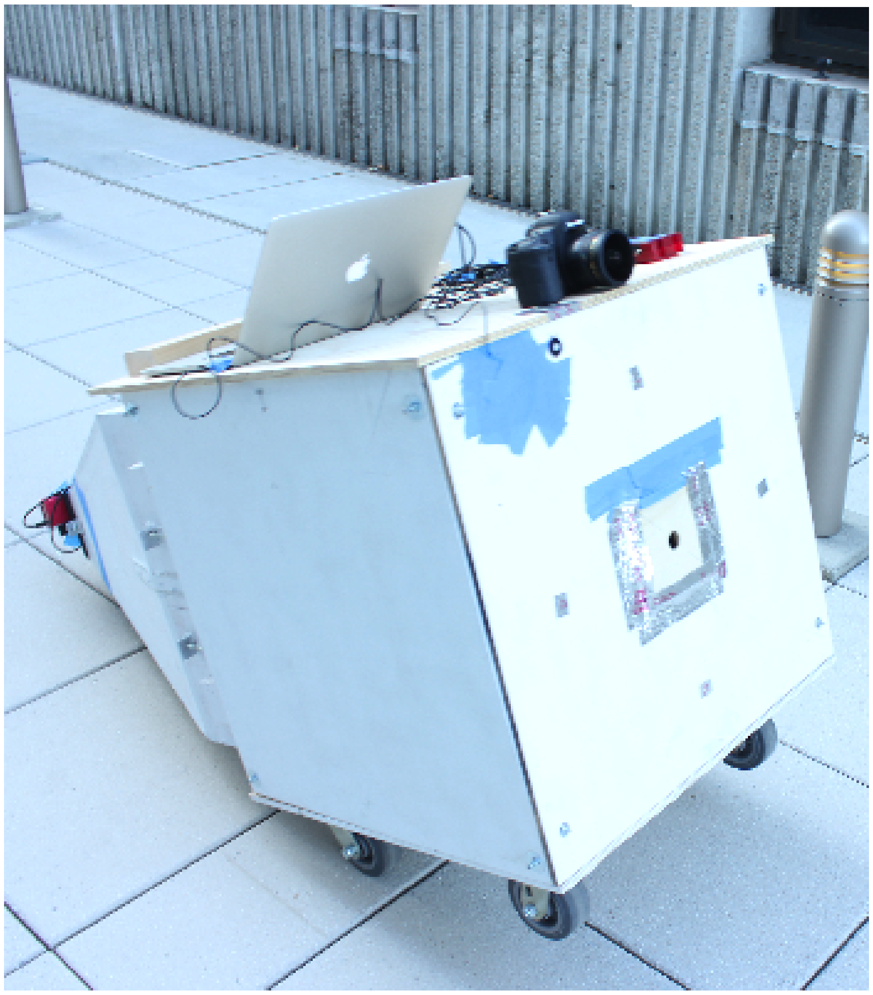


Figure E.38: Casters attached to the CUBE 2.0 for easy transportation. Note, the CUBE 2.0 is tilted intentionally, it remains balanced on all four casters without assistance.

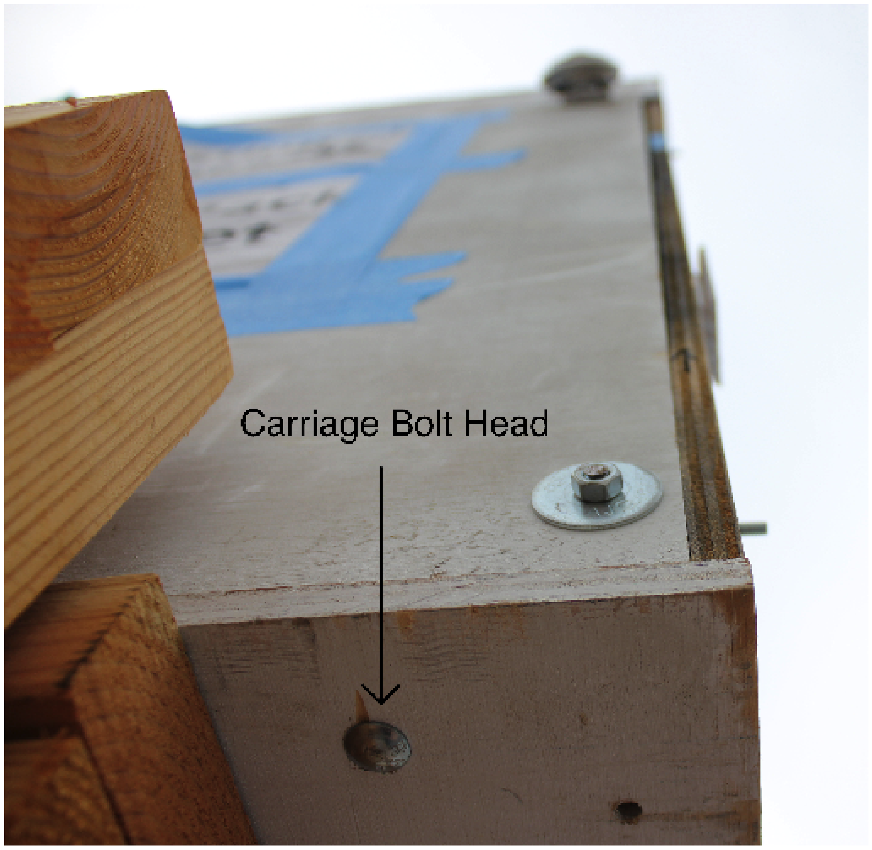


Figure E.39: Carriage bolt heads extending from the bottom of the CUBE 2.0. These steel "dishes" of sorts allow for low friction sliding of the CUBE 2.0 system.

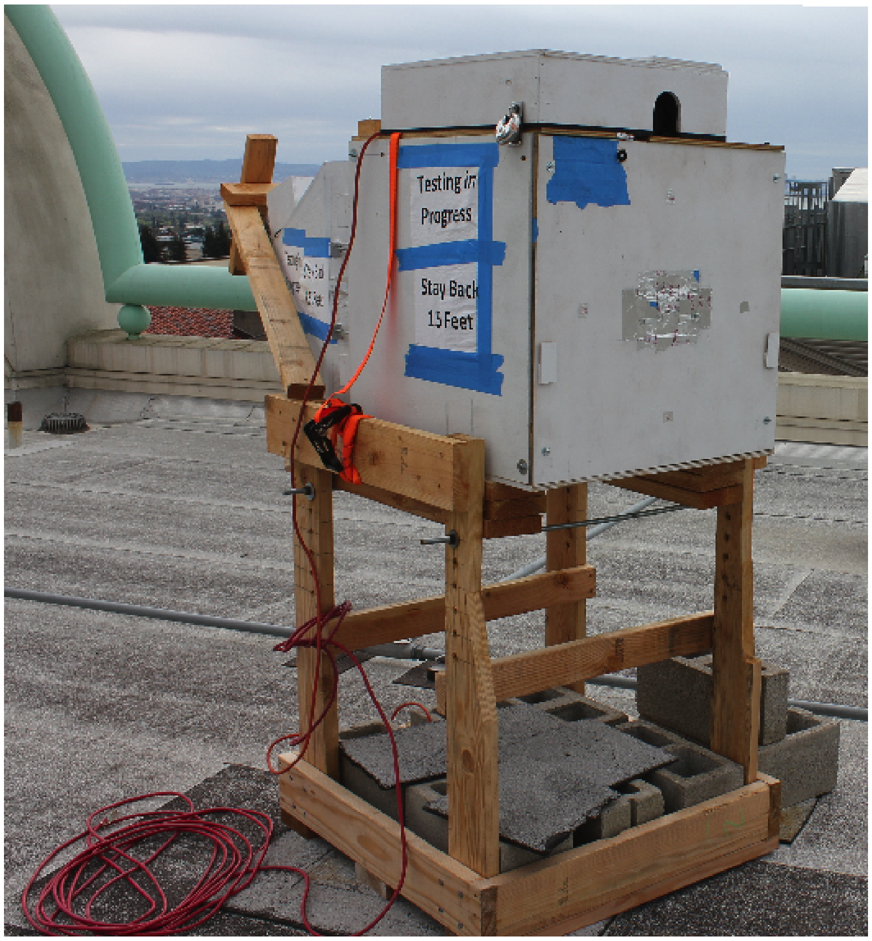


Figure E.40: CUBE 2.0 support frame, which is permanently deployed on the roof of Cory Hall at the University of California, Berkeley.

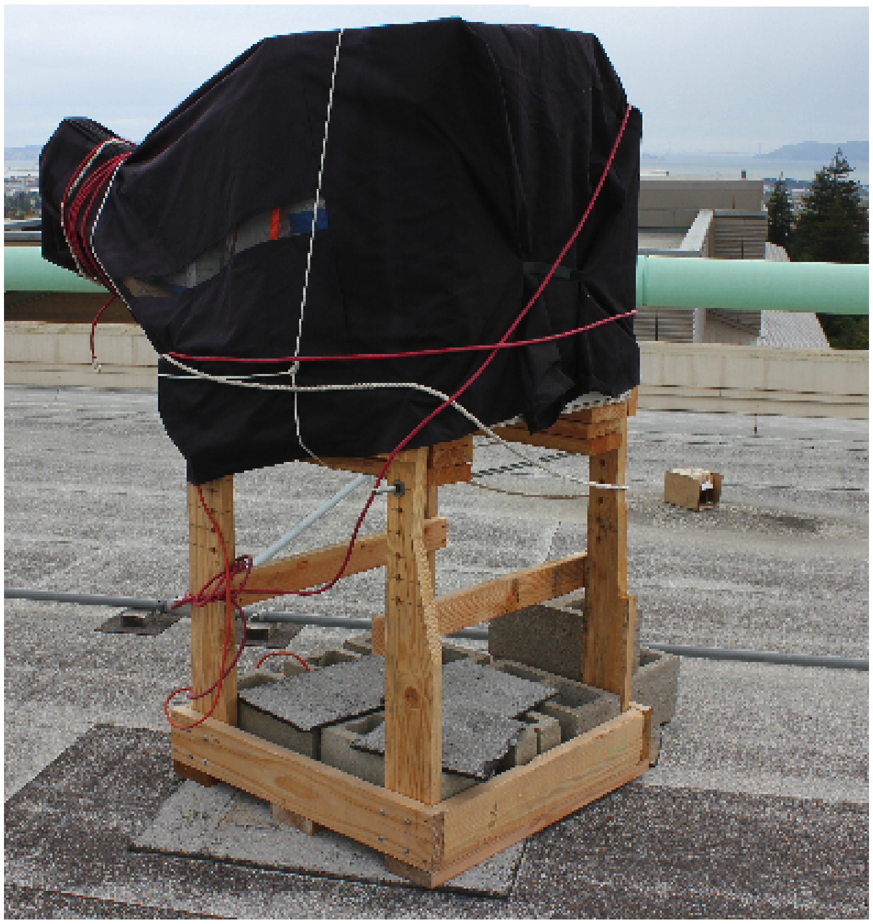


Figure E.41: CUBE 2.0 on roof of Cory Hall with rain cover installed. Notice iconic Gold Gate Bridge in the background.

E.10 Bench Top Testing

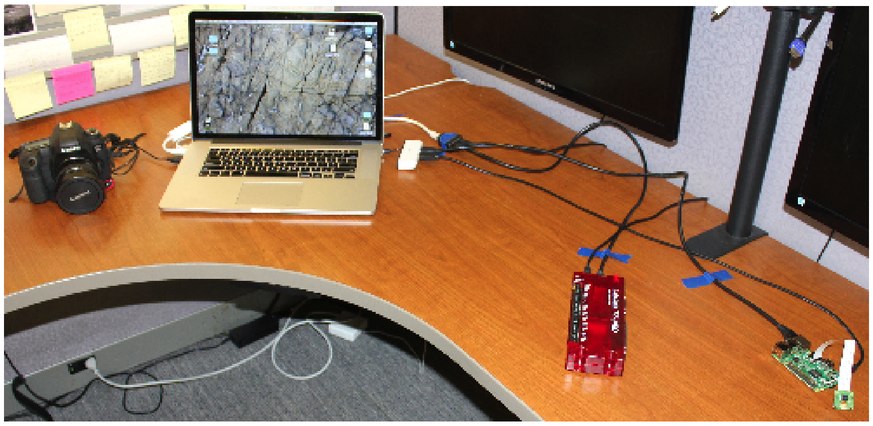


Figure E.42: View of the hardware (MacBook Pro, RPiCM, LabJack T7-Pro, Canon 6D) during a network testing deployment for the structure of the code.

Appendix F

Full Analysis: Canon 6D and Fisheye Lens, LI-210, and CFS Test Aperture Alignment

F.1 Simulation: Radiance Model - CUBE 2.0 on Plane

F.1.1 Model Construction

The Radiance model used for analyzing the input illuminance sensitivity is simply a plane surface with an associated sky, solar disk, and ground "glow." The illuminance measurements which are calculated are obtained by using an *xform* call to move the plane into the needed position. This is required so it can replicate the physical relation of the three components on the real CUBE 2.0 system.

F.1.2 Clear Sky Excitation Conditions

As justified by Mardaljevic [99], the CUBE 2.0 system was excited by 13 clear skies with solar disk position spanning the possible locations for Berkeley, California. The exact θ and ϕ position of the disk can be seen below in the results figures. It should be noted these values refer to the Radiance coordinate system. This offers an insight into the parallax input errors ($L(\theta,\phi)_{cfs} \approx L(\theta,\phi)_{fish} \approx L(\theta,\phi)_{LI-210}$), a characterization which is further explored below with real measurements using the Canon 6D and fisheye lens.

F.1.3 Radiance Parameter Calculations

The exact Radiance commands used for the analysis are presented below. Note, CFS-aperture.rad is simply a polygon of circular shape to represent the various apertures, further, because *rtcontrib* is used, three calls (i.e. for the sky, solar disk, and ground) are needed for the CFS test aperture, Canon 6D, and the LI-210. The results are summed in post analysis.

```
#!/bin/sh
echo 'stating the stript'
for i in `seq 1 13`

do

  # First, compile the new octree for the proper sky.
  oconv materials.rad putPlane$i.rad > new.oct
  echo 'New octree made'

  # Second, call the rtrace commands
  # CFS Aperture
  echo '0.9024 -0.075 1.3515 1 0 0'|rtrace -h -I -w -ab 5 new.oct
      |rcalc -e '$1=($1*0.265+$2*0.670+$3*0.065)*179' > '
     studyDataPlaneCor/sky'$i'_E_Ta.dat'
  # Fisheye Aperture
  echo '0.9024 -0.075 1.7615  1 0 0'|rtrace -h -I -w -ab 5 new.oct
      |rcalc -e '$1=($1*0.265+$2*0.670+$3*0.065)*179' > '
     studyDataPlaneCor/sky'$i'_E_Tb.dat'
  # LiCor Aperture
  echo '0.9024 -0.229 1.6815  1 0 0'|rtrace -h -I -w -ab 5 new.oct
      |rcalc -e '$1=($1*0.265+$2*0.670+$3*0.065)*179' > '
     studyDataPlaneCor/sky'$i'_E_Tc.dat'

  echo 'rtrace calculations done'

  # Third, call the rtcontrib commands
  # CFS Aperture
  genklemsamp -c 1000 -vd 1 0 0 -vu 0 0 1 ./CFSaperture.rad |
      rtcontrib @rtc.opt -V+ -c 1000 -m skyglow new.oct |tail -145
     > 'studyDataPlaneCor/sky'$i'_E_ka_sky.dat'
  genklemsamp -c 1000 -vd 1 0 0 -vu 0 0 1 ./CFSaperture.rad |
      rtcontrib @rtc.opt -V+ -c 1000 -m solar new.oct |tail -145 >
      'studyDataPlaneCor/sky'$i'_E_ka_disk.dat'
  genklemsamp -c 1000 -vd 1 0 0 -vu 0 0 1 ./CFSaperture.rad |
      rtcontrib @rtc.opt -V+ -c 1000 -m ground_glow new.oct |tail
      -145 > 'studyDataPlaneCor/sky'$i'_E_ka_ground.dat'
  echo 'rtcontrib for CFS done'

  # Fisheye Aperture
  genklemsamp -c 1000 -vd 1 0 0 -vu 0 0 1 ./fisheyeAperture.rad |
     rtcontrib @rtc.opt -V+ -c 1000 -m skyglow new.oct |tail -145
     > 'studyDataPlaneCor/sky'$i'_E_kb_sky.dat'
  genklemsamp -c 1000 -vd 1 0 0 -vu 0 0 1 ./fisheyeAperture.rad |
     rtcontrib @rtc.opt -V+ -c 1000 -m solar new.oct |tail -145 >
     'studyDataPlaneCor/sky'$i'_E_kb_disk.dat'
  genklemsamp -c 1000 -vd 1 0 0 -vu 0 0 1 ./fisheyeAperture.rad |
     rtcontrib @rtc.opt -V+ -c 1000 -m ground_glow new.oct |tail
     -145 > 'studyDataPlaneCor/sky'$i'_E_kb_ground.dat'
  echo 'rtcontrib for fisheye done'

  # LiCor Aperture
  genklemsamp -c 1000 -vd 1 0 0 -vu 0 0 1 ./licorAperture.rad |
     rtcontrib @rtc.opt -V+ -c 1000 -m skyglow new.oct |tail -145
     > 'studyDataPlaneCor/sky'$i'_E_kc_sky.dat'
  genklemsamp -c 1000 -vd 1 0 0 -vu 0 0 1 ./licorAperture.rad |
     rtcontrib @rtc.opt -V+ -c 1000 -m solar new.oct |tail -145 >
     'studyDataPlaneCor/sky'$i'_E_kc_disk.dat'
  genklemsamp -c 1000 -vd 1 0 0 -vu 0 0 1 ./licorAperture.rad |
     rtcontrib @rtc.opt -V+ -c 1000 -m ground_glow new.oct |tail
     -145 > 'studyDataPlaneCor/sky'$i'_E_kc_ground.dat'
  echo 'rtcontrib for licor done'

  sleep 0.3
  echo 'Done with sky '$i

done

sleep 0.3
tput bel
tput bel
tput bel
sleep 0.3
tput bel
tput bel
tput bel
sleep 0.3
```

F.1.4 Klems Basis Input Numbers

The results from analyzing the 13 different skies are as follows:

Sky 1: Sun Position $\theta = 14°$, $\phi = 252°$

Rendering of Sky View

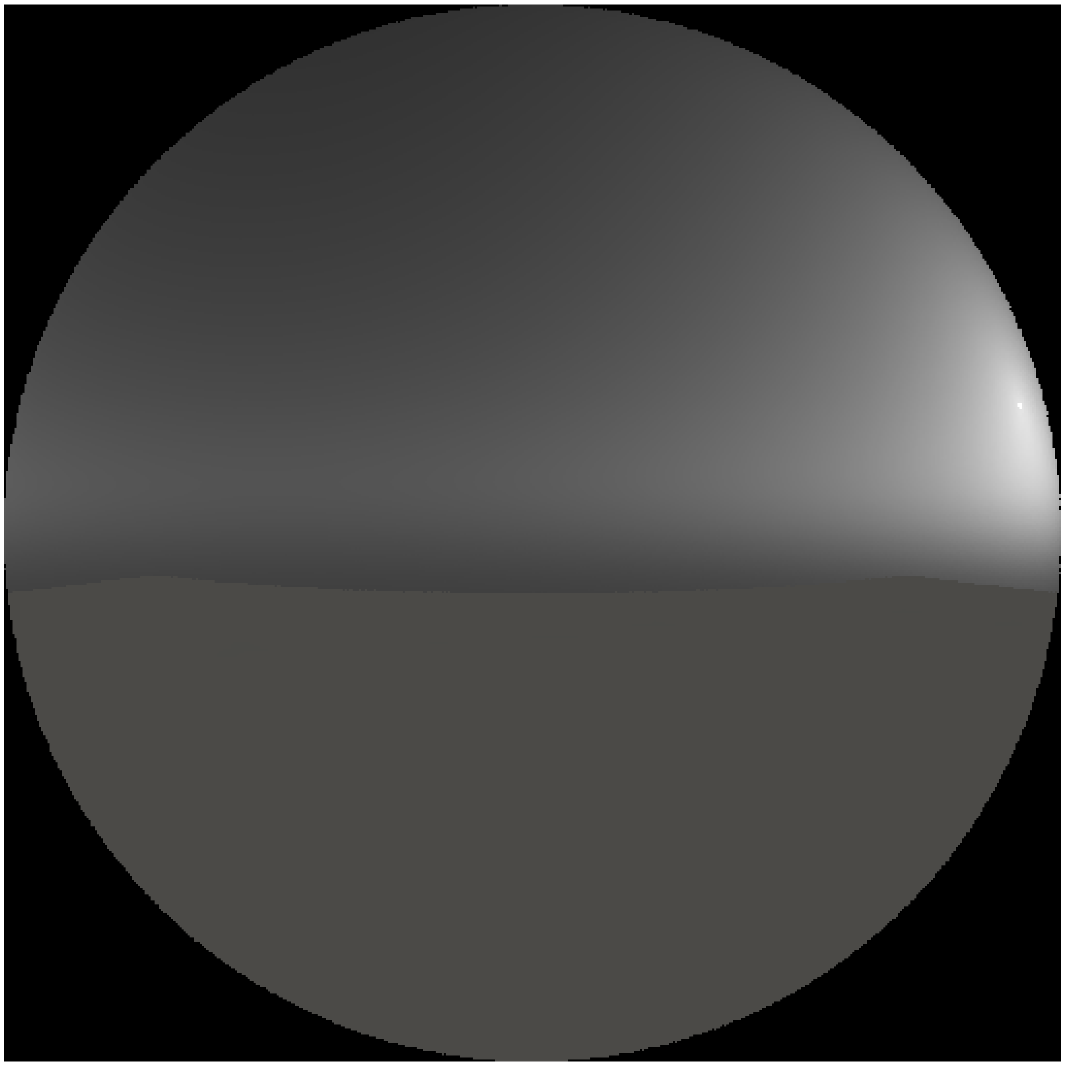


Figure F.1: Rendering looking out of the CFS test aperature, sky 1.

Absolute Input Illuminance per Klems Basis patch

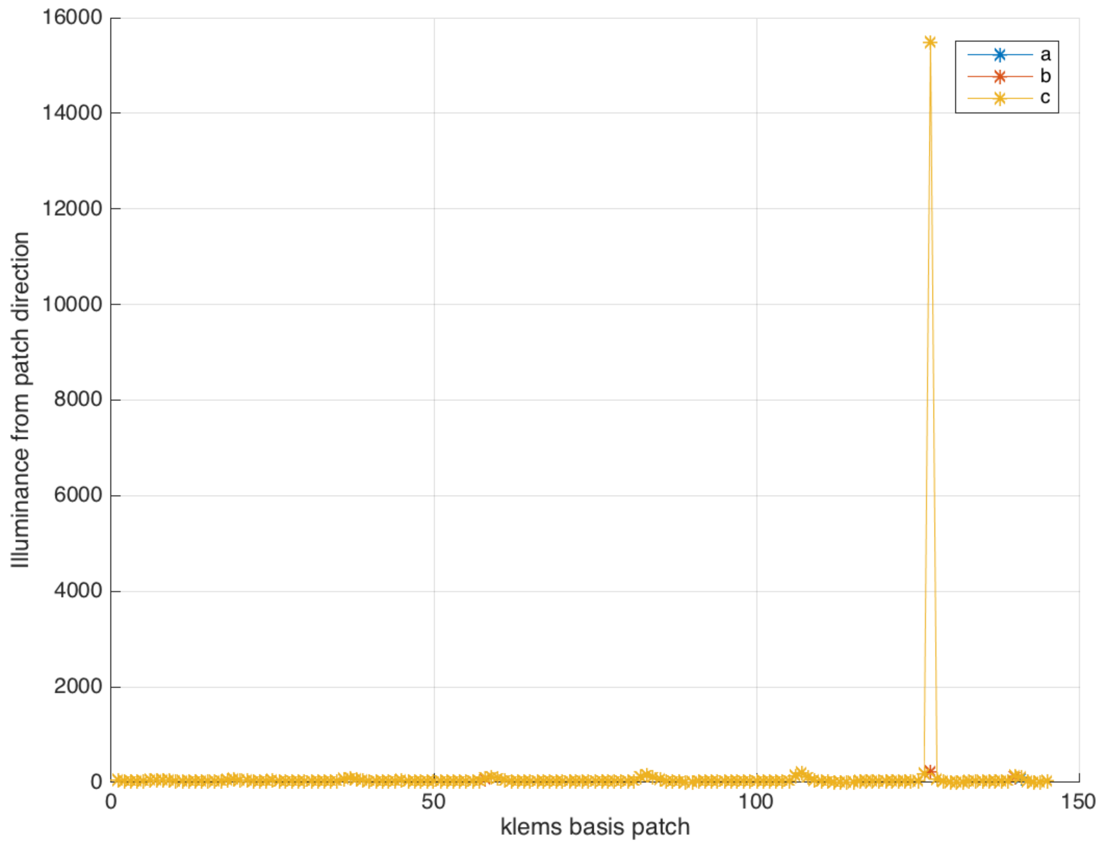


Figure F.2: Absolute input illuminance per Klems Basis patch.

Input Illuminance Error Percentage

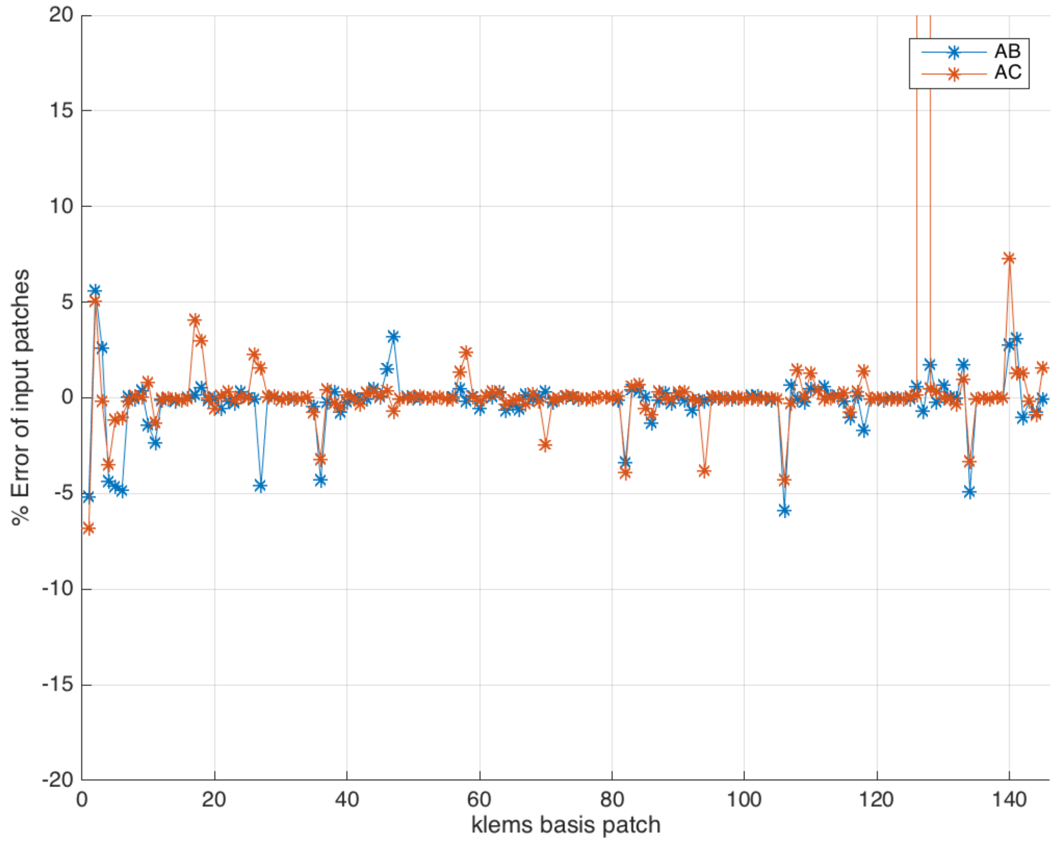


Figure F.3: Error between CFS test aperture and fisheye lens and Li-Cor respectively.

NOTE: Sun positions 2 through 12 have been omitted for length reasons. Their analysis is similar to that provided in positions 1 and 13.

Sky 13: Sun Position $\theta = 0°$, $\phi = 90°$

Rendering of Sky View

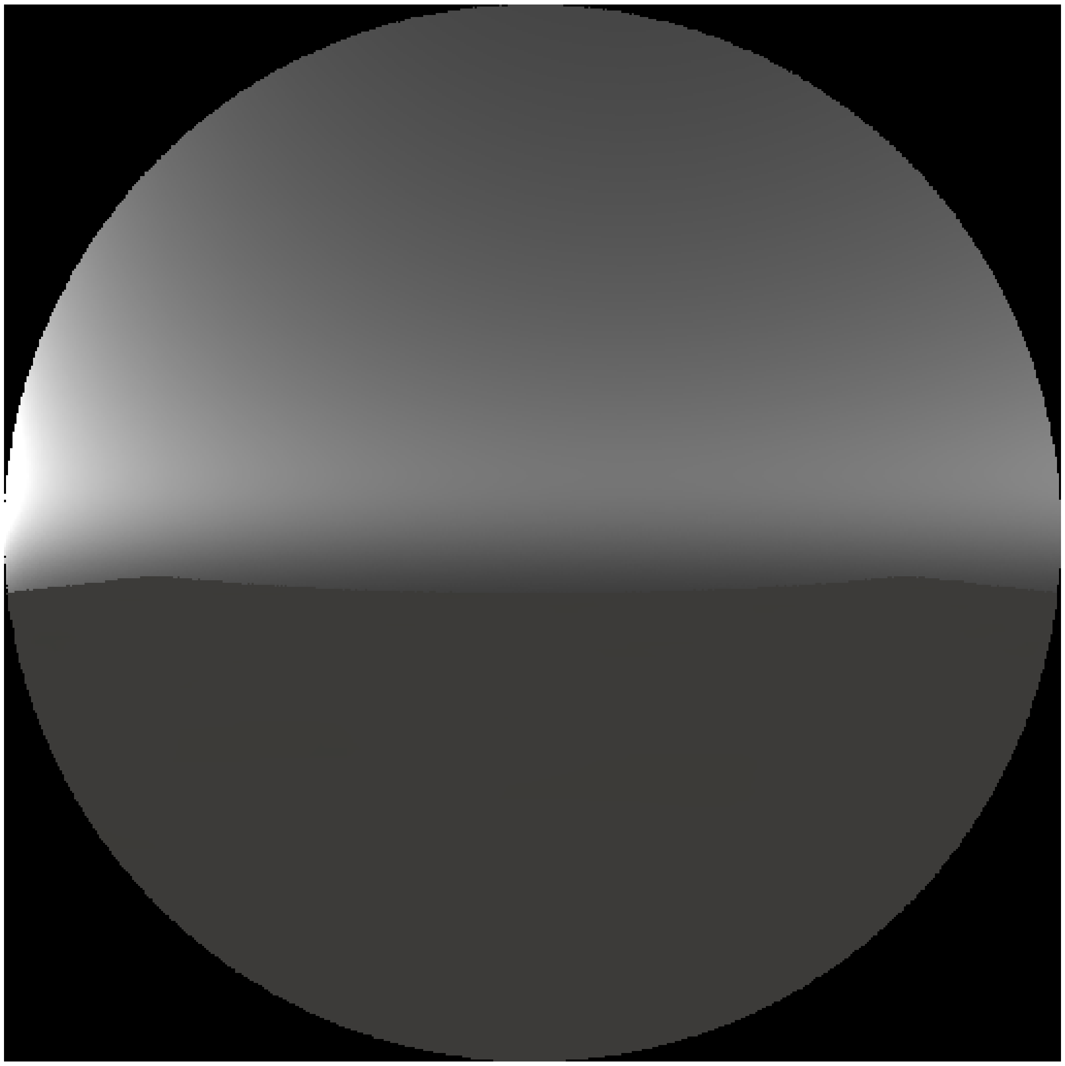


Figure F.4: Rendering looking out of the CFS test aperature, sky 13.

Absolute Input Illuminance per Klems Basis patch

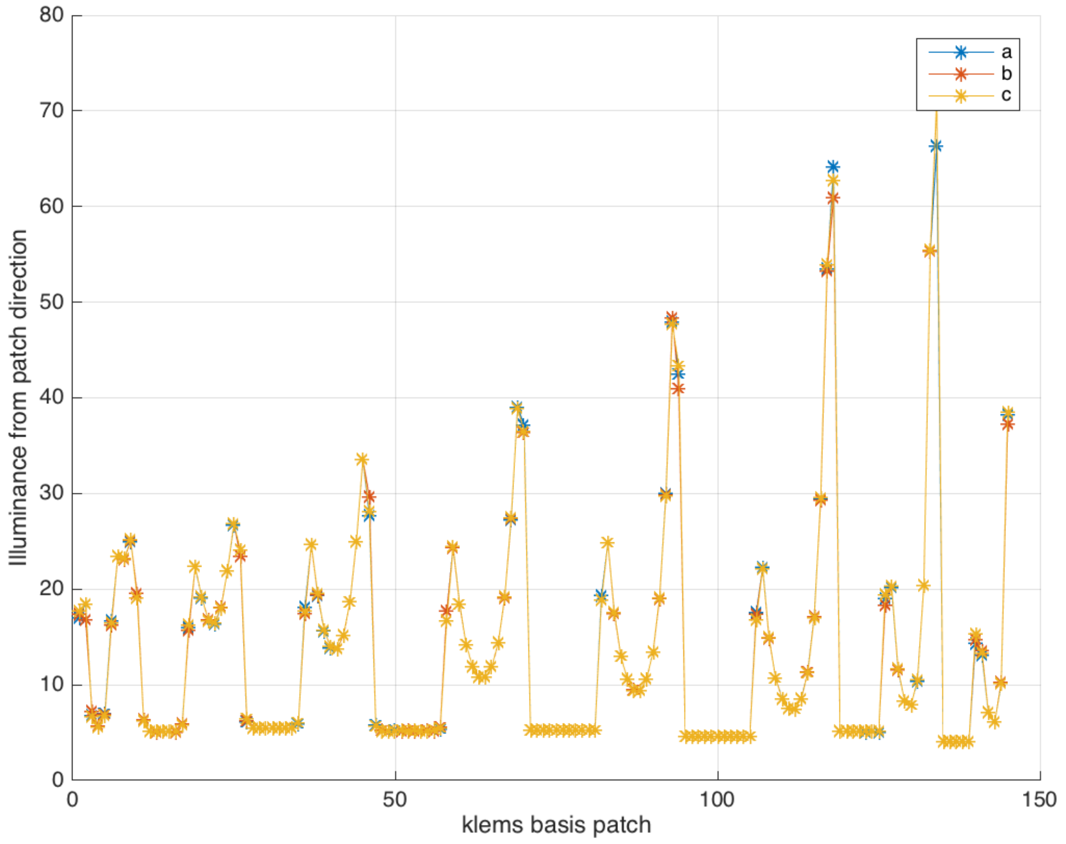


Figure F.5: Absolute input illuminance per Klems Basis patch.

Input Illuminance Error Percentage

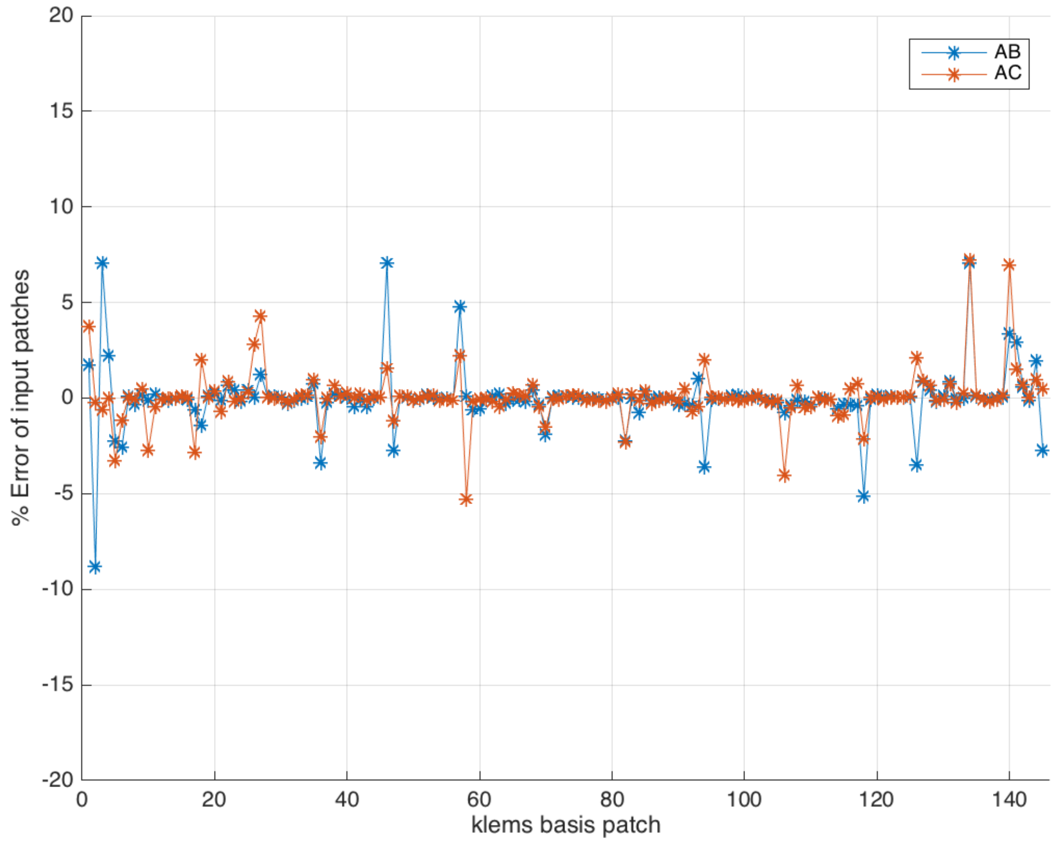


Figure F.6: Error between CFS test aperture and fisheye lens and Li-Cor respectively.

F.1.5 Conclusions

Examining the data in total, the agreement between $L(\theta,\phi)_{cfs}$, $L(\theta,\phi)_{fish}$, and $L(\theta,\phi)_{LI-210}$ is overwall quite good. The main exception comes when a large light source (e.g. the solar disk) or high gradiant luminance sources are near a boundary of the Klems Basis division and the vertical shift causes it to change patches. Examples of this are shown in sky 1 and sky 12. Here the solar disk influences the patches differently by the vertical movement because the solar disk, and associated aruora, shift the majority of their contribution between patches.

For diffuse emitters, the match is particularly good. This is very evident with regard to the plane material, here modeled as a uniform diffuser. Notice the excellent agreement between all pathces "facing" the ground plane. Intuition tells us, however, when dealing with real materials with non uniform diffusion, this will not be quite so good. A fact confirmed below in the measured input investigation.

Also, to reiterate, the distant source, which is the sky vault, is associated with certain patches and they are very consistent, meaning little error exists between them.

The vast majority of errors are below 2%, with most less than 5%, and all errors below 10%, except the two solar disk influences methoned for sky 1 and 12. This is on the order of the error of luminance measurement in laboratory conditions [64], thus it is deemed acceptable here.

There do exist exceptions with the solar disk, or high specular reflection, where errors grow. Given the vast majority of the hemisphere is well representative of $L(\theta,\phi)_{cfs} = L(\theta,\phi)_{fish} = L(\theta,\phi)_{LI-210}$, the simulation analysis adds quantization to the error associated with the offset of the Canon 6D and fisheye lens. Thus, the CUBE 2.0 should avoid direct solar excitation, a condition already planned to be avoided due to the calibration of only diffuse excitation.

F.2 Physical Measurements: Canon 6D and Fisheye Lens

In addition to the simulation analysis, the Canon 6D equiped with the fisheye lens was also used to explore the sensitivity of the vertical displacement in taking real measurements and investigating the binning process.

F.2.1 Description of the Exercise

To test the alignment difference with the actual Canon 6D and fisheye lens, a series of input illuminance measurements were taken. These measurements were taken in sets of two, where one was taken appriximately 14 inches vertically above the other. This height differential is designed to mimic the Canon 6D and the CFS test aperture as on the CUBE 2.0. A two inch difference as compared to the 16 inches on the CUBE 2.0 is due to the fact the tripod used could only extend 14 vertical inches. While not exact, it offeres a very similar situtaion.

F.2.2 Examples of Fisheye 6D Photographs

Two of the image sets (i.e. one “Up” photograph and one “Down” photograph) collected are included here for review:

Measurement One

Absolute Input Illuminance per Klems Basis Patch

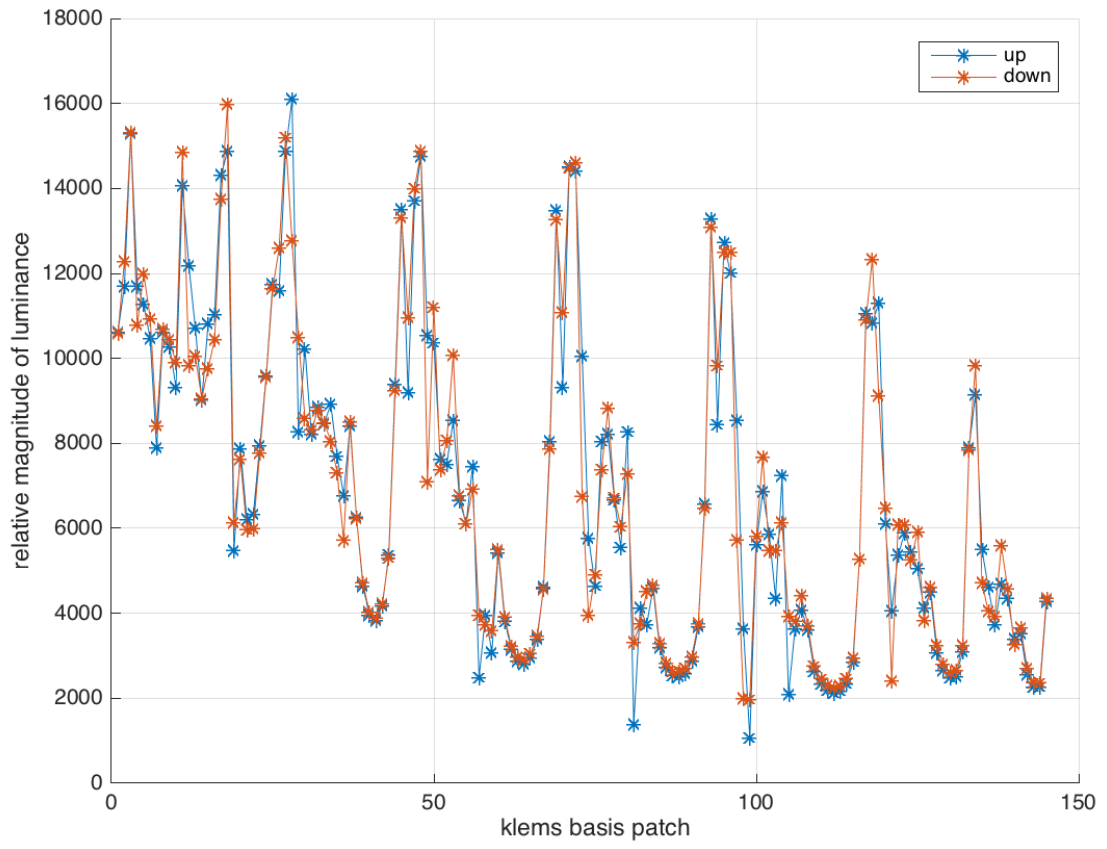


Figure F.7: The absolute input illuminance binned by Klems Basis patches.

Percentage Error Input by Klems Basis Patch

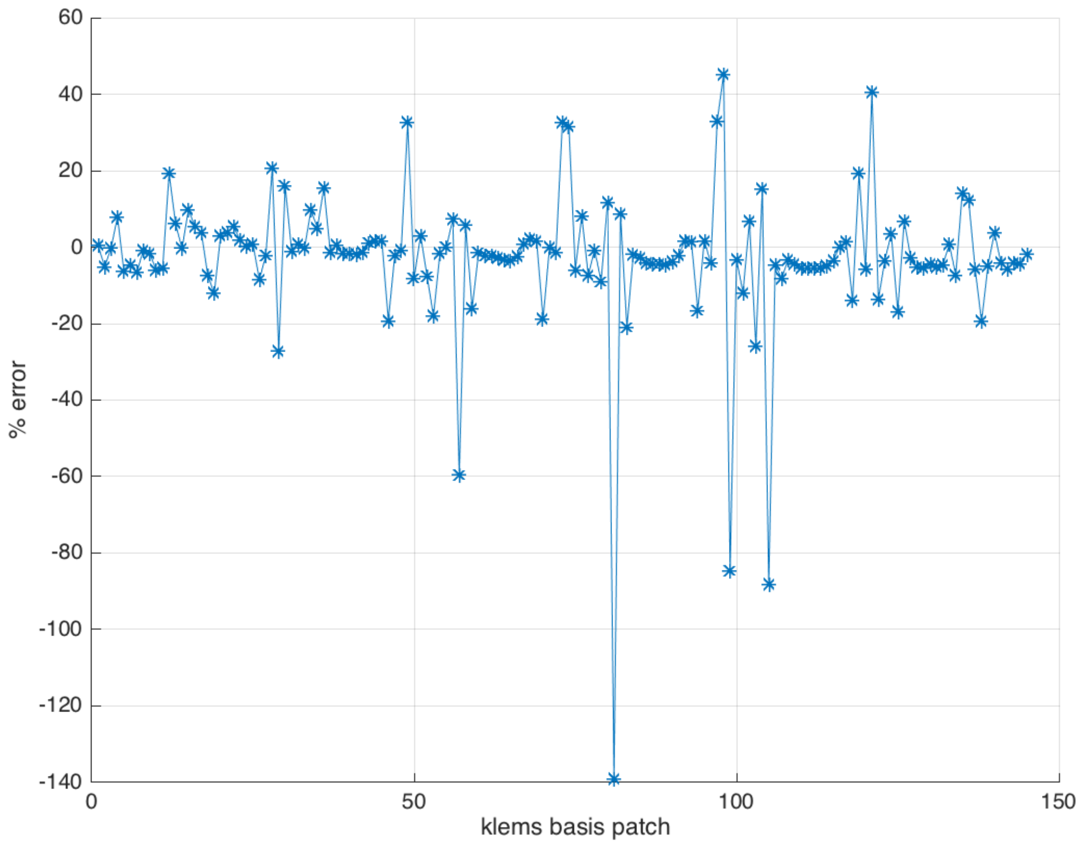


Figure F.8: The percentage error of the “Up” position image versus “Down” position image binned by Klems Basis patches.

Picture of Input Up

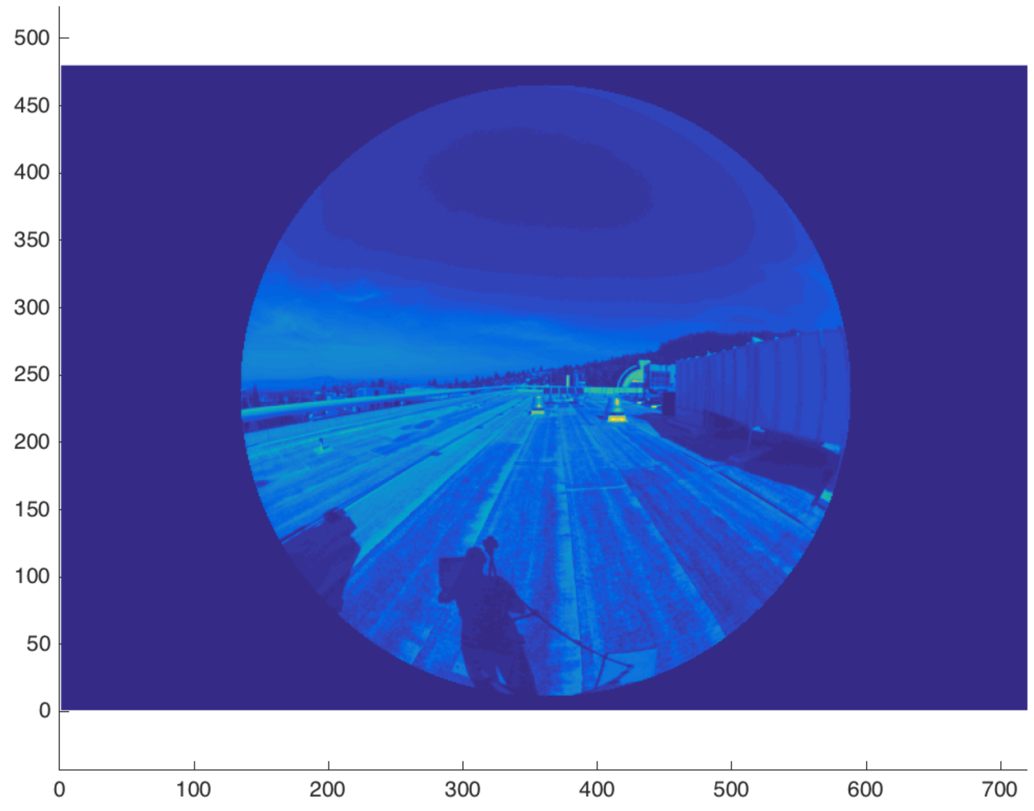


Figure F.9: Full plot of the input luminance as measured by the Canon 6D and fisheye lens in the "Up" position.

Picture of Input Down

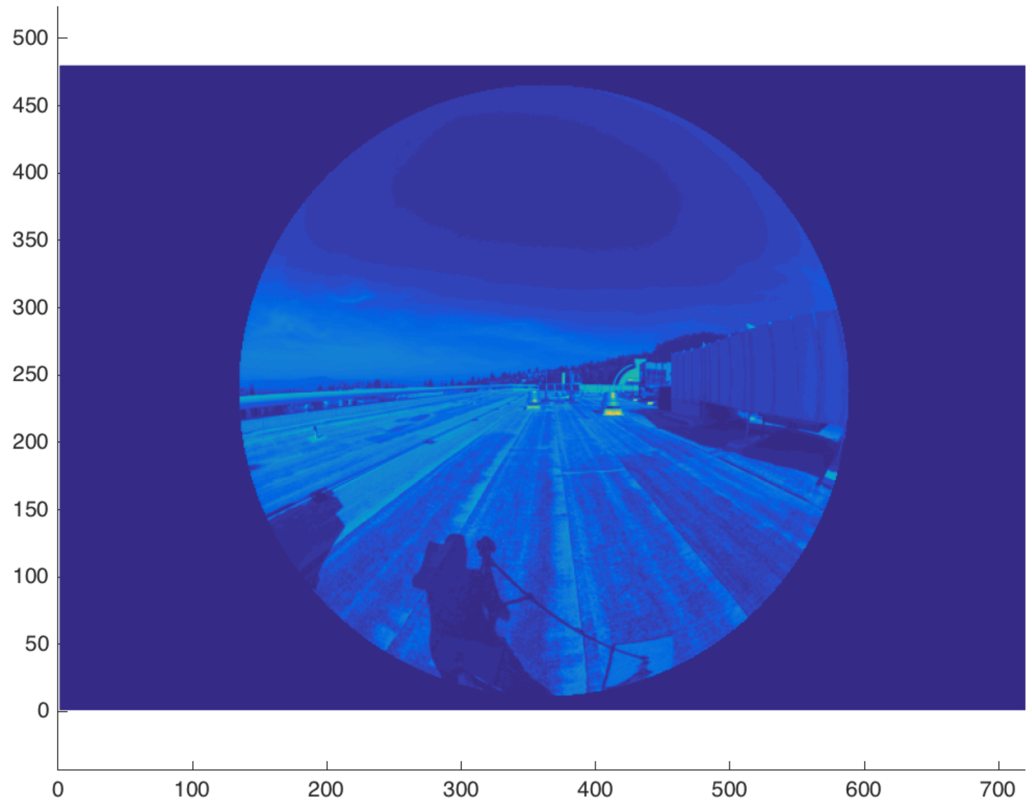


Figure F.10: Full plot of the input luminance as measured by the Canon 6D and fisheye lens in the “Down” position.

Difference of Up and Down

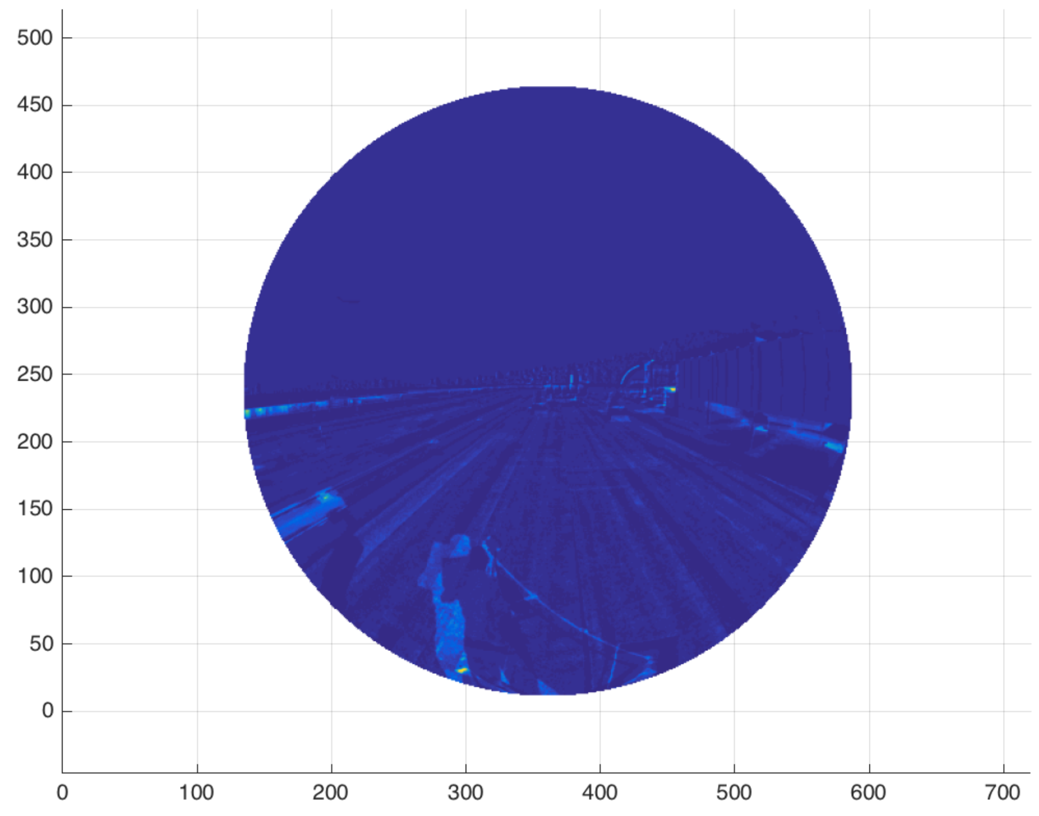


Figure F.11: Full plot of the input luminance difference between “Up” and “Down” as measured by the Canon 6D and fisheye lens.

Klems Basis Patches Overlayed

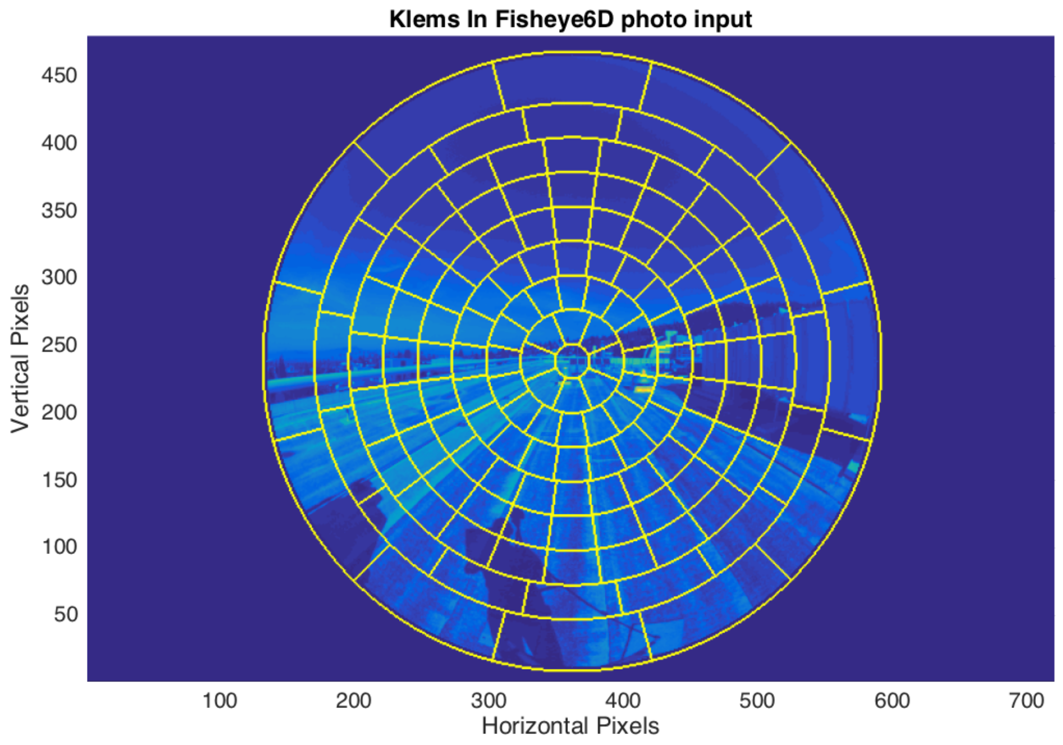


Figure F.12: Full plot of the input luminance as measured by the Canon 6D and fisheye lens overlayed with the Klems Basis patch boundaries.

Measurement Two

Absolute Input Illuminance per Klems Basis Patch

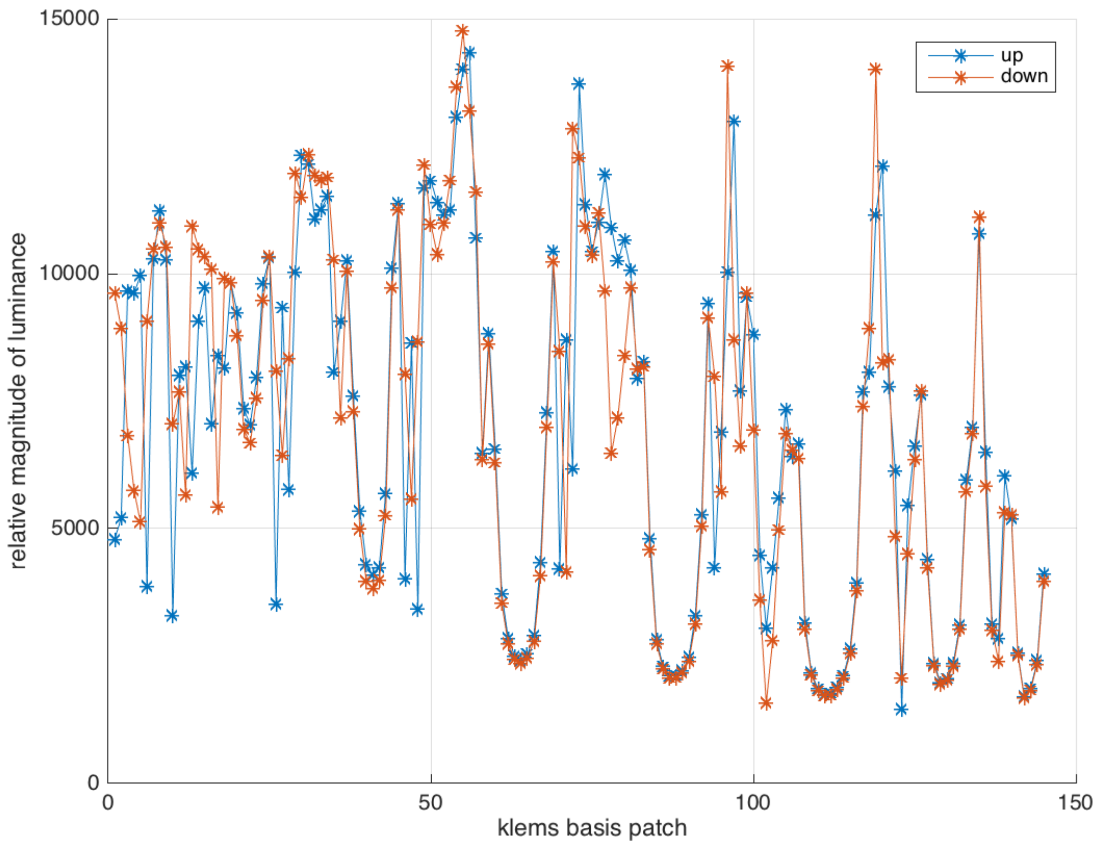


Figure F.13: The absolute input illuminance binned by Klems Basis patches.

Percentage Error Input by Klems Basis Patch

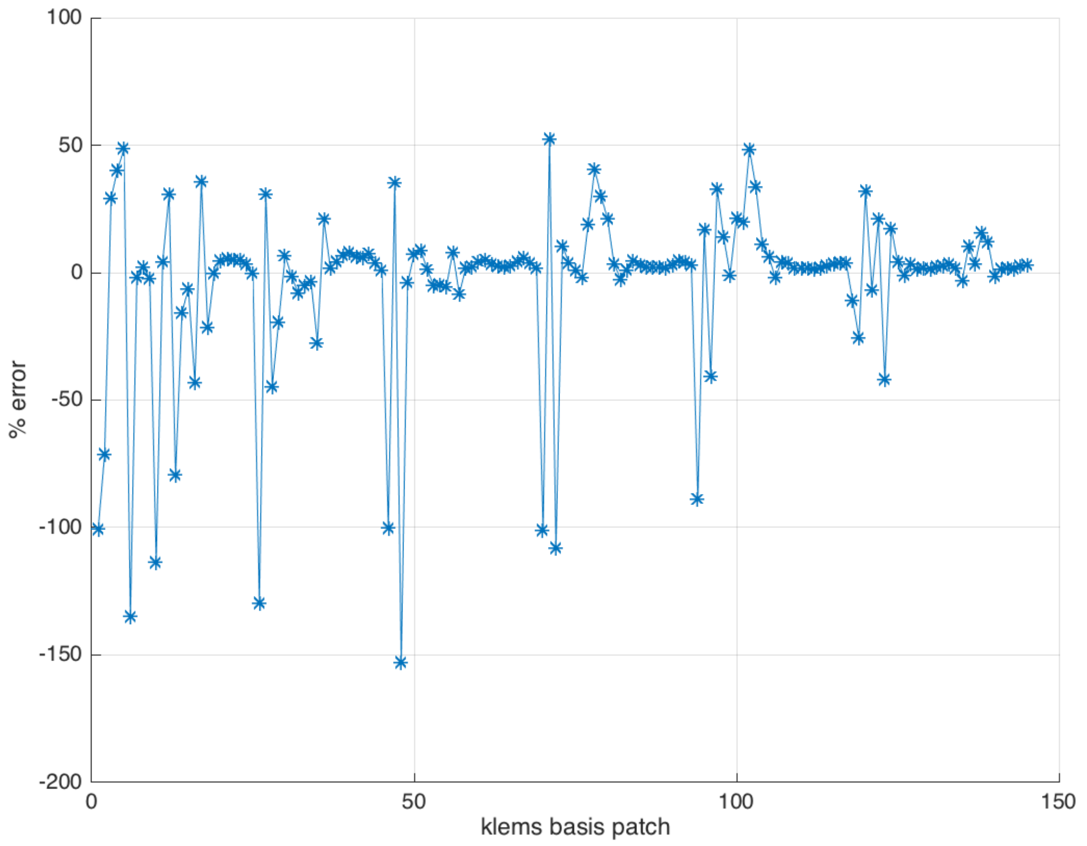


Figure F.14: The percentage error of the “Up” position image versus “Down” position image binned by Klems Basis patches.

Picture of Input Up

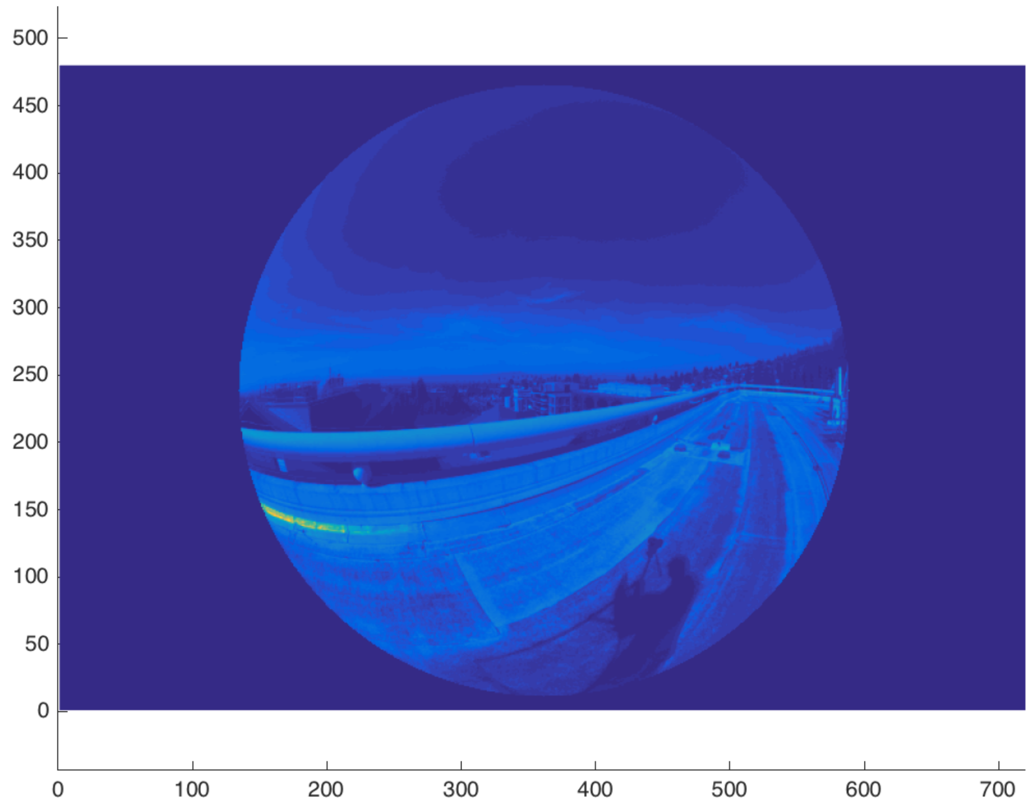


Figure F.15: Full plot of the input luminance as measured by the Canon 6D and fisheye lens in the “Up” position.

Picture of Input Down

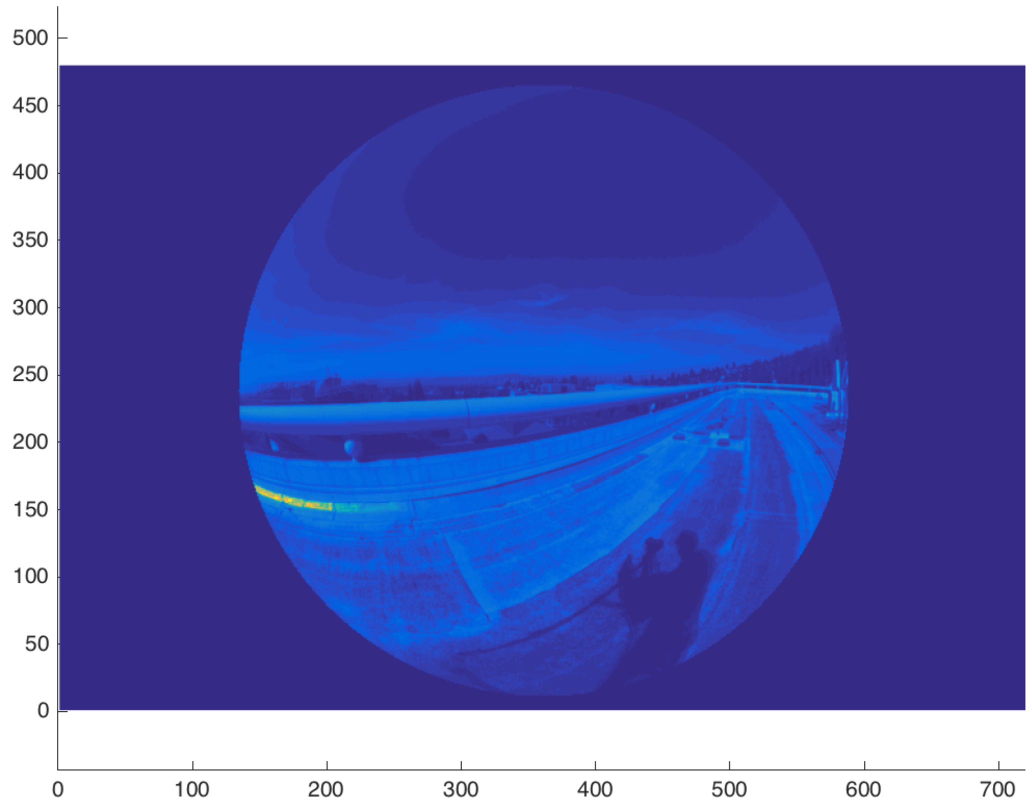


Figure F.16: Full plot of the input luminance as measured by the Canon 6D and fisheye lens in the "Down" position.

Difference of Up and Down

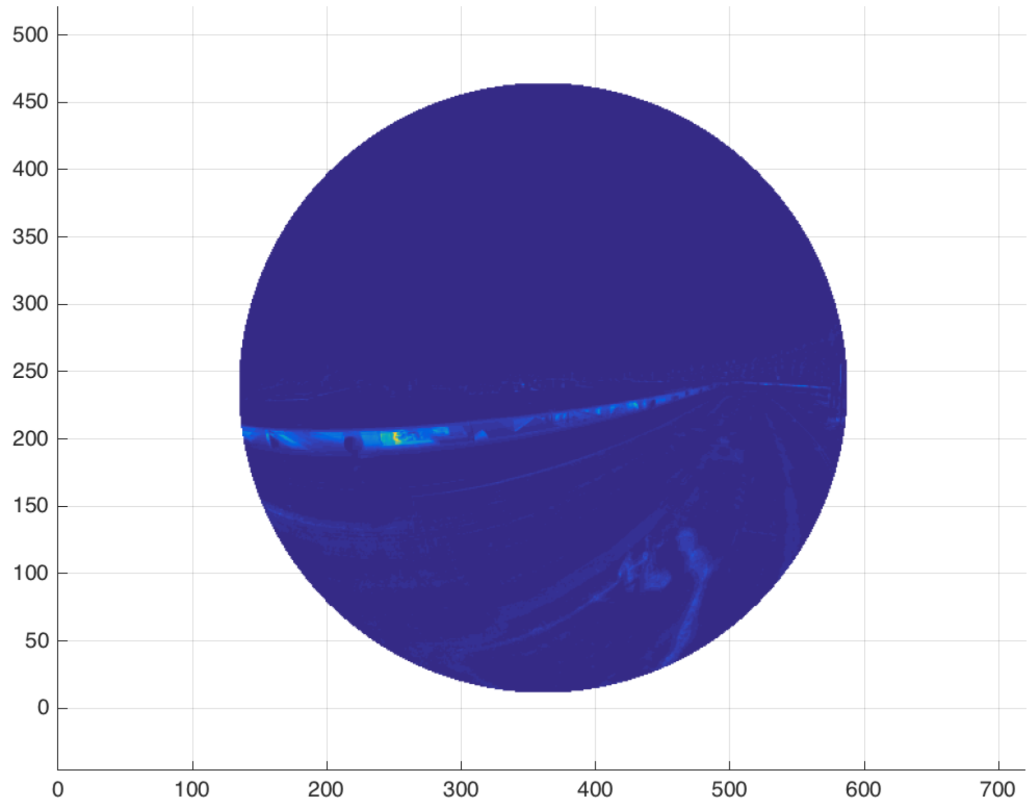


Figure F.17: Full plot of the input luminance difference between “Up” and “Down” as measured by the Canon 6D and fisheye lens.

Klems Basis Patches Overlayed

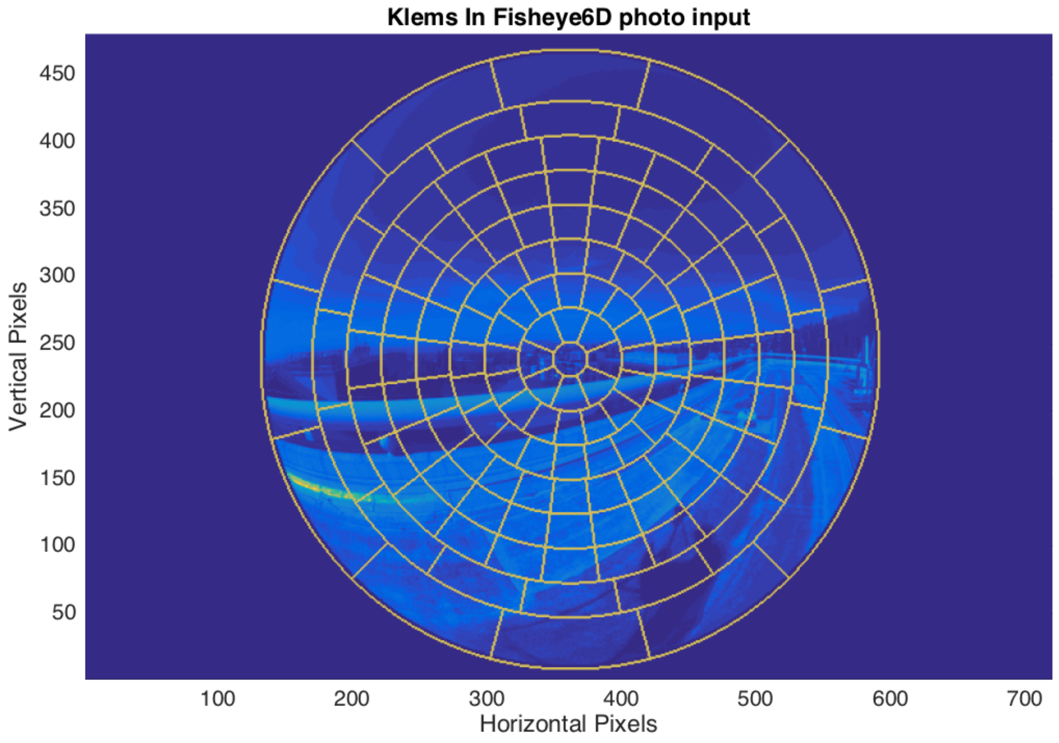


Figure F.18: Full plot of the input luminance as measured by the Canon 6D and fisheye lens overlayed with the Klems Basis patch boundaries.

F.2.3 Conclusions

To begin, one should notice the larger errors associated with the real measurements of the Canon 6D and fisheye lens with respect to the simulated Radiance model. For the first measurement set, the errors, however, are still mostly within typical magnitudes for experimental work. Most are well below 20% with only a few Klems Basis Patches errors rising above 20%. Quite evident, is the very close agreement of the Klems Basis Patches which face the sky. These are very close together, below 5% error. This property will be used in the calibration of the CUBE 2.0 system. The second measurement does have a wider spread of errors, yet careful observation will notice a band of specular reflection. This specular reflection is indeed the same behavior in which the Radiance simulation predicts higher errors with associated parallax errors. That is, these specular reflections change Klems Basis Patch when the Canon 6D and fisheye lens are displaced vertically. This alters the average meaningfully, hence the input error observed. Note, however, the sky patches too match up very well as expected. Finally, one should notice, the consistent qualitative agreement between the two measurements with respect to the overall Klems Basis Patches. This is advantageous, as while the inputs can be off by some factor, their relative qualitative behavior is indeed correct.

F.3 Conclusion

In examining both the simulation and experimental analysis with respect to Canon 6D fisheye lens, LI-210, and CFS test aperature alignment for the Klems Basis divided input illuminance distribution, it is concluded the large majority of klems patches are below 5% difference. This means, $L(\theta,\phi)_{cfs} = L(\theta,\phi)_{fish} = L(\theta,\phi)_{LI-210}$, is true for errors below 5% for the vast majority of Klems Basis Patch cases. With the reported error on the datasheet for the LI-210 illuminance meter at $\pm 5\%$ in mind, this is demed acceptable for the CUBE 2.0. While some patches demonstrate errors above this level, ranging into 20% (and even a small few above this), the general behavior is determined acceptable for the CUBE 2.0 setup. Further, as noted above, the qualitative behavior is very similar between the two inputs, hence adds confidence to at least the relative bahvior of the CFS systems being studied. Considering further the reasoning put forth in the design stage (i.e. no viable other options even exist to measure the input illuminance distribution) the 16 inch virtical displacement proposal is adopted whole heartidly for the CUBE 2.0 system.