RECONFIGURABLE ANALOG CIRCUITS FOR AUTONOMOUS VEHICLES

A Dissertation
Presented to
The Academic Faculty

By

Scott Michael Koziol

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Robotics

School of Electrical and Computer Engineering
Georgia Institute of Technology
December 2013

RECONFIGURABLE ANALOG CIRCUITS FOR AUTONOMOUS VEHICLES

Approved by:

Dr. Jennifer Hasler, Advisor
Professor, School of ECE
Georgia Institute of Technology

Dr. Magnus Egerstedt
Professor, School of ECE
Georgia Institute of Technology

Dr. Fumin Zhang
Associate Professor, School of ECE
Georgia Institute of Technology

Dr. Henrik Christensen
Distinguished Professor, School of Interactive Computing
Georgia Institute of Technology

Dr. Mike Stilman
Assistant Professor, School of Interactive Computing
Georgia Institute of Technology

Date Approved: August 20, 2013

ACKNOWLEDGMENTS

I owe a big thank you to my advisor, my commimttee, lab mates, friends, family, and my God. Proverbs 3:5-6.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Path planning and image processing are critical signal processing tasks for robots, autonomous vehicles, animated characters, etc. The ultimate goal of the path planning problem being addressed in this dissertation is how to use a reconfigurable Analog Very Large Scale Integration (AVLSI) circuit to plan a path for a Micro Aerial Vehicle (MAV) (or similar power constrained ground or sea robot) through an environment in an effort to conserve its limited battery resources. Path planning can be summarized with the following three tasks given that states, actions, an initial state, and a goal state are provided. The robot should: 1) Find a sequence of actions that take the robot from its Initial state to its Goal state. 2) Find actions that take the robot from any state to the Goal state, and 3) Decide the best action for the robot to take now in order to improve its odds of reaching the Goal. Image processing techniques can be used to visually track an object. Segmenting the object from the background is one subtask in this problem. Digital image processing can be very computationally expensive in terms of memory and data manipulation. Path planning and image processing computations are typically executed on digital microprocessors. This dissertation explores an evolution of analog signal processing using Field Programmable Analog Arrays (FPAAs); it describes techniques for mapping different solutions onto the hardware, and it describes the benefits and limitations. The motivation is lower power, more capable solutions that also provide better algorithm performance metrics such as time and space complexity. This may be a significant advantage for MAVs, ocean gliders or other robot applications where the power budget for on-board signal processing is limited.

CHAPTER 1

RECONFIGURABLE ANALOG CIRCUITS FOR PATH PLANNING AND IMAGE PROCESSING

Path planning and image processing are critical signal processing tasks for robots, autonomous vehicles, animated characters, etc. Figure 1 is a cartoon showing the ultimate goal of the path planning problem being addressed in this dissertation, namely how to use a reconfigurable Analog Very Large Scale Integration (AVLSI) circuit to plan a path for a Micro Aerial Vehicle (MAV) (or similar power constrained ground or sea robot) through an environment in an effort to conserve its limited battery resources. Path planning can be summarized with the following three tasks given that states, actions, an initial state, and a goal state are provided. The robot should:

1. Find a sequence of actions that take the robot from its Initial state to its Goal state
2. Find actions that take the robot from *any* state to the Goal state
3. Decide the *best* action for the robot to take now in order to improve its odds of reaching the Goal

Image processing techniques can be used to visually track an object. Segmenting the object from the background is one subtask in this problem. Digital image processing can be very computationally expensive in terms of memory and data manipulation. Path planning and image processing computations are typically executed on digital microprocessors. This dissertation explores an evolution of analog signal processing using Field Programmable Analog Arrays (FPAAs); it describes techniques for mapping different solutions onto the hardware, and it describes the benefits and limitations. The motivation is lower power, more capable solutions that also provide better algorithhm performance metrics such as time and space complexity [7, 2]. This may be a significant advantage for MAVs, ocean gliders or other robot applications where the power budget for Guidance, Navigation, and

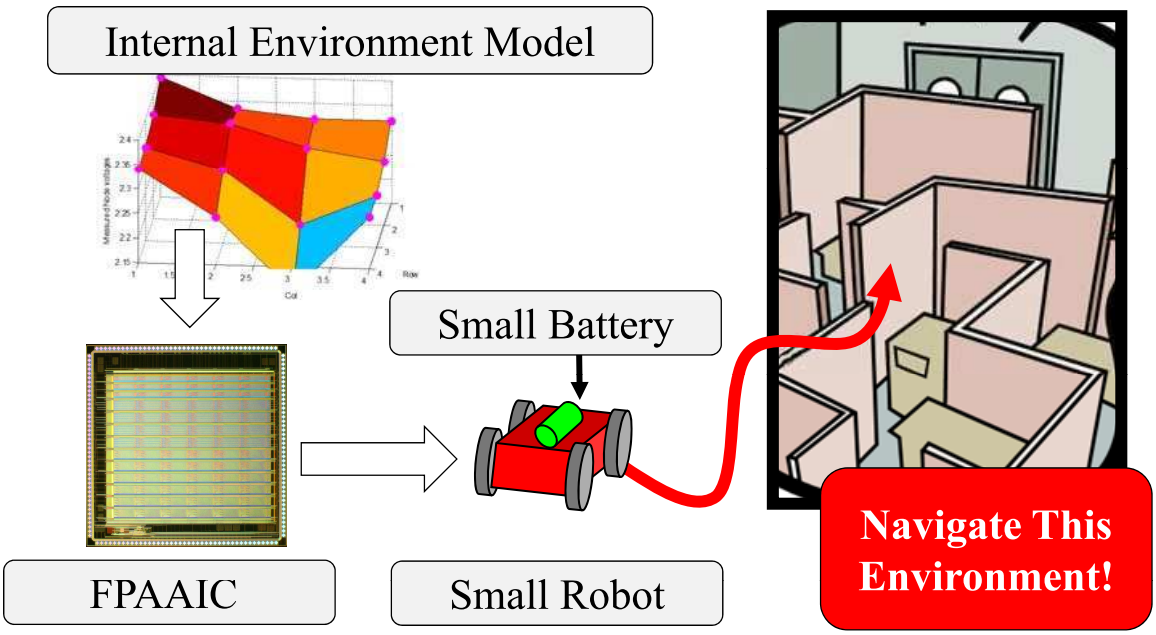


Figure 1. This shows a goal of this research: To use reconfigurable analog circuits called Field Programmable Analog Array Integrated Circuits (FPAA IC) to plan a path for small robot through an environment in an effort to conserve limited battery resources and extend operation times [2].

Control is limited [29, 30].

A custom Application Specific Integrated Circuit (ASIC) could be developed to implement analog path planning, however custom ASIC designs are fixed so any revisions would incur a long design cycle/fabrication time. FPAAs, however, allow the designer or the autonomous agent itself to reconfigure the analog connections within the Integrated Circuit (IC) using software and hardware infrastructure. This allows quick design changes and reuse of a single IC [31, 7]. Chapter 2 describes the FPAA embedded system infrastructure used in this path planning research.

In summary, analog path planning is explored because it represents a potential decrease in time and space complexity, a potential reduction in power needed for computation, and potential decrease of computation time. An FPAA analog path planning implementation is useful because reconfigurable AVLSI systems provide circuit tune-ability and flexibility that custom ASICs do not provide [7, 2].

1.1 The Problem Being Addressed

This dissertation answers the questions:

Can robot path planning and image processing problems be solved using new analog signal processing systems and techniques which are based on floating-gate based reconfigurable analog hardware? If so, what is the performance comparison of these systems and techniques to other computational approaches? How are different solutions mapped onto the hardware, and what are the limitations?

A major part of the "new" aspect of this solution is that it uses floating-gate transistors. Techniques will be described for mapping the algorithms on this floating-gate hardware. Performance comparisons will include complexity analysis of the algorithms which measure solution computation time and memory requirements. Also, computations per energy will be another metric used to assess performance.

1.2 Why This Problem Is Important

There is an energy efficiency wall trend, Figure 2, that has been identified with digital computation. This trend shows that the number of computations per unit of energy has been increasing over time however this trend is slowing down and approaching a line or "Efficiency Wall". If this trend continues it will negatively impact the growth of the computational capability of an all-digital approach to robotic computation [3]. Given this efficiency wall, the significance of this work is twofold: *Low power* and *more capable* hybrid analog-digital solutions will provide longer operation times and more computing capability than currently available for power budget constrained autonomous agents using an all-digital approach. This claim is founded on previous research which shows that reconfigurable analog systems called Field Programmable Analog Arrays (FPAA)s are providing a low-power hardware technology enabling analog computational efficiency of 1000 over digital solutions [32, 15]. This may be a significant advantage for Micro

Aerial Vehicles (MAVs) or other robot applications such as Autonomous Underwater Vehicles (AUVs) where the computational capability and power budget for signal processingis limited [33, 34, 35, 36, 29, 30, 37, 38]. The general goal is illustrated in Figure 3. In this figure, the lower left quadrant represents small, light autonomous systems which are limited in computational capability [39]. The upper right quadrant represents heavy autonomous systems which have the capability to carry a significant amount of computational equipment [40]. The goal is to be able to impact the left side of this graph and give smaller systems more capability.

Computation for robotics often takes on two forms: on-board computing, and cloud computing. There is a heavy movement of robotics computation into the *cloud* (instead of being resident on the robot) [41]. Leveraging computation from the cloud helps, for example, computationally expensive tasks such as image processing [42]. Cloud robotics is also important from a networked robotics point of view as it allows robots to communicate with one another and, among other things, coordinate movements and activities. However, if we want billions and billions of robots then putting a majority of computation in the cloud could be a problem. This is one reason why continuing to increase the capability of on-board computing and enable systems to push past the efficiency wall trend is important.

1.3 Dissertation Contributions

Contributions of this dissertation include the following three areas:

- Floating-gate based *resistive grid structures* can be used for path planning. This method is different from previous resistive grid research which use traditional transistors and active elements to make variable or fixed resistances [43, 44]. Instead, floating-gate transistors provide the ability to weight differently the edges in the map. Floating-gate weighting provides a simple circuit method to implement the weights. Floating-gate based weights also have the feature that implementation is non-volatile (it will hold it's weight even when the power is removed). Three dimensional grids

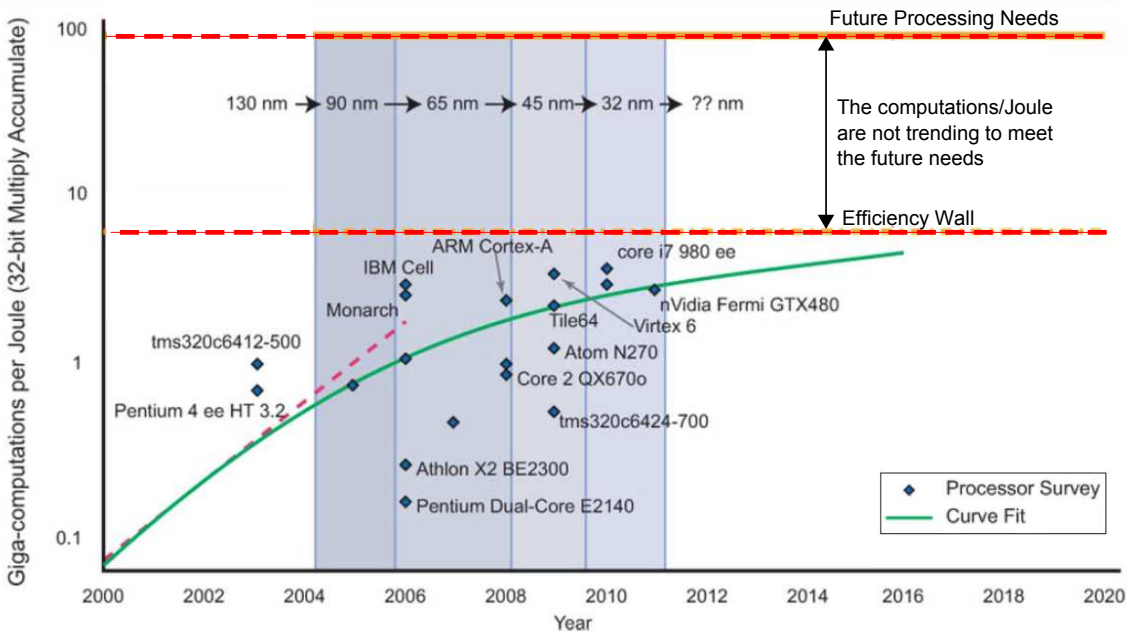


Figure 2. Reproduced from [3], this figure illustrates the details that the rate of improvement in computations per unit energy in processors over the last 30 years is decreasing, and if the trend illustrated in this figure continues it is approaching a "wall."

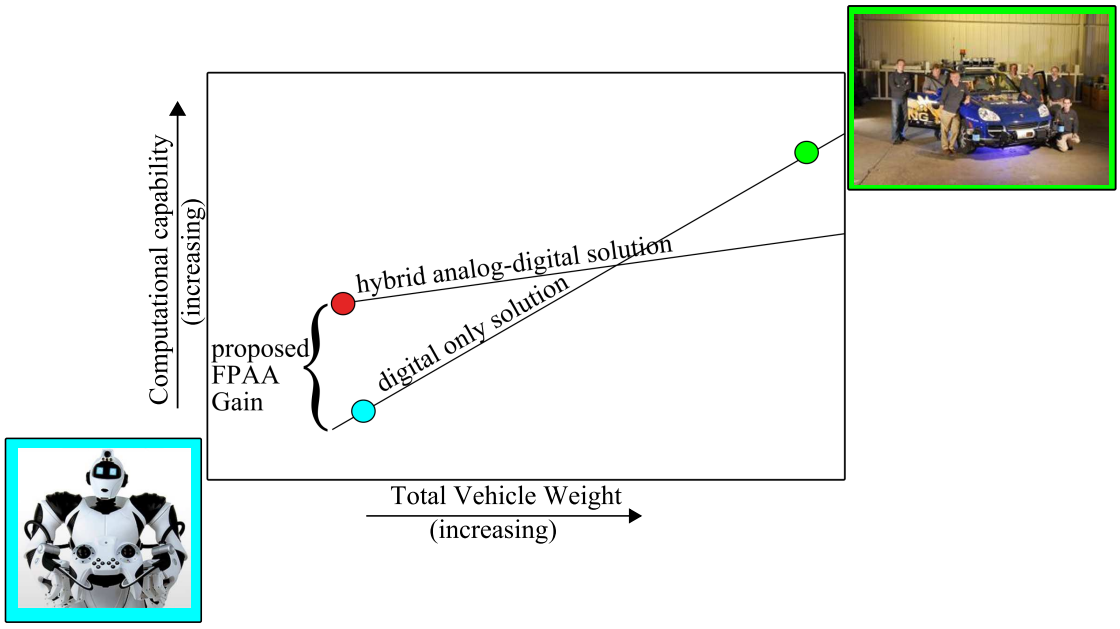


Figure 3. This cartoon depicts the idea of weight vs. computational capability and also shows how we plan to add more computation/gram using a hybrid analog-digital approach.

are also possible with the hardware described in this thesis.

- Floating-gate based *reconfigurable neuron structures* can be used for path planning. Analysis is shown that the parallel computation nature of the system provides a lower Time Complexity metric in certain conditions than typical wavefront planners. One of the items that distinguishes this research from other neural network planners, [45, 46], is that this work shows results and analysis from an actual analog hardware that uses neurons which have realistic dynamics.

- New computational hardware is described which is designed to do subsampled image convolution. Analysis on the analog image convolution technique is provided. Analysis is shown that image processing using analog vector matrix multiplication in this system has an expected capability of approximately 800 Giga to 132000 Giga Computations per Joule. This is a significant improvement over the energy wall trend of approximately 10 Giga Computations per Joule [3].

1.3.1 Design Principals Used in this Dissertation

This thesis applies a couple of computation principals described by Carver Mead [47]. Mead stated that there is a "Factor of 1 million unaccounted for between what it costs to make a transistor work and what is required to do an operation the way we do it in a digital computer [47]." This is largely because in digital systems, energy is spent charging wires and not the gate, and these systems use far more than one transistor to do an operation. A couple of solutions Mead proposes for increasing the efficiency of computation are:

1. Make Algorithms LOCAL (don't ship data all over the place)

2. Use the physics of a device to do the operation instead of using a bunch of AND and OR gates

He further points out that the above approaches require adaptive techniques to correct for differences between nominally identical components.

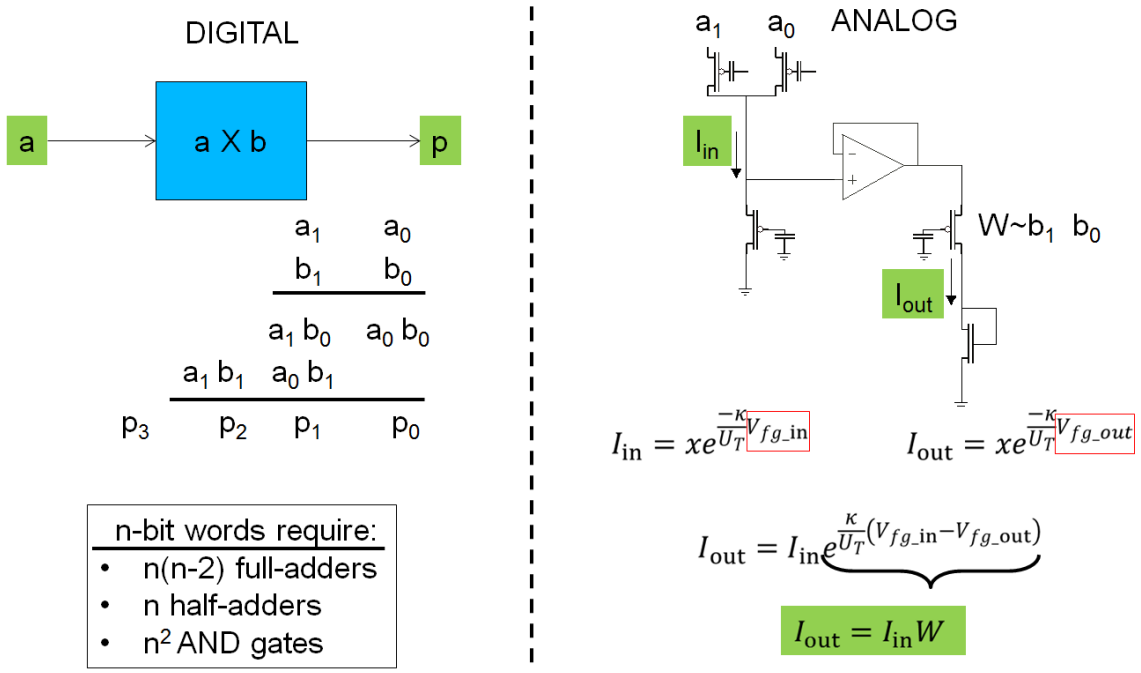


Figure 4. Using the physics of transistors to perform multiplication: Digital multiplication requires hundreds or more transistors to perform a multiplication [4]. Analog can perform a multiplication using about 10 transistors.

1.3.2 Principals in Practice

This dissertation applies the above two principles in the following ways:

1. *Principal*: Make Algorithms LOCAL; *Implementation*: Use analog memory to hold intermediate algorithm computations

2. *Principal*: Use the physics of a device; *Implementation*: Analog vector matrix multiplication and Parallel Processing

Figure 4 and Figure 5 illustrate how the physics of transistors are used to implement vector matrix multiplication with fewer transistors than in digital. Further, floating-gate transistors provide an adaptive technique to correct for differences between nominally identical components in analog processing.

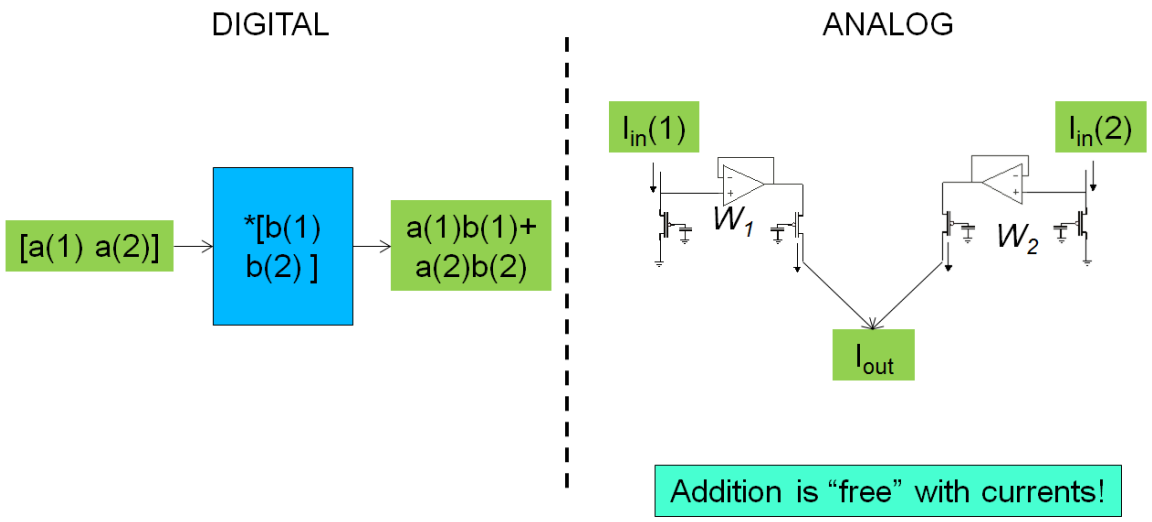


Figure 5. In analog, when the signal is a current, the addition operation of vector matrix multiplication is “free.”

1.4 Related Work

Previous works have used custom Application Specific Integrated Circuits (ASICS) or proposed to use circuits to implement path planning and in analog. Previous analog work in analog path planning includes the following: [48, 49, 50, 44, 51, 52, 53, 54, 55, 56, 57, 45]. A potential fields type path-planning method that can be implemented using parallel AVLSI circuits is found in [48]. Obstacles are represented as non-conducting elements in a conducting medium. The start point is a current source, and the goal is a current sink. The robot’s workspace, conducting, and non-conducting elements are implemented with a resistive grid. Obstacles have a high resistance and non-obstacles have a lower resistance. The path from start to goal is found by 1) placing a constant current source at the start node, 2) waiting until the resistive grid settles into a steady state, 3) reading the node voltages, and 4) finding the path from start to goal using voltage measurements from successive nodes. The authors propose hardware, but only show simulation results. Other AVLSI resistive grid networks used for robot path-planning are found in [44, 58]. The authors use three parallel structures to accomplish the planning. The first layer, the *Memory* layer, contains information about obstacles, the source, and the goal. The second layer, the *Resistive*

Net layer, uses PMOS or NMOS transistors to model resistors. These transistors (and a current source) establish a gradient in the resistive grid. The third layer, the *Comparison Circuit* layer, computes the path by using comparators or a *winner-take-all* circuit to compare node voltages on the resistive grid layer. The authors describe fabricated hardware, but only show simulation results. A couple of United States patent applications have also been identified which deal with analog processing. First, one uses an analog processing integrated circuit which receives obstacle data from an inertial measurement unit (IMU) and an obstacle detection sensor. Analog resistive grids are then used to map obstacles. An electronic device that solves the Laplace's equation is described [59]. Another patent application presents a Laplacian path planner. The system is designed to help autonomous vehicles avoid obstacles while navigating through a series of waypoints [60]. A system is described in [61] that attempts to use the AVLSI planner developed by [44]. Finally, [62] gives an analog robotic motion control scheme developed for obstacle avoidance and target-seeking. According to their paper, "a possible application is in the control of extremely lightweight autonomous machines in the style of Braitenberg Vehicles or Tilden BEAM robots."

A spike based neuron path planner and simulation results are presented in [45]. That work inspired the neuron planning algorithm described in this dissertation. Unlike neural networks which are often trained for classification and pattern recognition tasks [63, 64], the path planner neuron circuitry uses a propagating wavefront to perform the planning operation.

1.5 Other Application Domains

In addition to path planning, analog networks have been proposed for other signal processing tasks such as early vision (which is the problem of converting light into three dimensional shape [65]), surface interpolation [66], edge detection, shape from shading, velocity field estimation, color, and structure from motion (see [67] for this list with additional

citations) [43].

1.6 Dissertation Summary

There are two main drawbacks to using custom ASICs for analog solutions: 1) circuit designs are fixed to some extent (not changeable) and 2) long design cycle/fabrication time (order of months). Reconfigurable analog circuits such as FPAAs have been used to implement a variety of AVLSI circuits in a short time (order of minutes). This allows quick design changes and re-use of a single IC [31]. This work is therefore set apart from previous research in two major ways: 1) the implementation is on a reconfigurable floating-gate based analog platform, and 2) new hybrid analog-digital solutions that exploit the capability of this reconfigurable FPAA IC are developed.

The focus of this dissertation is integrating a reconfigurable analog processor capable of implementing hybrid analog-digital algorithms with a robot for path planning and image processing. Control with reconfigurable analog circuits is also explored. Path planning is often called *Guidance*, which is the process of determining a path for the robot to reach the goal; *Navigation* is determining the robot's state such as position, velocity and attitude; and *Control* is tracking guidance commands while maintaining stability. Figure 6 is a cartoon showing Guidance, Navigation, and Control tasks. Table 6 lists each of these functions and some associated standard methods. The ultimate long term goal of this research is to develop an autonomous micro robot that is capable of autonomous navigation, guidance and control using, in part, analog signal processing systems.

The complete Analog-Digital Hardware/Software/Robot co-design problem is considered. Multiple analog-digital embedded systems have been developed in our lab at Georgia Tech, and these are the initial computation platforms on which the hybrid analog-digital algorithms are developed and executed [68]. These embedded systems are also the foundation upon which more specific hardware is developed. At the heart of the embedded systems

	Function	Answers these Agent's Questions	Standard Method
Guidance	Path Planning	What roads should I take?	A* (A-star)
Navigation	Determining a robot's state	Where am I now?	Kalman Filter
Control	Tracks guidance commands while maintaining stability	How do I adjust the acceleration and steering?	Proportional, Integral, Derivative (PID) controller

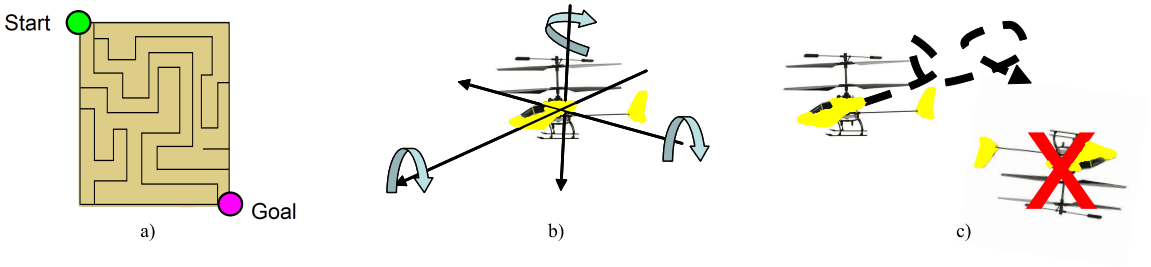


Figure 6. a) Guidance: The *high level* process of planning a path from the Start to the Goal. b) Navigation: Agent uses sensor input to determine its state (Position, Velocity, Acceleration). c) Control: The *low level* process of tracking guidance commands while maintaining stability.

is a reconfigurable analog integrated circuit (IC) called a Field Programmable Analog Array (FPAA) [31]. This custom IC, developed by the Integrated Computational Electronics (ICE) Lab at Georgia Tech, allows a user to configure circuits using basic analog component primitives (such as transistors, amplifiers, and capacitors). Much like digital's Field Programmable Gate Array (FPGA's), these FPAAs can be erased and re-programmed using a software interface. More detailed information about FPAAs and the embedded system is found in Chapter 2. A major thrust of this research is the design of hybrid analog-digital circuits that are equivalent (in function, not necessarily form) to their all digital counterparts.

Algorithm development has focused on performing path planning and image processing for a mobile robot. Previous work in combining digital and analog solutions are found in [69, 70, 71]. The overall goal for path planning is found in Figure 6a. In this cartoon a power constrained mobile robot uses an FPAA or other reconfigurable IC, programmed

with a model of the environment, to navigate a maze or an office environment. This dissertation demonstrates two hybrid analog-digital solutions for path planning. First, Chapter 3 shows that FPAAs can be used for Guidance (Path Planning) using a resistive grid based potential fields type approach [2, 7, 72]. Second, Chapter 4, shows hardware results for a wavefront path planning algorithm using a neuron array integrated circuit. These two approaches are described in the following sections. Figure 6c shows the overall goal of *control* is for the robot to track the guidance commands while maintaining stability. This is considered a low level control task.

1.7 Chapter Summaries

Chapter 2 presents three hardware and software infrastructures used with a family of floating-gate based FPAAs being developed at Georgia Tech. These compact and portable systems provide the user with a comprehensive set of tools for custom analog circuit design and implementation. The infrastructure includes the FPAA IC, microcontrollers for interfacing the FPAA with the user's computer, and Matlab and targeting software. The FPAA hardware can communicate with Matlab over a USB connection. When connected to a computer, the USB connection also provides the FPAA hardware's power. Some of the software tools include three major systems: a Matlab Simulink FPAA program, a SPICE to FPAA compiler called GRASPER, and a visualization tool called RAT. This chapter also presents a description of a floating-gate transistor because this is the key enabling technology that allows the FPAA to program arbitrary circuits (and also implement resistive elements).

Chapter 3 presents path planning using resistive grids implemented on two different FPAA ICs: the RASP 2.8a and the RASP 2.9V [31, 73]. The resistive grids elements are implemented with floating-gate transistors. The general idea is similar to the well-known potential field approach to path planning [74] in that the robot's location is the high point of an energy surface and the goal is at the low point and the path to goal follows the gradient. This chapter presents hardware results using reconfigurable AVLSI circuits developed at

Georgia Tech to implement a path planning algorithm. Experimental results are presented for a large number of environment scenarios. Also, an experimental result of interfacing the FPAA with a Pioneer robot is described.

Chapter 4 presents hardware results for a neuromorphic approach to path planning using a neuron array IC. The algorithm is explained and experimental results are presented showing 100% correct and optimal performance for a large number of randomized maze environment scenarios. Based on neuron signal propagation speed, neuron IC path planning may offer a computational advantage over state-of-the-art wavefront planners implemented on FPGAs. Analytical Time and Space complexity metrics are developed in this chapter for a Neuron IC planner, and these are verified against experimental data. Optimality and Completeness are also addressed. The neuron structure allows one to develop sophisticated graphs with varied edge weights between nodes of the grid. Two interesting cases are presented. First, asymmetric edge costs are assigned to describe cases which have a certain cost to travel a path in one direction, but a different cost to travel the same path but in the opposite direction. The application of this feature can translate to real world problems involving hills, traffic patterns, etc. Second, cases are presented where the nodes near an obstacle are given higher costs to visit these nodes. This is in an effort to keep the autonomous agent at a safe distance from obstacles. This grid weighting can also be used to differentiate among terrains such as sand, ice, gravel, or smooth pavement. Experimental results are presented for both cases.

Chapter 5 presents results of a mobile manipulator robot tasked to play the classic Towers of Hanoi game. First, the control algorithms necessary to enable necessary game-playing behavior are discussed and results are provided of implementing the methodology in a high fidelity 3D environment. After attaining success in the simulation environment, results are shown on implementation of the same control software using physical robot hardware. Additionally, analysis for implementing analog Proportional-Integral-Derivative

(PID) control on this platform using a floating-gate based reconfigurable analog IC is explored. Using this concept of floating-gate analog arrays for control enables off-loading of the processing, which could be helpful for real-time implementation of robot behavior.

Chapter 6 first describes an analog image processing algorithm which uses floating-gate transistors to implement multiplication weights. Further, analog storage elements are described as holding places for intermediate values in the computation. Two hardware systems being developed on which to implement this hybrid analog-digital approach to image processing are also described. One of these systems is based on the RASP 2.9v IC, and the second is based on the RASP 3.0 IC. Each has its own PCB embedded system which is also described.

Chapter 7 provides chapter summaries and current and future possible research using Hidden Markov Models and dendrite classifiers.

CHAPTER 2

FPAAS FOR EMBEDDED SIGNAL PROCESSING

The two main electronic hardware components used in the signal processing systems in this dissertation are the FPAA ICs and the embedded system into which the FPAAs are integrated. FPAA technology, while continuing to develop, is sufficiently advanced at this time to allow one to perform algorithm development and demonstration. This chapter will describe the FPAA ICs, embedded system, and will also provide information about floating-gate transistors because these unique devices, among other things, are a key enabling technology for FPAAs to be able to reconfigure their internal wiring.

2.1 FPAA Background and Foundational Material

Field Programmable Analog Arrays (FPAAs) are useful for both research and teaching [75]. Previous Georgia Tech FPAA ICs have been programmed using a self-contained development platform that included a commercial FPGA development board, a custom FPAA board, and a AC-DC power module. This previous hardware platform [76] fit into an enclosure about the size of a shoe box and communicated with the computer using Ethernet. The new platforms described in this chapter are significantly smaller and communicate over USB. The power supply systems have also been changed and improve upon the portability. The infrastructure is a proven system as it has seen use in a DARPA project workshop at the University of Southern California, two workshops in Telluride, CO, and two more workshops at Georgia Tech. The boards have also been the primary laboratory infrastructure for two semesters of a graduate course in Neuromorphic Analog VLSI Circuits at Georgia Tech.

FPAAs are the main enabling technology for this Analog-Digital robotic signal processing. They provide a low-power, low-cost, reconfigurable, and reusable hardware technology, while still enabling analog computational efficiency of 1000 over digital solutions

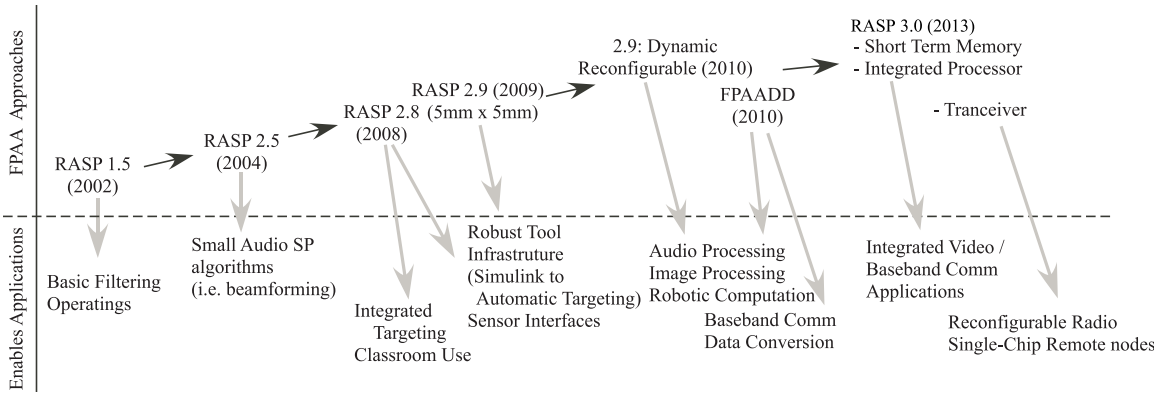


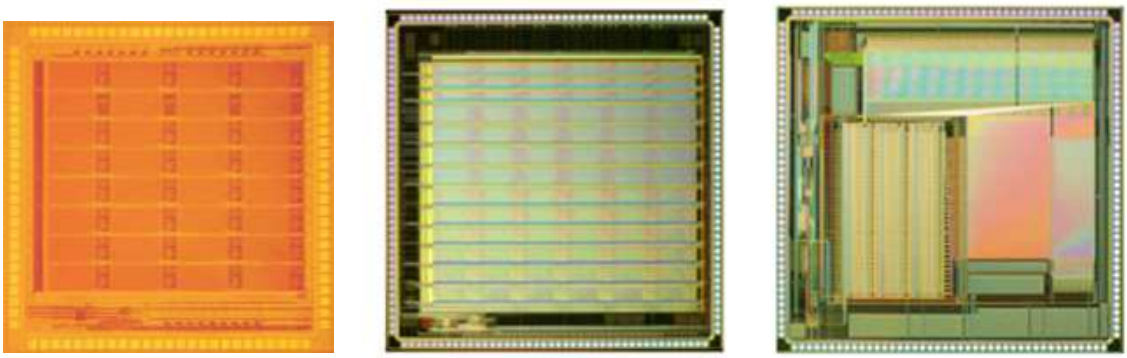
Figure 7. Roadmap of the current and estimated future GT FPAA IC development [5].

[15]. Figure 7 shows the timeline of many FPAA devices [5]. The FPAAs, developed at Georgia Tech, are based on the Reconfigurable Analog Signal Processor (RASP) family of ICs. Each generation of ICs enabled a higher level of computation and a wider range of applications. The ICs include the RASP 1.5 [77], RASP 2.5 [78], RASP 2.8 [79, 80], RASP 2.9, the recent digitally enhanced ICs RASP 2.9v: Dynamic Reconfigurable [73], FPAAD [81], and combination FPAA/microprocessor RASP 3.0. Further FPAA results can be found in [82, 83, 84, 85, 86, 87, 88, 89].

Section 2.2 discusses the basics of FPAAs as well as the embedded hardware interface, Section 2.3 discusses software tools, Section 2.4 shows the design flow using a demonstration test circuit, and Section 2.6 is a closing summary.

2.2 FPAA Hardware Infrastructure

This work uses four instances of the RASP family of ICs: RASP 2.8a, RASP 2.9v, Neuron IC, and RASP 3.0. Figure 8 shows three of these FPAA ICs. The RASP 2.8a IC is 3x3 mm in size and arranged in a 4x8 array of computational analog blocks (CAB)s [31] The RASP 2.9v IC is 5x5 mm in size and arranged in a 6x13 array of CABs [90, 73].The Neuron IC is 5x5 mm in size and arranged with 100 neurons and 30,000 synapses [10].



a) RASP2.8a b) RASP2.9v c) Neuron

Figure 8. Research in this dissertation was based on the following FPAA ICs: a) RASP 2.8a used for path planning and control, b) RASP 2.9v used for path planning, and c) Neuron IC used for path planning.

These reconfigurable analog platforms utilize a switch matrix of programmable floating-gate transistors as switch elements. These switch elements have a dual role as computational elements [91]. This specific feature is exploited in this work. The reconfigurable nature of the platforms allow rapid building and testing of different circuit configurations [31, 79].The arrays have a mixture of analog granularity, so that one has access to transistor-level functions, as well as some higher signal processing features. Programmable floating-gate circuit technology enables the FPAAs to provide area-efficient, accurately programmable analog circuitry that can be easily integrated into a larger digital/mixed-signal system [90, 76]. A closed loop control system is used to program the floating-gate elements [92]. Some floating-gate switches may be *fully on* whereas others may be programmed to specific currents. In our present implementation, Matlab interfaces with the control board's ARM Core microprocessor to implement the control algorithm. In the future, this control algorithm can be moved entirely to the ARM core microprocessor. The FPAA ICs contain on-chip programming structures which measure the level to which the floating-gates are programmed.

The main embedded hardware developed consists of a family of five PCB designs which include the following:

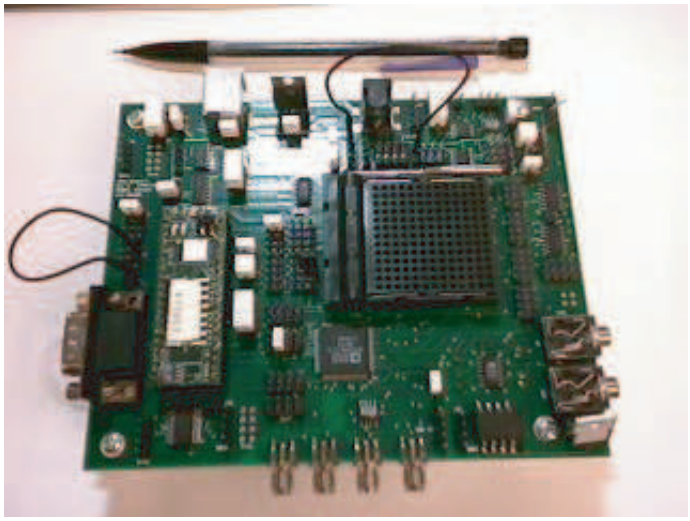


Figure 9. FPAA Programming and Control Board (25.76 square inches). Note the USB connection on the top left; 40 pin DIP microcontroller module to the right of the serial connector; the 100 pin ZIF socket for inserting the FPAA ICs; many 2x4 pin headers connected to FPAA I/O, DAC outputs, ADC inputs, FPAA control pins, and power/ground; 4 SMA for FPAA I/O interface; and the audio jacks (on the lower right).

1. Research-oriented FPAA board which allows maximum access to and control of the FPAA pins

2. Class-oriented board which is a scaled down version of the research board

3. Stand-alone board for integrating the FPAA into robots and other systems

4. Modular board for integrating the FPAA into robots and other systems

5. FPAA/Microcontroller interface board to support the new RASP3.0 ICs

The first three embedded systems are described in the following sections, and the last two are described in Chapter 6.

2.2.1 Research Board

The Research Board, Fig 9, was for many years the workhorse of our FPAA infrastructure family. It has header pins which allow easy access to most of the FPAA pins. This is helpful for many things including power measurements, circuit debugging, etc. The research board has a 100 pin zero insertion force (ZIF) socket into which the FPAA IC is placed. This

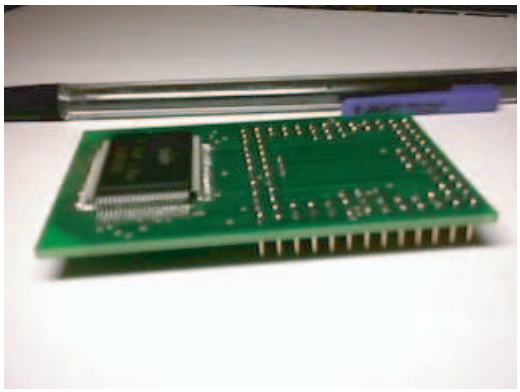


Figure 10. Adaptor Board (4.16 square inches). This custom PCB has a quad flat pack (QFP) packaged FPAA IC on one side and pins on the other side. The pins plug into the 100 pin ZIF socket on the programming and control board.

socket makes the Research Board a good general platform for testing many families of FPAAs such as our General, Sensor, Bio, Mite, and Adaptive versions. The FPAAs are typically packaged in plastic surface mount packages. We have developed adaptor board PCBs which convert the surface mount packages to pins, Fig. 10. These pins then plug into the ZIF socket on the programming and control board.

The programming and control board has the following features: USB or Serial communication capabilities, USB power or external DC power, SMA connectors for connecting to FPAA I/O pins, a discrete 14-Bit DAC IC which has forty channels (most of which can be used as inputs to FPAA I/O pins), a discrete 8-bit ADC that can also be used to connect to FPAA I/O pins, amplifiers to be used as I/O buffers, and finally an audio amplifier and audio jacks which can be used for audio input and output connections to the FPAA. The board also has 3.3V, 5V, and 12V supplies. The board uses an Atmel AT91SAM7S ARM based microcontroller to communicate via USB to a desktop or laptop computer. The software emulates a serial communications device class (CDC) connection, and most modern operating systems have drivers for this software out-of-the-box. The ARM Core microprocessor on the board was purchased as a 40 pin DIP plug-in module. A block diagram of the system is found in Figure 11.

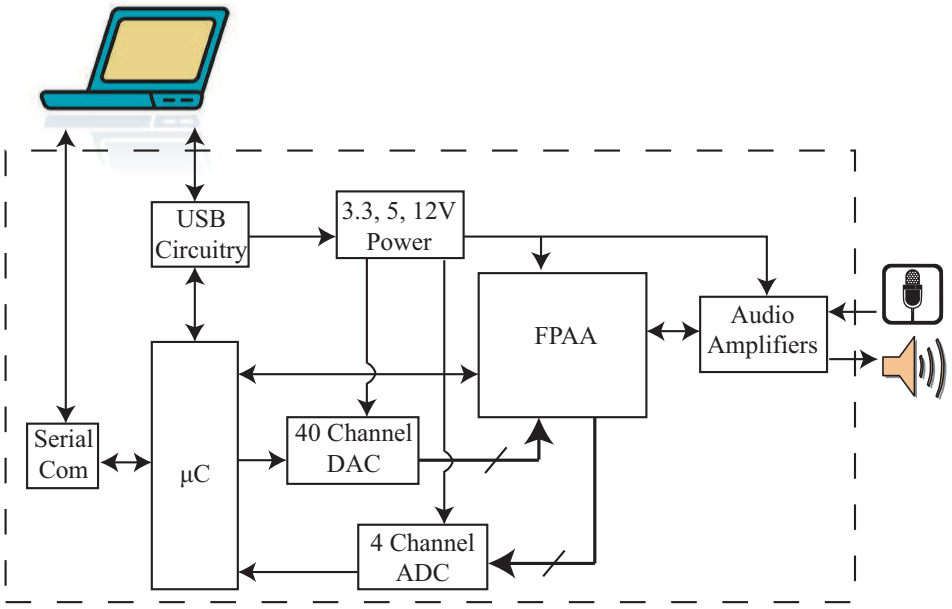


Figure 11. Block Diagram of the FPAA programming and control board. The board has been designed to be self-contained and portable, only needing a laptop. The user chooses between USB or serial communication. The power is supplied by the USB port. The microcontroller (μC) is a 40 pin DIP plug-in module which uses an ATMEL 32 Bit ARM processor. The FPAA I/O can be reconfigurably connected to the discrete ADC and DACs using headers and jumpers. MP3 players can easily be used as inputs to the FPAA by using the audio input port and audio amplifiers.

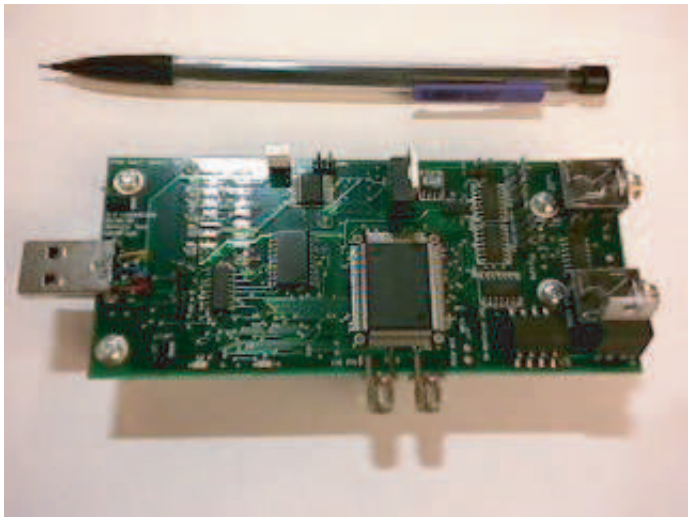


Figure 12. Class-oriented Board (8.74 square inches). Note the USB connection on the left for direct connection with the user's computer. The FPAA is the large square IC in the center above the two SMA jacks. The audio interface jacks are on the right.

2.2.2 Class-oriented Board

The small Class-oriented board, Fig 12, is basically a subset of the Research Board. It was developed with three main goals in mind: 1) easy to learn and use, 2) inexpensive, and 3) small. The easy-to-learn-and-use goal was accomplished by removing as many of the "intimidating header pins" as possible. To make this possible, many of the FPAA I/O pins that are connected to headers on the Research Board are now hardwired to ADC or DAC pins. Only a small number of I/O are pinned out to headers. There are thirty-two header pins on the Research Board that are considered "factory settings" and are normally jumpered to certain pins. These pins were hardwired to their respective signals. The goal of reducing the cost was accomplished by reducing part count and changing some parts. For example, the discrete ADC IC was removed in favor of using the ARM Core microprocessor's onboard ADC, and the forty channel DAC was replaced with a less expensive eight channel DAC. The ARM Core microprocessor on the Research Board was purchased as a 40 pin DIP plug-in module. To save cost, the class-oriented board has the microprocessor IC and associated circuitry integrated directly onto the FPAA board. The cost goal was accomplished. The parts alone for the class-oriented board are more than fifty percent cheaper than the Research Board ($100 vs. $230). The size goal was accomplished too as it is sixty-six percent smaller than the Research Board.

2.2.3 Stand-alone Board

The stand-alone board, Fig 13, was developed as a way to separate the FPAA from the programming infrastructure so that an FPAA can easily be integrated into robotic and other systems. The stand-alone board consists of a surface mounted RASP 2.8 FPAA IC, power circuitry, a header used for programming, and connections to FPAA I/O pins. The board also has an IC footprint for a motor driver circuit.

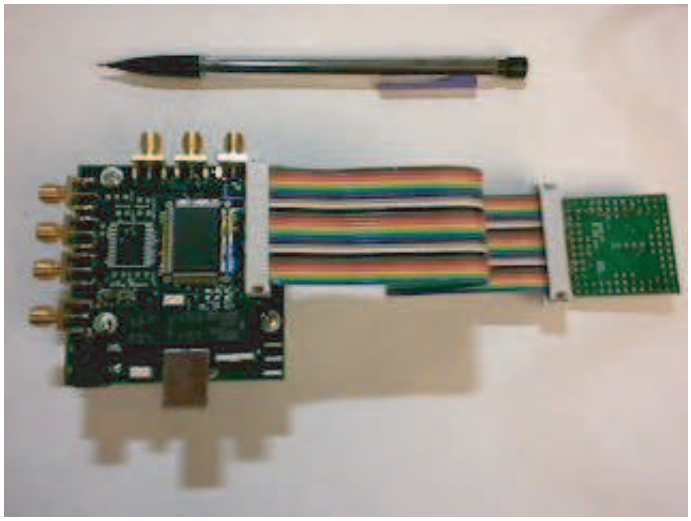


Figure 13. Stand-alone Board on the left (6.2 square inches). Note the removable ribbon cable plugged into the stand-alone board's header. The other end of the ribbon cable plugs into a Research Board's ZIF socket as a means of programming the stand-alone board.

2.3 FPAA Software Infrastructure

The pre-existing FPAA software tools include a Matlab Simulink FPAA program, a SPICE to FPAA compiler called GRASPER (Generic Reconfigurable Array Specification & Programming Environment), and a visualization tool called the RAT (Routing Analysis Tool). Two new software tools were developed to support this work. These are 1) a tool to convert the obstacle map into a transistor map on the FPAA and 2) a map/obstacle visualization and modification program called the Path RAT.

Regarding the existing tools, the Matlab Simulink Tool is an automation tool which converts Simulink models to a SPICE netlist, which can then be automatically compiled to FPAA targeting code and implemented on an FPAA [93]. The GRASPER tool converts a circuit's SPICE file into a list of FPAA switches that implement the circuit on the FPAA [94, 95, 96, 97]. The RAT is a Matlab GUI which graphically shows the topology of how a circuit is routed on the FPAA switches, Figure 17d. Using the RAT, new designs can be created or existing designs can be modified by pointing and clicking with the mouse [68].

Figure 14 (a-b) illustrates the basic idea of how circuits are created on the FPAA. Assume that Figure 14a represents a circuit schematic that one would like to program onto an

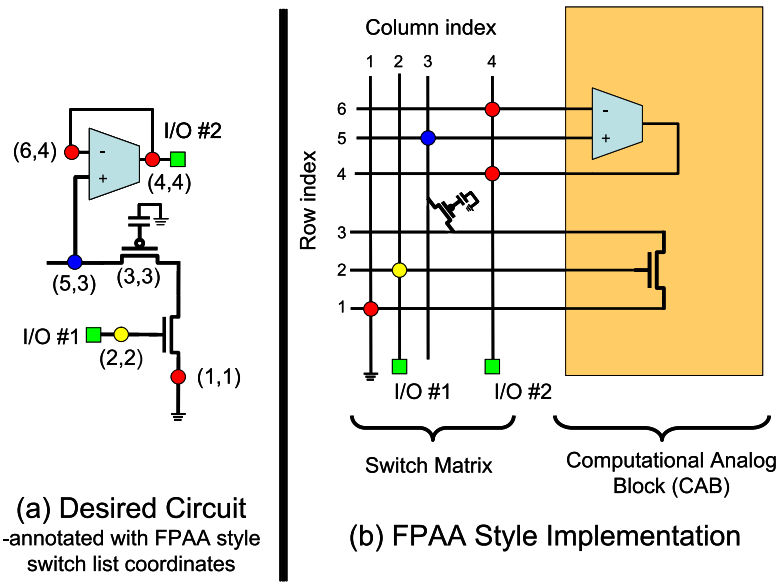


Figure 14. This illustrates the basic idea of how circuits are created on the FPAA.

FPAA. It consists of three components: an OTA, a floating-gate pFET, and an nFET. Figure 14b represents a simplified, small portion of what's inside an FPAA. The colored dots in (a) and (b) correspond to each other. In this case, six coordinate parameters specify the topology of the circuit: (1,1), (2,2), (3,3), (4,4), (5,3), and (6,4). The floating-gate pFET is shown at an angle in Figure 14b. It is programmed to have a certain conductance value. Actually, although not shown explicitly in the illustration, there is a floating-gate pFET at every row-column intersection of the switch matrix. The colored dots in the switch matrix that represent connections are actually fully turned on floating-gate pFET transistors that are used to make the connections.

Once an FPAA is targeted, the user can interface with the FPAA I/O in a couple of ways. First the user can jumper FPAA I/O to the discrete DAC and/or ADC ICs. These ICs are controlled through the Matlab interface. Table 1 lists a few sample Matlab commands. The user can also interface with the FPAA by using the audio amplifier ports. Fig. 15 shows the legend used to identify the various headers. There are thirty-two header pins on the board that are considered "factory settings" and are normally jumpered to specific control pins.

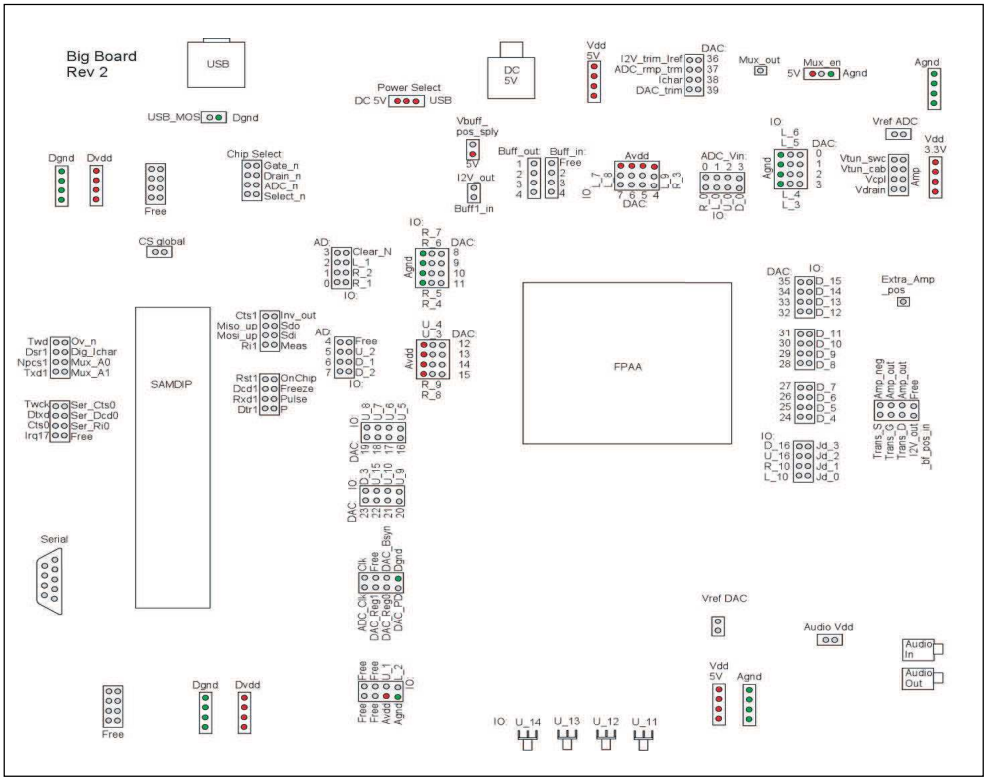


Figure 15. Header Map used as a legend to identify pins on the programming and control FPAA board. U_*, D_*, L_*, R_* are FPAA I/O pins.

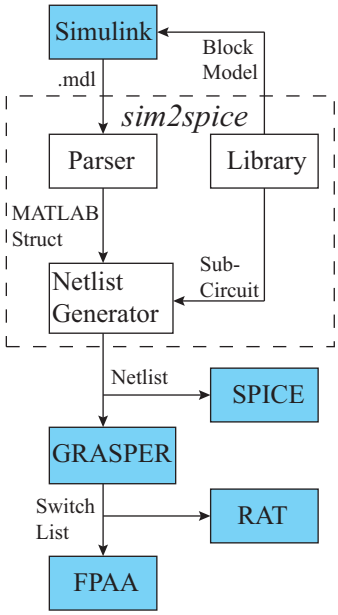


Figure 16. Software flow for designing systems on the FPAA. Top level designs are done in Simulink. *Sim2Spice* converts it to a Spice netlist, which can then be compiled into an FPAA switch list [6].

Table 1. Sampling of Matlab commands used to interface with FPAA

Matlab Function	Description
SET_DAC_USB	sets one of the 40 channels on the DAC IC
READ_ADC_USB	reads into Matlab a value from the ADC IC.
PROGRAM_aa	used to program a list of elements on the FPAA

2.4 Design Flow Using a Demonstration Test Circuit

Figure 17 shows the design flow for implementing a lowpass filter on an FPAA. First a Simulink block diagram of the system was generated, Figure 17a. Although not shown in this picture, this system can be digitally simulated in Matlab. Next, the *Sim2spice* tool, Figure 16, is used to generate a SPICE file from the Simulink block diagram, Figure 17b. Figure 17c shows the text file which is the output of the GRASPER compiler. The first two numbers in each line are the row and column locations of a particular floating-gate transistor on the FPAA, and the last number on the line represents the desired current in the transistor. Figure 17(d) shows the topology of the GRASPER routing on a RASP 2.9 IC. The switch list was targeted onto a RASP IC and a step input (blue) was applied to the input pin. The result was measured and shown in black in Figure 17(e).

2.5 Floating-Gate Transistors

Floating-gate transistors are well known for their use as the nonvolatile memory element in flash memories [98]. They are also the *secret sauce* of the FPAA. This is largely because, among other things, floating-gate transistors are used to make circuit connections within the FPAA. This process is illustrated in Figure 14. floating-gate transistors are unique because they can be programmed to conduct current by adjusting the charge on the gate terminal of the transistor. Unlike a conventional transistor, floating-gate transistors have an isolated gate terminal which, like a capacitor, can hold charge. This is advantageous because one does not need to actively maintain a voltage on the transistor's gate terminal in order to maintain its state (on, off, or resistive). Once programmed, the gate is set and the circuit has memory so it will maintain its map even if the power is removed from the IC. This is

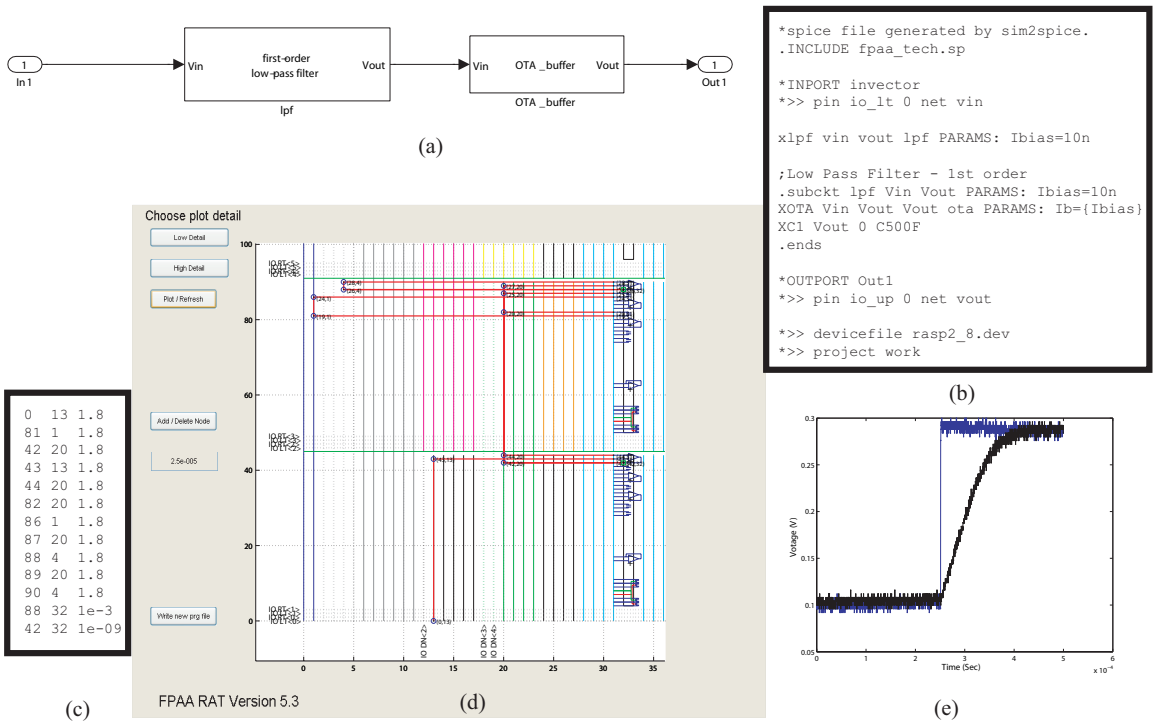


Figure 17. Design Flow for a low pass filter. (a) Simulink Block Diagram. (b) SPICE list generated by *Sim2Spice* tool. (c) FPAA switch list generated by GRASPER tool. (d) RAT Figure showing switch list routing on RASP 2.9 IC. (e) Measured Results from RASP IC: blue is the input signal, black is the lowpass filtered output.

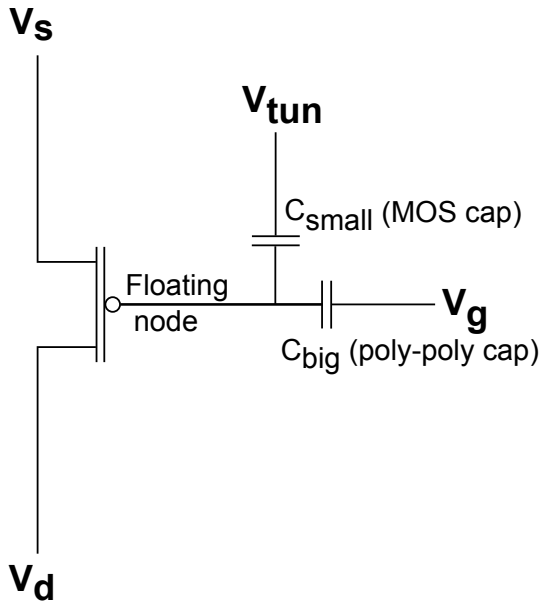


Figure 18. Floating-gate transistor

called nonvolatile.

Floating-gate transistors are programmed using Hot-Electron Injection and Tunneling processes [99, 98, 100, 101]. Hot-Electron Injection reduces the amount of charge on the gate (adds electrons), and Tunneling increases the amount of charge on the gate (removes electrons). There are four parameters that can be used when programming a floating-gate transistor: Drain voltage, Source voltage, Gate voltage, and Tunneling voltage, Fig. 18.

2.6 Summary

This chapter has described a comprehensive set of software and hardware tools that lets users quickly and easily create custom AVLSI circuits. The software tools allow users with varying levels of circuit design experience to be successful at synthesizing circuits. The hardware infrastructure platform allows users extreme flexibility to monitor and control the FPAA pins. The adaptor board allows users to quickly interchange FPAAs on the main programming and control board.

CHAPTER 3

RESISTIVE GRID PATH PLANNING

Path planning is a critical task for robots, autonomous vehicles, animated characters, etc. Figure 1 is a cartoon showing the ultimate goal of the problem being addressed in this chapter, namely how to use a reconfigurable Analog Very Large Scale Integration (AVLSI) circuit to plan a path for a Micro Aerial Vehicle (MAV) (or similar power constrained ground or sea robot) through an environment in an effort to conserve its limited battery resources. Path planning can be summarized with the following three tasks given that states, actions, an initial state, and a goal state are provided. The robot should:

1. Find a sequence of actions that take the robot from its Initial state to its Goal state
2. Find actions that take the robot from *any* state to the Goal state
3. Decide the *best* action for the robot to take now in order to improve its odds of reaching the Goal

Figure 19 shows the system view of path planning in the context of an unknown environment. Path planning computations are typically executed on digital microprocessors. This chapter presents results of using two different floating-gate based Field Programmable Analog Arrays (FPAA) for the path planning computation, the RASP 2.8a and the RASP 2.9V. A motivating reason for using AVLSI for path planning is the potential for better time and space complexity and lower power processing capabilities when compared to a microprocessor [7, 2]. This may be a significant advantage for MAVs, ocean gliders or other robot applications where the power budget for Guidance, Navigation, and Control is limited [29, 30]. Path planning for UAVs is further addressed in many other sources [102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112]. MAVs "have been defined to have no length dimension greater than 6 in. with gross takeoff weights of approximately 200 g or less [113]." The electronics are estimated to be between 10% and 25% of the MAVs mass,

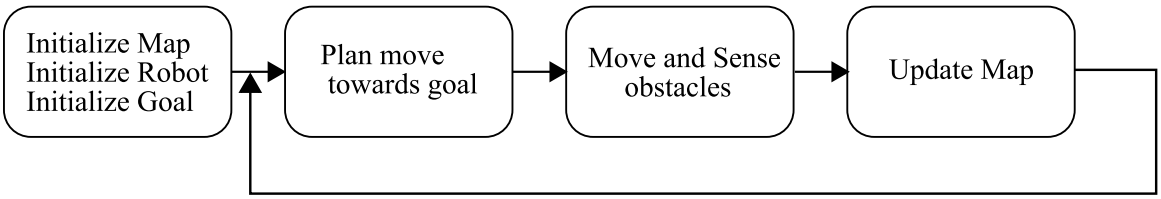
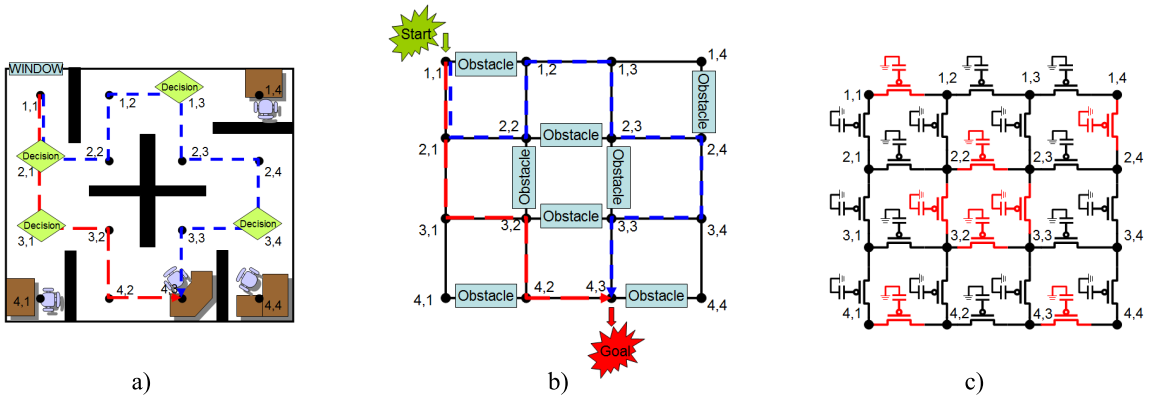


Figure 19. This shows the system view of path planning in an unknown environment.

and the battery mass is estimated between 25% and 42% [102, 104, 105]. Because the battery is such a significant portion of a MAV's mass, power savings could be significant in this application. Woods estimates a preliminary power budget of 50 mW for a 2220 mg MAV glider [103]. Although not a MAV, the microcontroller and embedded computer for a common research robot (Pioneer 3DX) has been calculated to use approximately 48% of the power [114]. Existing FPGA planners include an implementation of an entire Genetic Algorithm-based path planner entirely on FPGA hardware [115] and FPGA implementations of planners based on wavefront or stencil based gradient calculations [116, 12]. In [12] the authors describe processing 33 maps with a resolution of 1024x1024 per second on a midsize Virtex-5 FPGA. They also point out that a main bottleneck in applications like path planning on FPGAs is often the external memory bandwidth. In the FPAA planner, the nodes themselves contain the important path information so memory bandwidth does not seem to be an issue with this system.

A custom Application Specific Integrated Circuit (ASIC) could be developed to implement analog path planning, however custom ASIC designs are fixed so any revisions would incur a long design cycle/fabrication time. FPAAs, however, allow the designer or the autonomous agent itself to reconfigure the analog connections within the Integrated Circuit (IC) using software and hardware infrastructure. This allows quick design changes and re-use of a single IC [31, 7]. Chapter 2 describes the FPAA embedded system infrastructure used in this path planning research. This chapter is based upon work presented at two conferences [7, 2] and upon a journal paper to be reviewed [72].



a) b) c)

Figure 20. Converting the office grid world into an AVLSI representation [2]. a) Office with walls as obstacles. b) Simplified grid representation of office. c) floating-gate transistors used to implement the obstacles.

In summary, analog path planning is explored because it represents a potential decrease in time and space complexity, a potential reduction in power needed for computation, and potential decrease of computation time. An FPAA analog path planning implementation is useful because reconfigurable AVLSI systems provide circuit tune-ability and flexibility that custom ASICs do not provide [7, 2]. A mathematical analysis of an analog planner is presented in Section 3.1. A reconfigurable analog implementation of analog path planning is described in Sections 3.2. Experimental results using the RASP 2.8 FPAA hardware along with analysis are shown in Section 3.3. Experimental results using the RASP 2.9v FPAA hardware along with analysis are shown in Section 3.4.

3.1 Mathematical Analysis of Analog Planner

Assume that there exists a two dimensional resistive grid where the edges between nodes are resistive elements, Figure 20c. Further assume that there is a single source of current input into one node of the grid (of dimension X by Y), and there is a single current sink on a node of the grid. At some time, *t*, the node voltages will settle to a steady state voltage. One can represent this by the scalar function in (1).

$$V = f(x, y) \tag{1}$$

The gradient in this problem is a vector field which can be evaluated at each node in the grid. The gradient (or vector) at each node points in the direction of the node where the voltage increases most. The magnitude of the vector tells how fast the voltage rises in that direction. Expressed using unit vectors (2) [2]:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = \frac{\partial f}{\partial x}\hat{i} + \frac{\partial f}{\partial y}\hat{j} \tag{2}$$

The Laplacian, (3), is a differential operator that effectively provides the partial derivatives of the gradient (which in this problem is the second partial derivative of the voltage function) [117]. Some of the first implementations of using Laplace's equation for path planning are [118, 119, 7, 2].

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3}$$

One can approximate the Laplacian for our problem by using the discrete Laplacian and specifically, the Five Point Stencil finite difference method, (4), where h is the grid size. One can derive (4) using the Taylor series and assuming that higher order terms are negligible [120, 118, 7].

$$\Delta f\,(x,y) \approx \frac{f(x-h,y)+f(x+h,y)+f(x,y-h)+f(x,y+h)-4f(x,y)}{h^2} \tag{4}$$

The *Laplace's equation*, (5), uses the Laplace operator on a function and sets it to zero [117]:

$$\Delta f = 0 \tag{5}$$

Substituting (4) into the Laplace equation, (5) gives (6). This shows that if the function describing the grid's voltage is the solution to the Laplace equation, then the voltage at each node is the average of its four neighbors:

$$
\begin{aligned}
&f(x-h,y)+f(x+h,y)\\
&+f(x,y-h)+f(x,y+h)=-4f(x,y)
\end{aligned}
\tag{6}
$$

3.2 Constraint Based Path Finding in AVLSI

Figure 20 shows how an office environment is modeled using a transistor based resistive grid. The transistors in the free path regions are turned fully on by programming the floating-gates to conduct current, and the transistors representing obstacles (in red) are turned off by programming the floating-gate transistors such that they would not conduct current. This resistive grid is then used to solve a path planning problem. Low conductance transistors represent blocked paths, and high conductance transistors represent clear paths. The transistors used for planning in the FPAA are a special type of transistor called *floating-gate* transistors. This sets this work apart from earlier works in VLSI path planning which used either resistors or non-floating-gate transistors for the grid [49, 48, 50, 44, 51, 52, 53, 54, 55, 56, 57].

The key part of the floating-gate devices is that each of them can be used in an analog way, as well as in a configurable approach. Other approaches do not allow all of those options. A transistor can be operated in above threshold or sub-threshold regimes. Subthreshold operation results in much lower power use because less current is being conducted. The current in a pFET transistor in subthreshold operation may be described in (7) [121].

$$
I = I_0 e^{\frac{V_{dd}(\kappa-1)}{U_T}} e^{\frac{-\kappa V_g + V_s}{U_T}} \left[1 - e^{\frac{-V_{sd}}{U_T}}\right] \tag{7}
$$

In (7), I_0 is a constant representing pre-exponential factors, κ is a constant representing the capacitive coupling ratio from gate to channel, and U_T is the thermal voltage [122, 121]. Taylor expanding (7) with respect to V_{sd} about zero and dropping all higher order terms gives an expression for the pFET's resistance when V_{sd} is small.

$$R = \frac{U_T}{I_x} \tag{8}$$

Where I_x, (9), can be programmed from fA to uA [123]. Programming accuracy can be achieved at 9 bits of floating-gate voltage [31].

$$I_x \equiv I_0 e^{\frac{V_{dd}(\kappa-1)}{U_T}} e^{\frac{-\kappa V_g + V_s}{U_T}} \tag{9}$$

Floating-gate transistors are unique because they can be programmed to conduct current by adjusting the charge on the gate terminal of the transistor. Unlike a conventional transistor, floating-gate transistors have an isolated gate terminal which, like a capacitor, can hold charge. This is advantageous because one does not need to actively maintain a voltage on each gate terminal of the grid. Once programmed, the gate is set and the circuit has memory so it will maintain its map even if the power is removed from the IC.

There are two novelties to FPAA implementations. First, they are novel because this path planner is being implemented on a generic reconfigurable analog platform. Second, the transistors used to represent the paths and obstacles in the resistive grid are not, as used in previous papers, conventional transistors. The resistors are implemented using floating-gate transistors.

A family of floating-gate based large-scale FPAAs is being developed at Georgia Tech. As described in Chapter 2, these reconfigurable analog platforms utilize a switch matrix of programmable floating-gate transistors as switch elements. These switch elements have a dual role as computational elements [91]. This specific feature is exploited in this chapter. The reconfigurable nature of the platforms allows rapid building and testing of different circuit configurations [31]. In the next two sections, two FPAA ICs are used for path planning.

3.3 Path Finding on the RASP 2.8 FPAA

The RASP 2.8 IC was used to demonstrate FPAA based AVLSI path planning [7]. A die photo of the RASP 2.8 IC is found in Figure 8a. Two experiments are described.

3.3.1 Experiment 1: Solving a simple grid problem

A toy problem was first developed to illustrate how an FPAA can be used for path planning. Consider an indoor office environment as in Figure 21a [7] and the problem of planning a path for a MAV from the window to a specific desk. The space has been discretized into grids and the nodes labeled by their row and column numbers. The robot starts at node (1,1) and the goal is node (4,3). It is assumed that all obstacle free edges have the same cost value. There are two solutions to this problem. If we assume that traversing each edge has a cost of 1, the best solution has a cost of five, and the second solution has a cost of nine. The toy problem can be re-drawn generically as in Figure 21b. Two solution paths are shown, and decision points are labeled. The solution tree for this problem is found in 21c. Two solutions are distinguished by red or blue. Traditionally, this grid world can be evaluated for a solution using methods such as the Breadth-First-Search (BFS) or Depth-First-Search (DFS) algorithm [1].

A resistive grid similar to the one in Figure 20c was implemented on the hardware platform of Figure 9 and using a RASP 2.8 FPAA IC. Each node in this grid also has buffer circuitry in the actual hardware version for proper reading of the grid voltages. The grid voltages can be read with either an an Analog-to-Digital Converter (ADC) on the FPAA control board or an external multimeter or oscilloscope. The buffer circuit is a floating-gate input Operational Transconductance Amplifier (fgOTA) in a unity gain buffer configuration.

Finally, the method used to insert current into the system was to use a conventional pFET (a component in the FPAA CAB) connected to three I/O pins. The source is connected to Vdd, the drain is connected to the start grid node, Node (1,1), and the gate was connected to a DAC channel on the FPAA programming and control board. Similarly, the

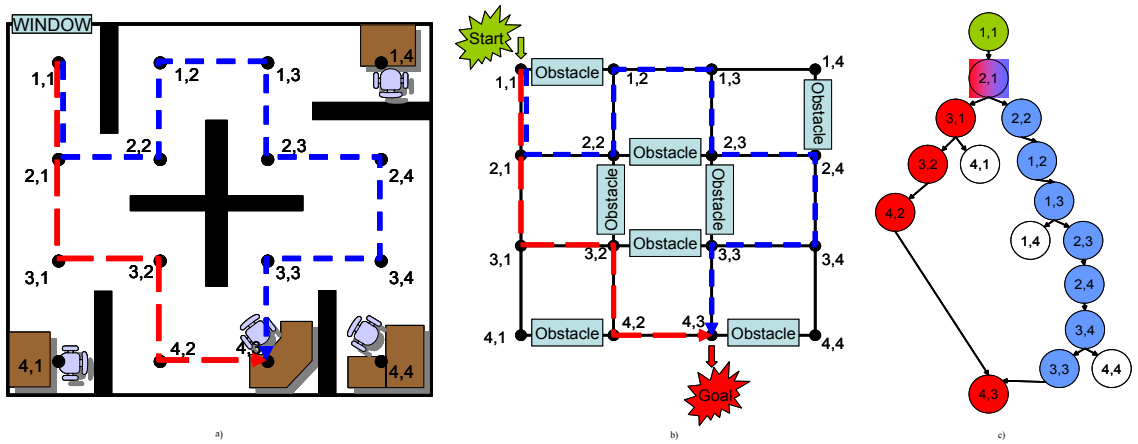
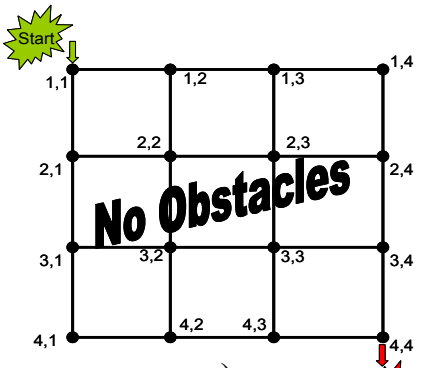


Figure 21. a) Toy problem two dimensional office environment where the goal is to plan a path for a MAV from the window to a specific desk. The environment that has been discretized into a grid with labeled nodes. Two path solutions are identified. b) This figure shows the locations of the Start, Goal, and Obstacles in the Cadence simulations. The shortest path to the goal is through the red line: Start=(1,1)-(2,1)-(3,1)-(3,2)-(4,2)-(4,3)=Goal. The alternate path to the goal is by the blue line. c) The solution tree to the toy problem has two solutions (red and blue paths) [7].

goal Node (4,3) was connected to the drain of a pinned-out diode connected nFET in the FPAA CAB, and its source was connected to ground.

3.3.1.1 Results

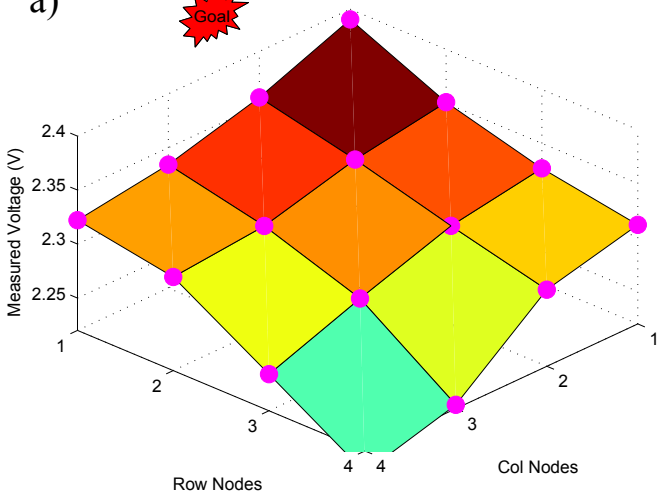
Figure 22 through Figure 29 show measured results from the RASP 2.8 FPAA. Figure 22 shows a grid with no obstacles. Figure 23 shows the data for the problem proposed in Figure 21. There are four decision points of interest in this map. Two decision points for the optimum red path are at Nodes (2,1) and (3,1), and two decision points for the non-optimum blue path are at Nodes (1,3) and (3,4). When interpreting the data in Figure 23b and c, one is looking for the path of largest current. Since there is not a current meter on each path, one may use node voltage measurements. When at a decision node, the best route to take is the route with the lowest voltage on the next available node (so that there is the greatest voltage drop). The data corresponding to the decision points for the blue and red routes guide the robot. Figure 24 through Figure 27 show a broader range of the grid's response to varying input currents. Also, experiments were performed to characterize the performance of the floating-gate OTAs, Figure 28. Finally, Figure 29 shows that, as expected, the current into the grid is equal to the current out of the grid. In order to show



a)

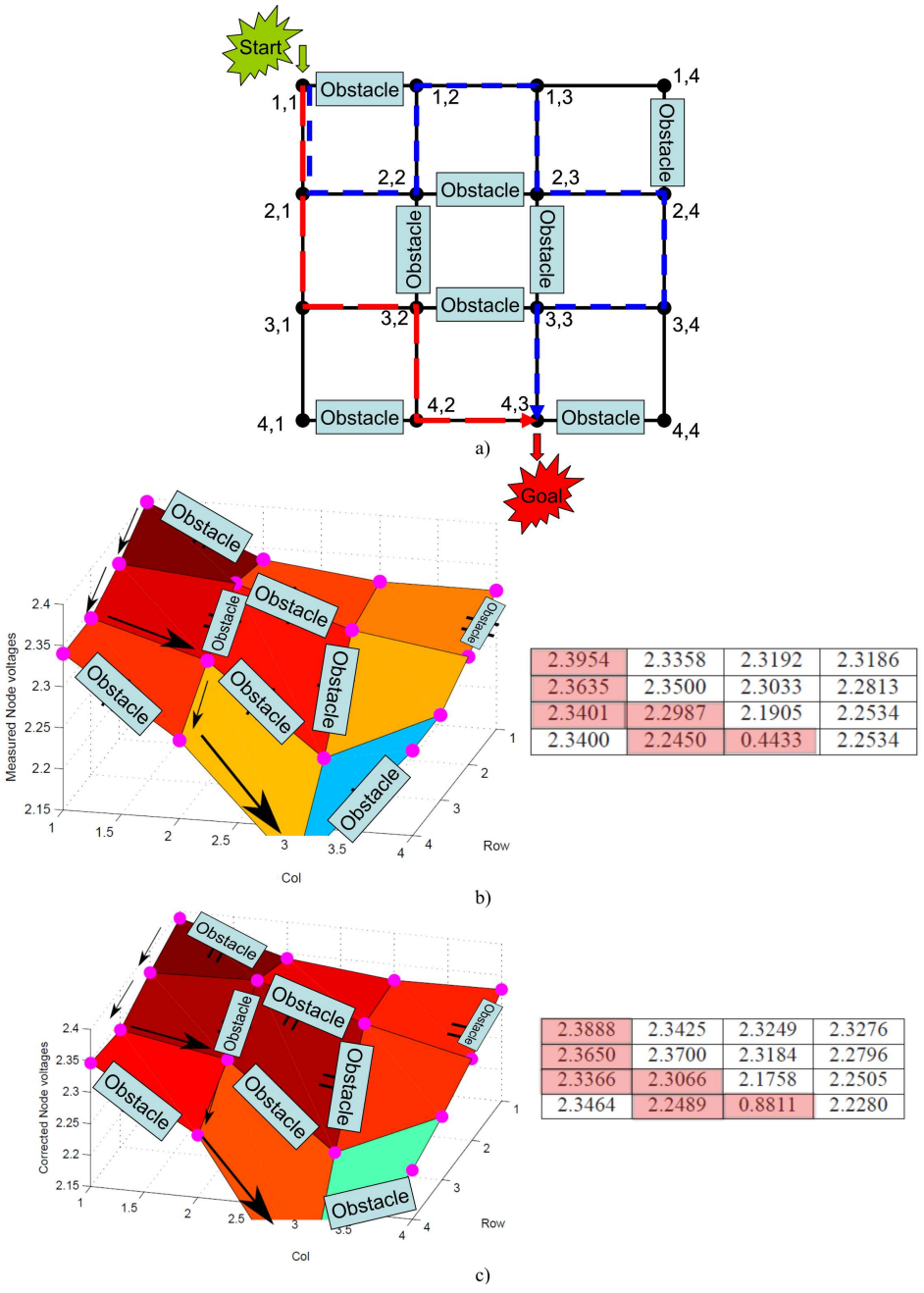
2.3904	2.3509	2.3269	2.3119
2.3574	2.3370	2.3128	2.2910
2.3344	2.3147	2.2847	2.2235
2.3222	2.3067	2.2539	0.4903

b)



c)

Figure 22. a) This experiment was for a 4x4 grid structure with no obstacles. The robot is at node (1,1), and the goal is at node (4,4). The floating-gate pFETs were each programmed to 1e-006 A and the measurements were performed with the current source pFET's gate voltage at 1.5V. b) Node voltage measurements were made directly at the node and did not use the fgOTA buffers. Coordinate (1,1) is top left, and coordinate (4,4) is bottom right. c) Surface plot of the data in b [7].



2.3954	2.3358	2.3192	2.3186
2.3635	2.3500	2.3033	2.2813
2.3401	2.2987	2.1905	2.2534
2.3400	2.2450	0.4433	2.2534

2.3888	2.3425	2.3249	2.3276
2.3650	2.3700	2.3184	2.2796
2.3366	2.3066	2.1758	2.2505
2.3464	2.2489	0.8811	2.2280

Figure 23. This experiment compares the results of using the fgOTA buffers when measuring the node voltages [7] a) This experiment was for a 4x4 grid structure with 8 obstacles. The robot is at node (1,1), and the goal is at node (4,3). The floating-gate pFETs for the paths were each programmed to 1e-006 A and the obstacles were set to 0 A. The measurements were performed with the current source pFET's gate voltage at 1.5V for b and 1.5421V for c. b) Node voltage measurements were made directly at the node and did not use the fgOTA buffers. c) Node voltage measurements were made using fgOTA buffers. The measurements were calibrated using fgOTA characterization curves, Figure 28.

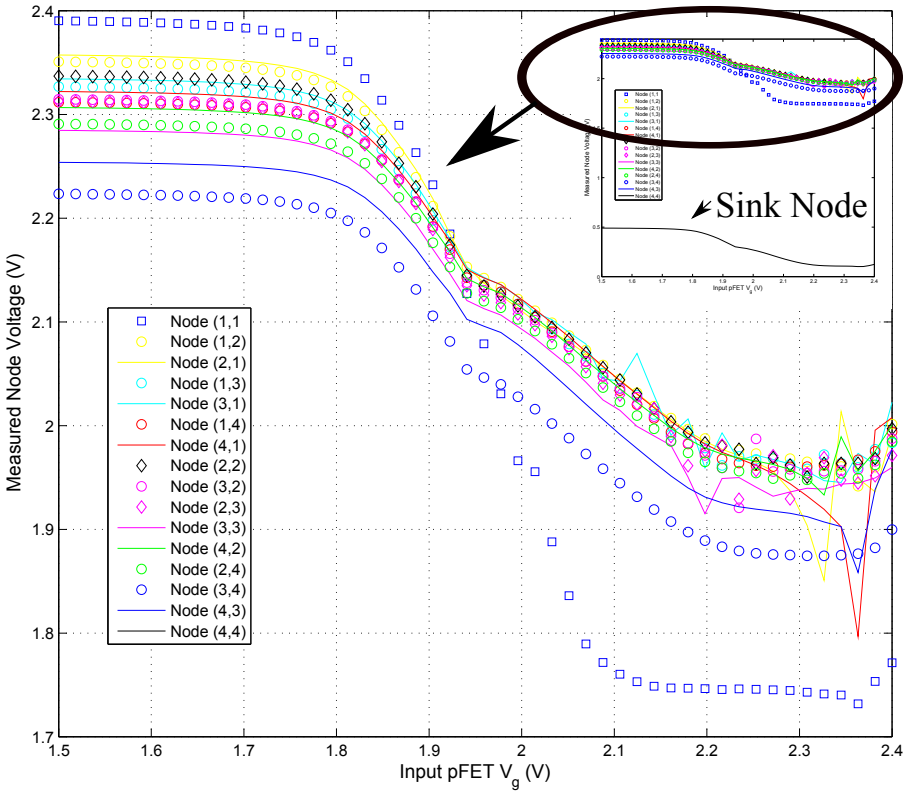


Figure 24. The data in Figure 22 is from a vertical slice of this data when the x-axis is 1.5 V [7]. This shows each node's steady state response to varying levels of input current.

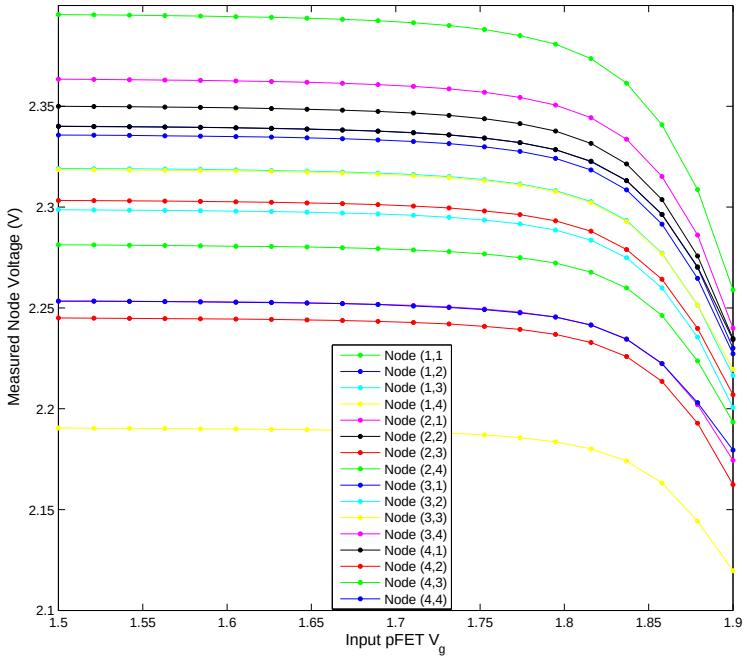


Figure 25. The data in Figure 23b is from a vertical slice of this data when the x-axis is 1.5421 V. This shows each node's steady state response to varying levels of input current. The sink node is about 0.44 V, but is not displayed [7].

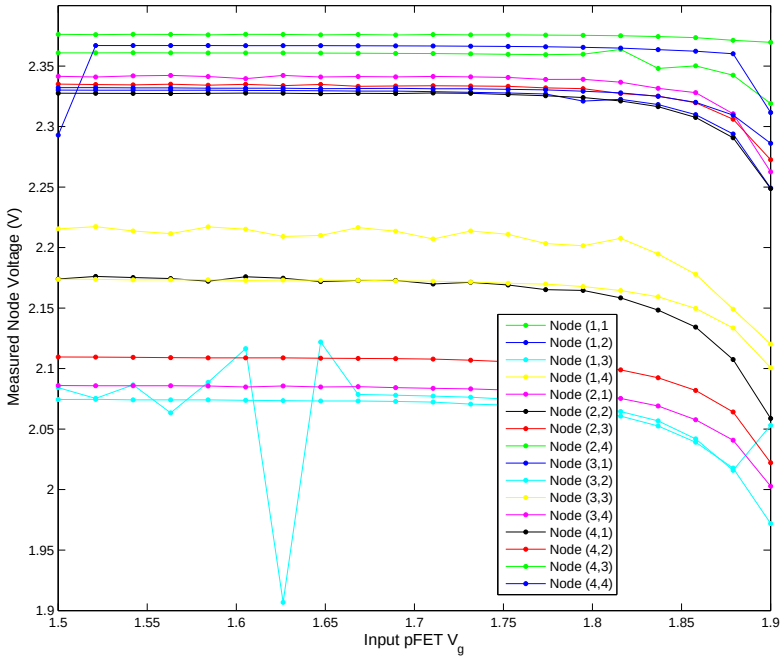


Figure 26. The data in Figure 23c is *derived* from a vertical slice of this data when the x-axis is 1.5421 V. These are the raw measurements made from the fgOTA buffers at the nodes. Because offsets exist in the buffers, this data needs to be calibrated to the correct value. The measurements were calibrated using fgOTA characterization curves shown in Figure 28.

this, current measurements were taken at the input node to the grid to characterize the input current. The output current was correlated to the input current by measuring the sink node's voltage response for various input currents and then deriving the output current from the diode's characterized current/voltage curve.

3.3.1.2 Experiment 1 Summary

In the toy problem discussed in this chapter, it is feasible to apply the input at the start node, and measure all of the outputs to establish the entire plan from start to goal. In a real system, however, with a much larger grid, the node voltages become too small to definitively measure all nodes because the current has to spread over a much larger grid [44]. In this case, it is common to re-apply the input to the robot's current grid square when the robot reaches a new node, and then measure the surrounding node voltages. The drawback to this system is more computation time and power used. A second issue is

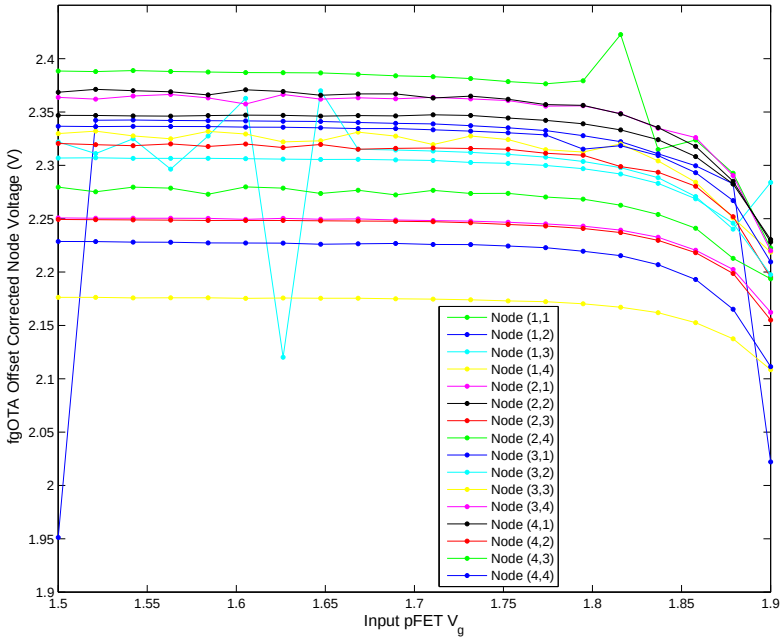


Figure 27. The data in Figure 23c is from a vertical slice of this data. These are the calibrated measurements derived from the fgOTA buffer measurements at the nodes.

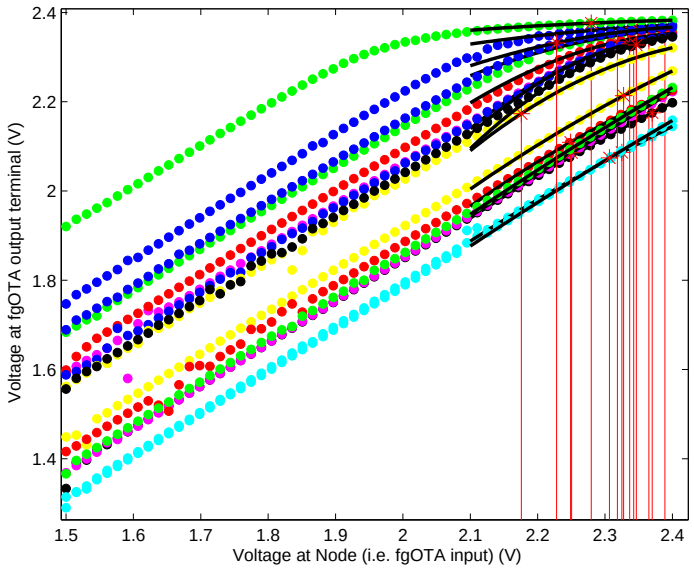


Figure 28. These curves show the fgOTA characterization results [7]. Voltages between 1.5 and 2.4 V were applied to the input to the buffer and the output was measured. Ideally, the output should equal the input, but that is not the case as the curves are shifted up and down from the ideal. Curves are fit to these data points and are used to calibrate the fgOTA measurements. The red lines show the calculation of the calibrated values used to produce the surface plot in Figure 23c.

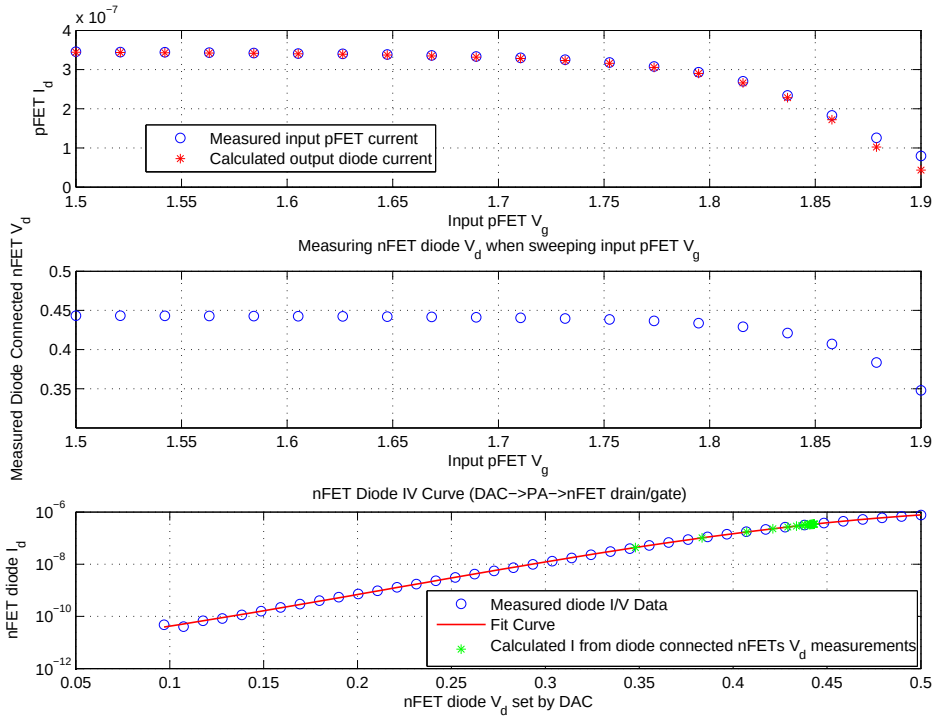
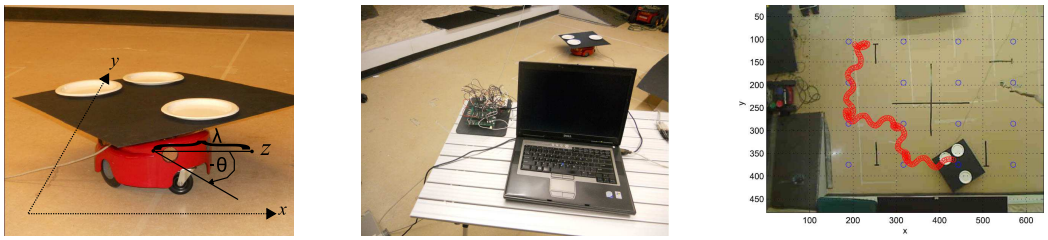


Figure 29. Verifying Input Current = Output current: Input current pFET is connected to Node (1,1) of a 4x4 grid. Node 15 is grounded through a diode connected nFET.



(a) Mobile Robots AmigoBot [125] (b) Experimental setup (c) Measured Results

Figure 30. Our experimental environment showing a) the robot with coordinate axes and b-c) the implementation of an FPAA generated plan.

reading the node voltages. In this toy problem, we can easily read out each individual node voltage, but larger grids would likely need to use another system.

3.3.2 Experiment 2: Integration of FPAA and Robot

We are using an AmigoBot robot to demonstrate the analog planning system, Figure 30a. The FPAA and AmigoBot robot are integrated such that the FPAA acts as a "planning co-processor" for the robot. A block diagram showing how the analog resistive grid's planner fits into the larger robot system is found in Figure 31a. The Executor's function is to act as an interface between the FPAA and the low-level digital controller. An example software flow is found in Figure 31b. This simple, proof-of-concept flow assumes four things: a known map, a static environment, the robot's starting location is known, and the goal location is known. More complex flows could move the current source to follow the robot in the grid [48] and can also incorporate re-planning. The task of the Navigation block in Figure 31a is to convert high level plans such as "Move from the window to the desk at grid (4,3)" in Figure 20a to low level commands. A position-stabilizing controller adjusts the robot's forward and angular velocity and is used to drive the robot to points in the grid [124]. The control equations are based on feedback linearization. The kinematic equations of motion are shown in (10).

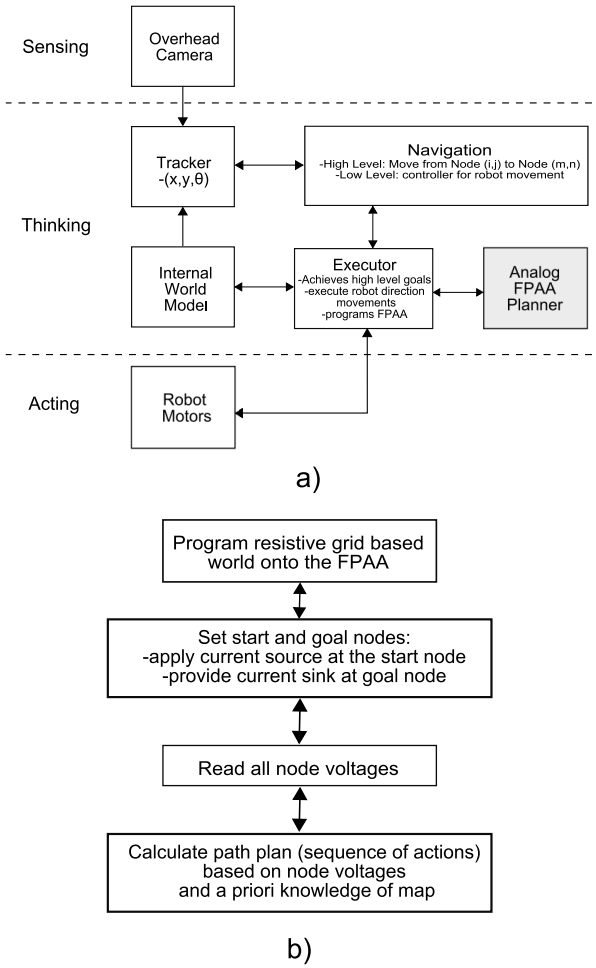


Figure 31. a) High level control system block diagram and b) software flow of the Executer designed to integrate the analog planner and the robot.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -c(\lambda - \varepsilon) \end{bmatrix} + \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{10}$$

This uses a coordinate transformation where a λ offset is chosen from the center of rotation (see axis overlaid over Figure 30a. An overhead camera is used for localization. Image processing routines segment three dots on the back of the robot and these are used to locate the robot in (x,y, θ) image space [126].

For this proof-of-concept system, two programming environments were integrated: the

Matlab of the FPAA and the C++ code for *Player*. An extensive body of Matlab code has been developed by the Integrated Computational Electronics (ICE) Lab at Georgia Tech to program and communicate with the FPAA board. Open source *Player* software, [127], is used for interfacing with the FPAA (via Matlab) and for controlling the AmigoBot robot. The FPAA Matlab code is called by *Player* using Matlab engine functions [8].

3.3.2.1 Experimental Results

This section presents our initial results of integrating a robot with an AVLSI co-processor. The experimental setup is shown in Figure 30b. The robot is shown in the background, the FPAA is shown on the left corner of the desk, and the overhead camera is above (not shown). All are tethered to the laptop running *Player* via USB cables. Figure 30c shows our initial experimental results from a robot in a four by four grid world. The AVLSI FPAA hardware has been integrated into *Player* as a co-planner for the AmigoBot robot. This is an image taken from the overhead camera used for localization. The results are for an experiment in a 4x4 office grid world. The cubicle partitions are marked in black tape on the floor. The overlaid red dots are the recorded trajectory of the robot moving from node (1,1) to node (4,3) The overlaid blue circles mark the grid nodes. At each node traversed by the robot, the FPAA was consulted for the next node.

Although this is a trivial planning problem, it demonstrates two major goals. First, our system can make complete plans using floating-gate resistive grids (based on our limited experiments). Second, the supporting FPAA hardware and software are at a level of sophistication where they can be reliably integrated into robot platforms. This system has three modes of operation. These three methods provide useful options when debugging various parts of the system. A brief discussion of each of the modes is now presented.

Real Robot, Real FPAA Results: The FPAA and an AmigoBot robot were integrated together and localization was performed using an overhead camera (640x480 pixel resolution). The robot successfully navigated its path on the floor as directed by the FPAA. Figure

30c shows in red the path the robot made from its start to its goal. At each node (represented by a blue circle), *Player* queried the FPAA co-processor to help decide whether to go straight, or turn left or right at each node in order to reach the goal.

Real FPAA, Simulated Robot Results: This is a Hardware in the Loop (HWIL) environment where the actual FPAA hardware is being called by *Player* and is interacting with a virtual robot in a three dimensional robot simulator with dynamics. This environment is called *Gazebo* and interacts with *Player* using the same control code. Ideally, one can take the same *Player* control code to control a virtual robot or the real thing. Figure 33 shows an image from a *Gazebo* simulation. Virtually identical software is used for this HWIL simulation as in the section regarding *Real Robot, Real FPAA Results*.

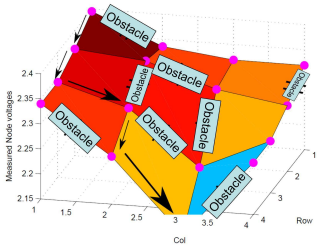
Simulated FPAA, Simulated Results: Finally, it is possible to simulate the FPAA results by using Matlab to solve for node voltage values using, for example, Kirchhoff's laws.

3.3.2.2 Analysis

This path planning problem can be formulated as a tree search problem. These problems are typically evaluated with four metrics: Completeness, Optimality, Time Complexity, and Space Complexity [1]. Time and Space Complexity are addressed further in the following sections. Time complexity is typically measured by the number of nodes generated [1]. Space complexity is measured in terms of the maximum number of nodes stored in memory [1].

3.3.2.2.1 Time Complexity Time complexity is not as simple as number of nodes generated with the FPAA. There are three items to consider when calculating the total time cost of the FPAA planner: FPAA grid programming time, solution computation time, and time to read the solution from the grid. Each of these are addressed below.

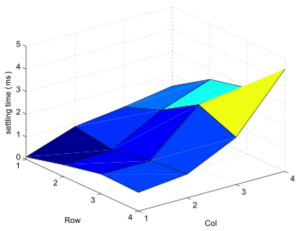
3.3.2.2.1.1 FPAA Programming Time Measurements Programming a grid map onto the FPAA is done in two main phases. First, the FPAA is erased and prepared for programming. Second, the new map is programmed onto the FPAA. With our current software,



a)

2.3954	2.3358	2.3192	2.3186
2.3635	2.3500	2.3033	2.2813
2.3401	2.2987	2.1905	2.2534
2.3400	2.2450	0.4433	2.2534

b)



c)

Figure 32. a) Measured FPAA hardware results for a 4x4 grid like the configuration of the robot start, goal, and obstacles in Figure 20c. b) A table of the measured voltages with path identified by the pink squares. c) Measured node voltage settling times of the example office 4x4 resistor grid as a function of grid location.

it takes approximately 35 seconds to erase and prepare the FPAA for programming. This amount of time is independent of the size of map that will be programmed. The time needed to program the map is a function of two parameters: size of map and type of paths which connect the nodes on the map. There are three types of paths that we will consider. Type 1: impassable paths, Type 2: completely passable paths, and Type 3: a path which is passable, but with some degree of difficulty. This may be due to terrain such as sand, an incline, etc. The programming times for each of these paths is summarized in Table 2. As the state of the art in floating-gate programming advances, these times are expected to decrease.

In the 4x4 grid example, 38 floating-gate switches were used in the circuit. Of these, 22 switches were *overhead*. That is, they were needed to program the grid, but were not *path* elements. This overhead number changes according to grid size. The switches were generated automatically using *GRASPER* software [31]. In this example, this overhead represents about 58% of the total number of switches. Due to obstacles, the number of free paths was only about 67% of the total paths possible in a 4x4 grid. If we consider N as the number of nodes on the side of an NxN square grid, the number of possible paths is $O(2N^2)$.

3.3.2.2.1.2 Solution computation time The computation time for the FPAA is based on the time it takes for all of the grid's node voltages to settle to steady state in response to a current step input. For the 4x4 grid example, the computation time of this grid based

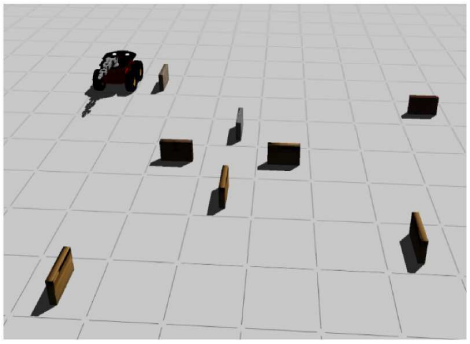


Figure 33. This is an image of a simulated office environment used in a FPAA Hardware in the Loop (HWIL) test.

Table 2. Grid Programming Times according to path type

	Type 1	Type 2	Type 3
Measured Erase and Initialize grid (sec)	35	35	35
Measured Program time per path (sec)	0	0.0486	4.4332
Expected Program times (sec) [31]	0	0.001	.050

FPAA planner is approximately 4.5ms. Figure 35 shows the transient response of each of the sixteen nodes in the grid. The limiting factor in this case is Node 15 which took about 4.5ms to settle. Figure 32a-b shows node voltage measurements from a 4x4 grid world, (c) shows the node settling times for each of the nodes. A step input voltage was placed on the pFETs gate at node $(1, 1)$ and this implemented a step input current to represent the robot's location at this node. A current sink was implemented at node $(4, 3)$ to represent the goal. The last node to reach steady state took 4.5ms.

3.3.2.2.1.3 Solution access time In the FPAA, once the nodes have settled, the solution is found by reading d nodes, where d is the depth of the shallowest goal node. We could say then, that the FPAA has Time complexity of $O(d)$. For comparison, Breadth-First-Search (BFS) has Time Complexity of $O\left(b^{d+1}\right)$, where the branching factor $b = 4$, and d is the depth of the shallowest solution. Figure 34a compares Analog-to-Digital Time Complexity as a function of shallowest solution.

3.3.2.2.2 Space Complexity To calculate a final path solution the FPAA planning system needs to maintain an adjacency list. This lists, for each node, all nodes that are one

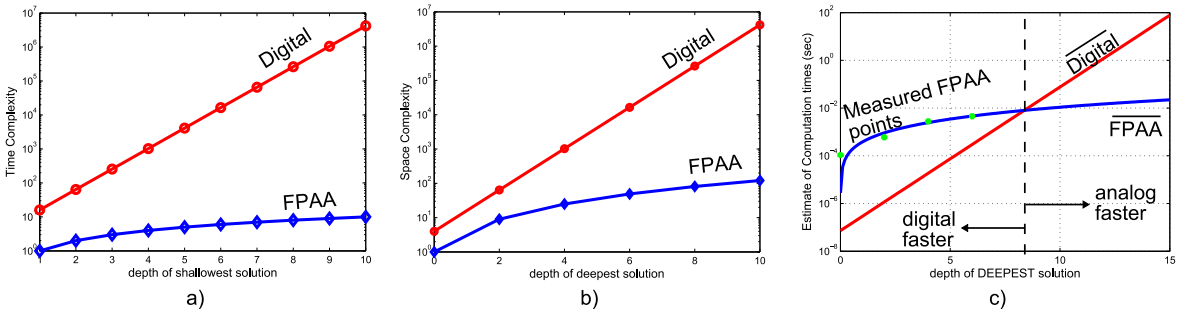


Figure 34. a) Comparing the Time complexity of the FPAA to BFS. b) Comparing worst case Space complexities of the FPAA to BFS. c) Comparing Computation Time of the FPAA to an estimate for BFS.

Table 3. Rasp 2.8: Comparing FPAA to BFS

Criterion	FPAA	Breadth First
Complete?	Yes (based on limited experiment)	Yes
Time	$O(d)$	$O\left(b^{d+1}\right)$
Space	$O\left(4N\left(N-1\right)+1\right)$	$O\left(b^{d+1}\right)$

step away through Type 2 or Type 3 paths. This list can have the form *[source node , list of adjacent nodes]*. For example, in the 4x4 grid of Figure 20, the robot can reach nodes (1,1), (2,2), and (3,1) from node (2,1). The corresponding adjacency list would be [(2,1), (1,1) (2,2) (3,1)]. This information is contained in MATLAB and combined with the node voltages read from the FPAA to choose a path. Assuming no obstacles for maximum space complexity, the worst case space complexity of the FPAA is $O\left(4N\left(N-1\right)+1\right)$, where N is the number of nodes on a side of a square map, i.e. NxN map. This is calculated using (11) where N_x terms are numbers i.e. $N_{middle-nodes}$ is the number of middle nodes, and A are numbers of adjacencies. BFS worst case Space Complexity is $O\left(b^{d+1}\right)$, where b = 4 and d is the depth of the DEEPEST solution.

$$\begin{aligned} Space\_complexity_{FPAA} &= \left(N_{corners} * A_{corner}\right) \\ &+ \begin{pmatrix} N_{non-corner-nodes-on-grid-edge-side} \\ *A_{non-corner-nodes-on-grid-edge-side} * 4 \end{pmatrix} \\ &+ \left(N_{middle\_nodes} * A_{middle\_nodes}\right) + 1 \end{aligned} \tag{11}$$

Table 3 summarizes the Time and Space Complexity comparisons between the FPAA

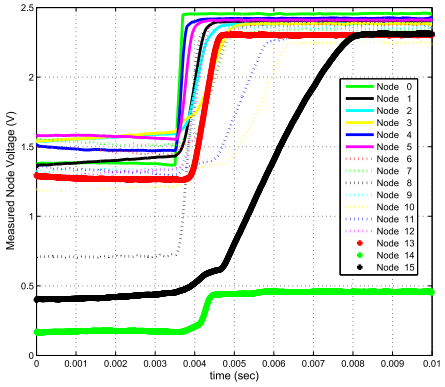
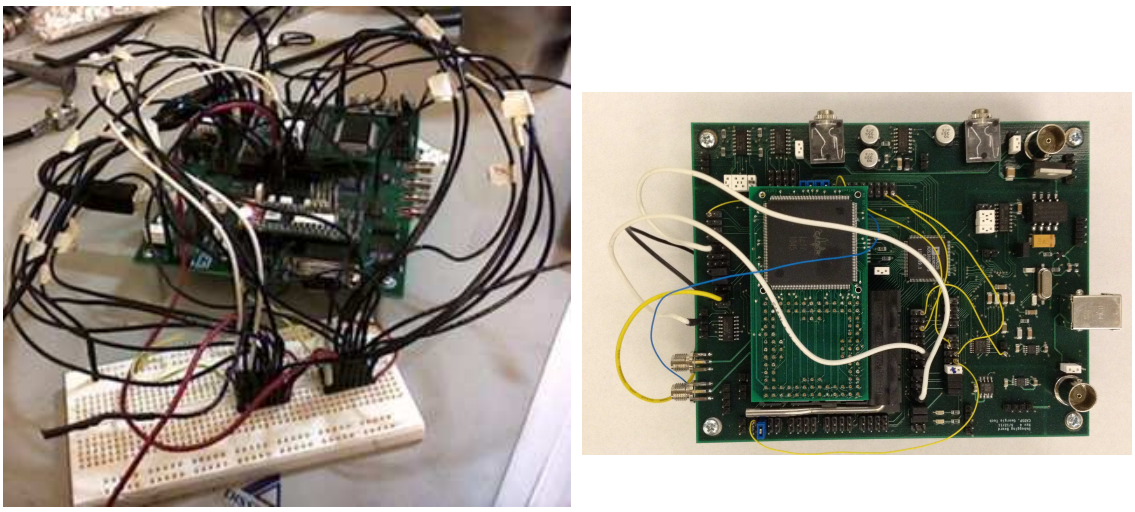


Figure 35. Measured transient responses for node voltages.

and Breadth-First-Search (BFS) [1].

3.3.2.2.3 Calculation Time Estimate Ideally, one would like to compare the actual solution times of the digital and analog solutions and not just operation numbers like Time Complexity. As an estimate, assume that the BFS algorithm is being executed on an a processor such as the ATMEL ARM7TDMI RISC processor operating at 55MHz max clock speed. Further assume that the solution is at the deepest solution of the grid. If one multiplies the BFS Time Complexity number by the inverse of the ARM7 clock then we can have a crude estimate of the digital computation time. To estimate the computation time of the FPAA, we extract a curve from the diagonal delay times of Figure 32c. Since BFS is in terms of *b* and *d*, and the FPAA settling time estimate is in terms of *N*, we use $N = (d/2)+1$ as the transformation. A comparison plot is shown in Figure 46c. Estimate of BFS Computation Time for 55 MHz processor: $O\left(b^{d+1}\right) * \left(\frac{1}{55Mhz}\right)$ where b = 4. Estimated FPAA Computation Time is based on extrapolation of the diagonals of the 4x4 delay measurement data in Figure 32c. Based on this graph, the prediction is that an FPAA solution may be faster than digital for solution depths greater than 8. This plot estimates that the FPAA will be quicker at solving plans where the solution depth is greater than 8. This corresponds to the deepest solution of a 5x5 grid.



a) RASP2.8 path planner

b) RASP2.9v path planner

Figure 36. This figure compares the "rat's nest" of wires and multiplexer ICs used to measure the data with the RASP 2.8a IC path planning system vs. the simplified system for the RASP 2.9v IC.

3.3.2.3 Summary

Figure 34a and b have shown that the Time Complexity and Space Complexity of the FPAA are orders of magnitude lower than that of BFS. Figure 34c also describes the solution depth at which FPAAs may find a solution quicker than BFS. Finally, the FPAA embedded planning system was successfully integrated with a real robot. One of the difficult parts of using the RASP 2.8a for path planning was accessing and measuring each of the nodes. Figure 36a shows the "rat's nest" of wires and multiplexer ICs used to measure the data. In the next section, the RASP 2.9v IC is used, and features of this IC greatly simplify the readout of the node voltages.

3.4 Path Finding on the RASP 2.9v FPAA

This section presents the results of path planning using the RASP 2.9v FPAA IC. The RASP 2.9V has a special feature that can be used to access any point in the routing fabric. This is accomplished using horizontal and vertical registers that are connected to the FPAA's routing fabric. These registers are addressed and controlled by the ARM core microprocessor and a Matlab interface. The Matlab interface could be removed for a FPAA system connected to an autonomous agent. This special feature allows one to avoid such a "rat's

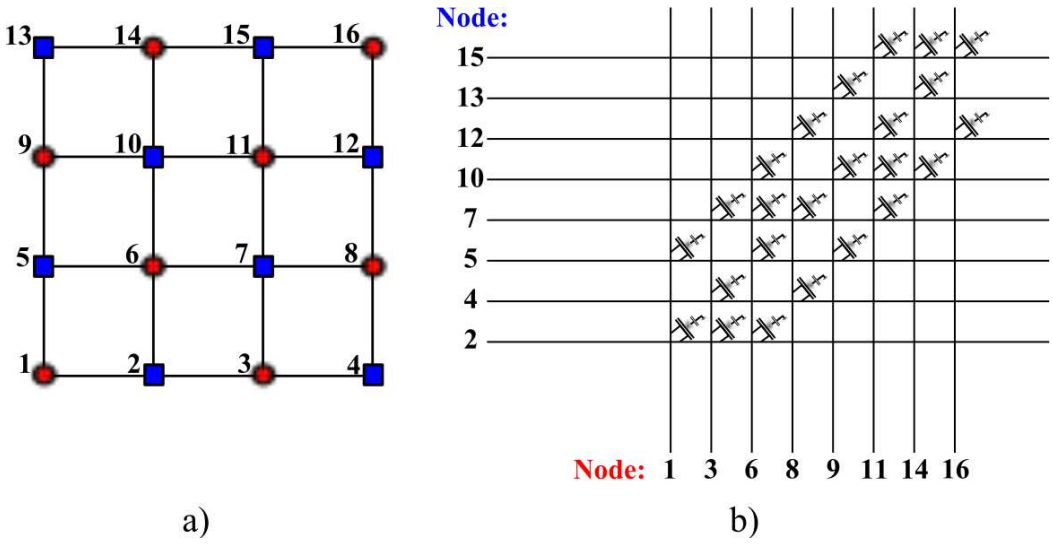


Figure 37. This figure illustrates how a 4x4 grid is implemented as a bipartite graph in the FPAA's routing fabric.

nest" of wires as depicted in the RASP 2.8 IC system in Figure 36a in favor of Figure 36b.

The grid structure for a path planning problem such as Figure 20 is implemented on the 2.9v FPAA using a bipartite graph configuration. With the current routing algorithm, if there are *N* total nodes, one needs *N/2* horizontal rows and *N/2* vertical columns to implement this structure in the FPAA's routing fabric. Figure 37 illustrates how a 4x4 graph (*N*=16) is implemented on the FPAA routing fabric. The floating-gate transistors, when programmed to conduct, will make a connection between the horizontal and vertical lines. If a transistor is turned off, then this represents an obstacle.

A non-floating-gate pFET transistor is used to input current into the grid. This nominally represents where the robot is located. A diode configured nFET transistor is used as a current sink in the grid. This is placed at the goal node. Next, the algorithm commences with measuring the voltages of the neighbors of the input current location. The neighbor with the lowest voltage is the place where the robot should be moved. A flow chart of the algorithm is found in Figure 38. Performance was increased when the input pFET was actually moved only every five iterations on the path planner [48]. To calculate a final path solution the FPAA planning system needs to maintain an adjacency list to define neighbor

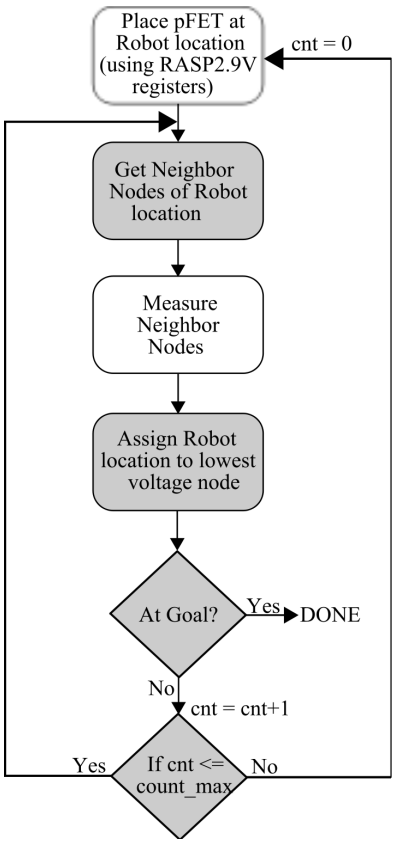


Figure 38. The basics of the FPAA Path planner algorithm flow.

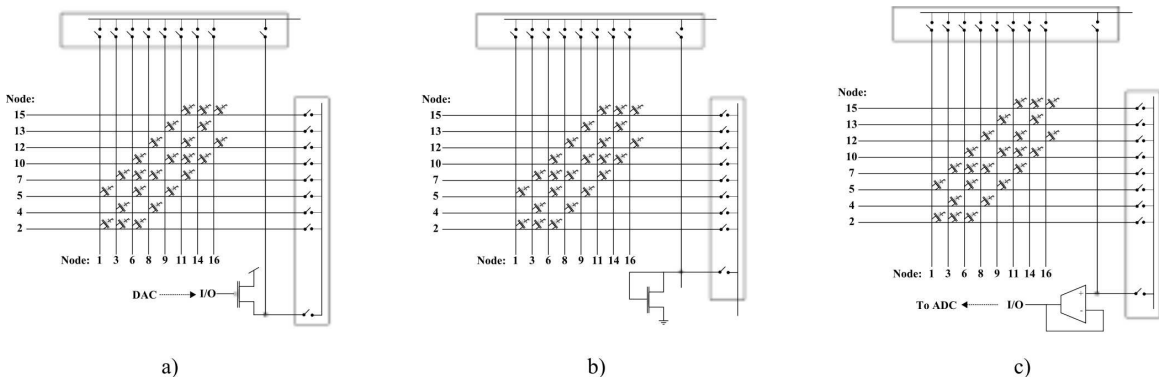


Figure 39. This figure shows how the input, output, and buffers are implemented in the RASP 2.9V FPAA IC. a) The input current to the grid is supplied by a pFET transistor. b) The current sink (representing the goal) is implemented by a diode connected nFET transistor. c) A buffer connected Operational Transconductance Amplifier (OTA) is used as part of the node voltage sensing circuitry. Multiplexers are used to place the pFET, diode connected nFET and OTA buffer on any of the nodes.

nodes. This lists, for each node, all nodes that are one step away through passable paths. This list can have the form *[source node, list of adjacent nodes]*. For example, in the 4x4 grid of Figure 20, the robot can reach nodes (1,1), (2,2), and (3,1) from node (2,1). The corresponding adjacency list would be [(2,1), (1,1) (2,2) (3,1)]. This information is combined with the node voltages read from the FPAA to choose a path. The present system implementation uses Matlab for this operation, but the operation can be moved to the microprocessor in the future. A buffer configured Operational Transconductance Amplifier (OTA) is used in the voltage sensing circuitry. The buffer helps one to measure the node voltage with the Analog-to-Digital Converter (ADC) without affecting the grid circuit. Figure 39 illustrates how the registers on the RASP 2.9v IC allow one to place the input pFET, output nFET, and OTA at specific locations in the grid. An image of how these electronic components are routed onto a RASP 2.9 IC is shown in Figure 40.

Concerning grid scalability, in this bipartite implementation of the grid on the FPAA, the grid size is limited by the number of global and horizontal lines in the FPAA routing fabric. The RASP 2.9v has the capability for 100 global rows and 150 global columns. Therefore, with the RASP 2.9v and this particular bipartite algorithm, one is limited to a grid with two hundred nodes. The limiting factor is the 100 global rows. The maximum

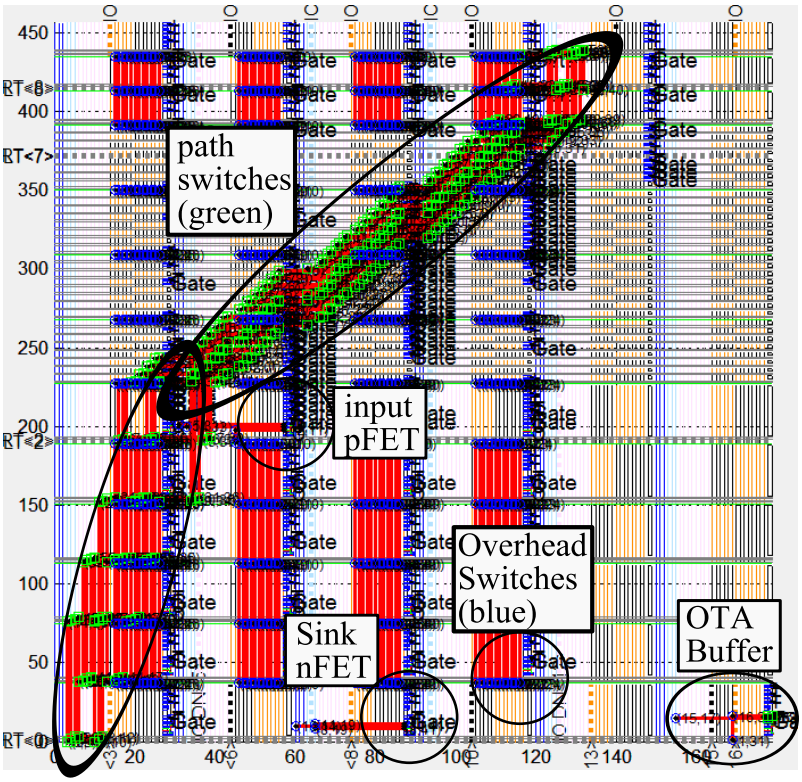
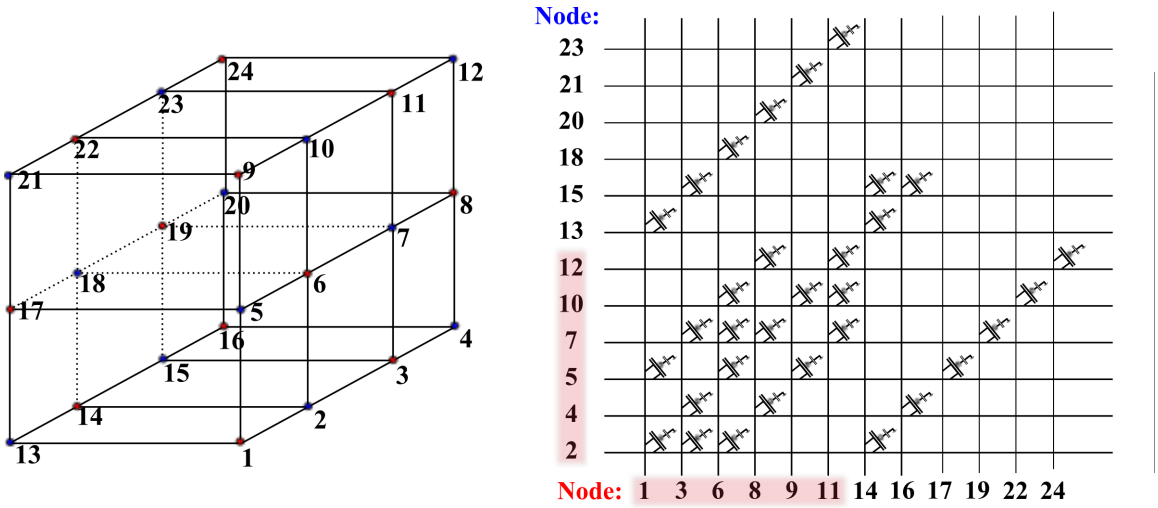


Figure 40. This shows how the input current pFET, output current (sink) nFET, path switches, and OTA buffer are routed onto a RASP 2.9V IC for a completely connected 14x14 grid. This illustration can be compared to the IC die photo in Figure 8b.

square grid is then 14x14 (N=196). Future IC designs could increase the number of global horizontal and vertical lines to allow for larger grids and also include microprocessors integrated with floating-gate routing fabric. With this integrated microcontroller many other control functions could be expected to be integrated together as an important co-design problem.

3.4.1 Software

Two new software tools were developed to support this work. These are 1) a tool to convert the obstacle map into a transistor map on the FPAA and 2) a map/obstacle visualization and modification program called the Path RAT. The tool which converts the obstacle map into a transistor map on the RASP 2.9v FPAA uses an algorithm to arrange the map and obstacles into a bipartite graph. A *bipartite graph* "consists of two distinct kinds of nodes, and all links go between nodes of opposite type [128]." Figure 37a shows one type of node as red



a) b)

Figure 41. a) Three dimensional grid space. b) Mapping of Nodes 1-12 of a onto a RASP 2.9V FPAA using a bipartite grid.

circles and the second type of node as blue squares. The figure illustrates that all links in this two dimensional grid configuration go between nodes of the opposite type. Figure 37b illustrates how the routing algorithm uses global horizontal FPAA rows for nodes of one type and global vertical FPAA rows for nodes of the second type. In theory, the bipartite grid algorithm is amenable to three dimensions. Figure 41 shows how a simple 3D grid is mapped onto the RASP 2.9v IC. The 3D planner may also be useful for cases such as a non-holonomic robot [129].

The second new tool developed for FPAA path planning is the Path Routing Analysis Tool or, *Path RAT*, in Figure 42. This was developed to allow the user to visually construct a map scenario of obstacles and free paths [7].

3.4.2 FPAA Hardware Results and Analysis

The FPAA hardware was used to evaluate twenty-four different experimental cases and was 95.8% correct. The cases are generated randomly as follows. The number of obstacles in each case is a pseudorandom integer value drawn from a discrete uniform distribution in the range from 1 to 90. The location of the obstacles in each case is randomly placed using a pseudorandom integer value drawn from a discrete uniform distribution in the range from

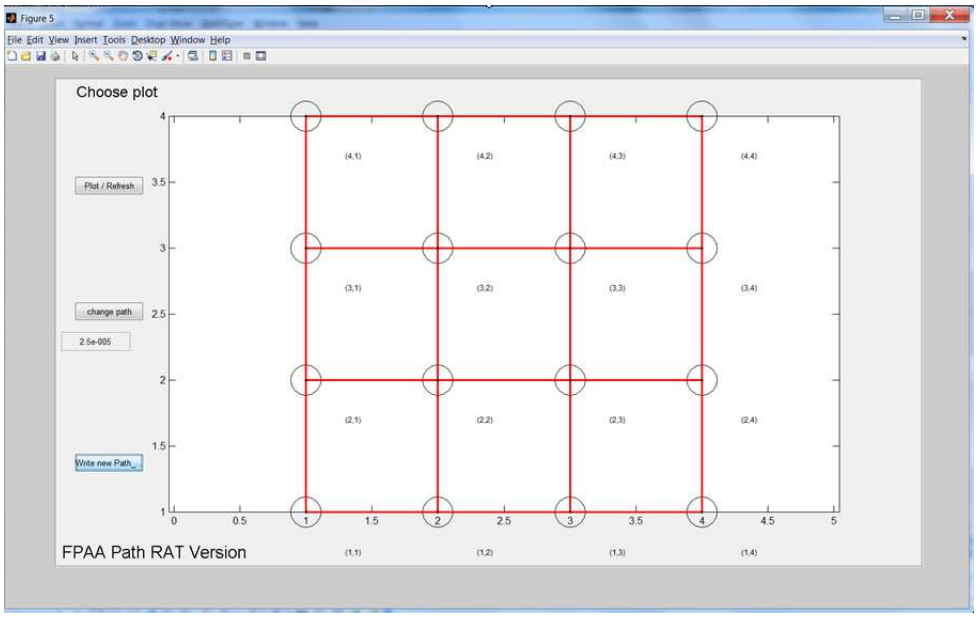
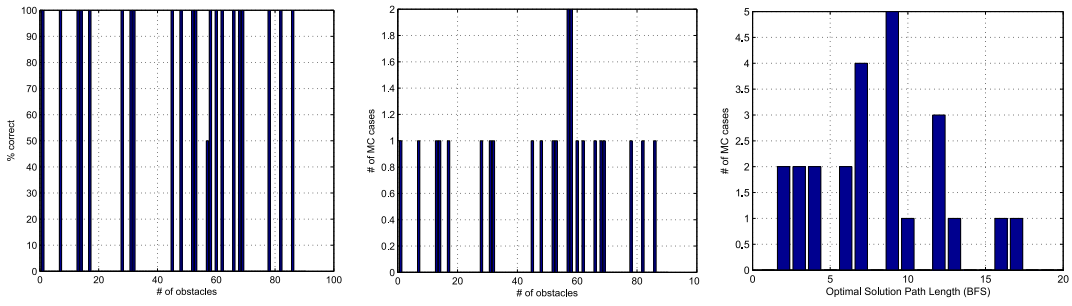


Figure 42. This MATLAB GUI allows users to visually modify grid paths by pointing and clicking on lines. Each red line represents a floating-gate connection between nodes (red=obstacle; clicking between nodes makes a red line appear OR disappear) [7].



a) b) c)

Figure 43. Statistics of the Monte Carlo scenarios used in the experiments. a) Performance as a function of the total number of obstacles. b) Distribution of obstacles in the scenarios. c) Distribution of optimal solution path length.

1 to the maximum number of edges in the grid, *((nR-1)*nC) + ((nC-1)*nR)*, where *nR* is the number of rows in the grid and *nC* is the number of columns in the grid. The node locations of the starting node and goal nodes are each drawn from a discrete uniform distribution in the range from 1 to the maximum number of nodes in the grid (196 for a 14x14 grid). The statistics of the path and obstacle scenarios are found in Figure 43. We compare the FPAA hardware results to a common search strategy called Breadth-First-Search (BFS), where the BFS algorithm is implemented on a computer processor. BFS is a *uninformed search* method. This type of strategy can only generate successor nodes and evaluate if a node is the goal. This is in contrast to an *informed* strategy such as A* (pronounced A-star), where heuristics are used to generate successor nodes that are more promising than others. BFS is complete (assuming a finite depth of the solution) and is optimal for the specific case when all paths have the same cost [1].

This path planning problem can be formulated as a tree search problem. These problems are typically evaluated with four metrics: Time Complexity, Space Complexity, Completeness, and Optimality, [1]. Time Complexity is typically measured by the number of nodes generated [1]. Space Complexity is measured in terms of the maximum number of nodes stored in memory [1]. Completeness ensures that a solution is found if it exists and also must ensure that that if a solution does not exist then the algorithm terminates and says so. Optimality ensures that the *best* solution is found if it exists.

3.4.2.1 Time Complexity

Time Complexity is typically measured by the number of nodes generated [1]. Time Complexity is not as simple as number of nodes generated with the FPAA. There are three items to consider when calculating the total time cost of the FPAA planner: FPAA grid programming time, solution computation time, and time to read the solution from the grid. Each of these are addressed below.

Table 4. Grid Programming Times according to path type

	Type 1	Type 2	Type 3
Measured Erase and Initialize grid (sec)	5.52	5.52	5.52
Measured Program time per path (sec)	0	0.059	3.96
Expected Program times (sec) [31]	0	0.001	.050

3.4.2.1.1 FPAA Programming Time Measurements Programming a grid map onto the FPAA is done in two main phases: First, the FPAA is erased and prepared for programming. Second, the new map is programmed onto the FPAA. With our current software, it takes approximately 5.52 seconds to erase and prepare the FPAA for programming. This amount of time is independent of the size of map that will be programmed. The time needed to program the map is a function of two parameters: size of map and type of paths which connect the nodes on the map. There are three types of paths that we will consider. Type 1: completely impassable paths, Type 2: completely passable paths, and Type 3: a path which is passable, but with some degree of difficulty. This may be due to terrain such as sand, an incline, etc. The programming times for each of these paths is summarized in Table 4. As an example, if one has a 3x3 map with all completely passable paths, the total programming time = 5.52 + (12*0.059) seconds. The first number is the erasing and preparation phase (a constant), and the second number represents 12 total edges between nodes which each take 0.059 seconds to program. As the state of the art in floating-gate programming advances, these times are expected to decrease.

In addition to the path switches, some extra routing switches are also needed when using the routing algorithm and RASP 2.9v IC. Most of these extra switches are used to create global lines out of local lines. In theory, this overhead number could change according to grid size. For simplicity, however, the current routing algorithm activates all overhead switches in the global columns that are used.

3.4.2.1.2 Solution computation time The computation time for the FPAA is based on the time it takes for all of the grid's node voltages to settle to steady state in response to a current step input. Figure 44 shows measured data showing the voltage gradients for two

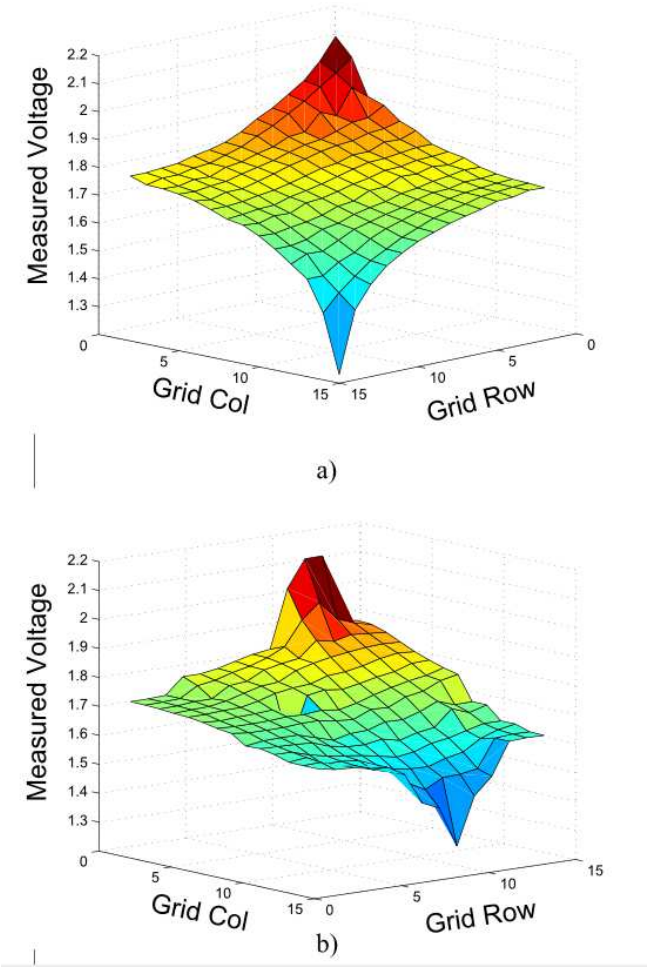


Figure 44. a) Measured Steady State FPAA hardware results for a 14x14 grid with no obstacles. b) Measured FPAA hardware results for a 14x14 grid with obstacles. Input current at 141, Sink node at 126.

14x14 grids.

The computation time for the grid based FPAA planner is approximately 0.245ms. Figure 45b shows the transient response for four nodes along the diagonal of a fully connected 14x14 grid. As shown in Figure 45a, a step input voltage was placed on the pFETs gate at node $(1, 1)$ and this implemented a step input current to represent the robot's location at this node. A current sink was implemented at node $(14, 14)$ to represent the goal. On grids of this size, the sensing capacitance is the limiting factor on the transient convergence. The transient settling time as a function of grid size was characterized in Figure 45c. The trend is that smaller grids exhibited a slower settling time.

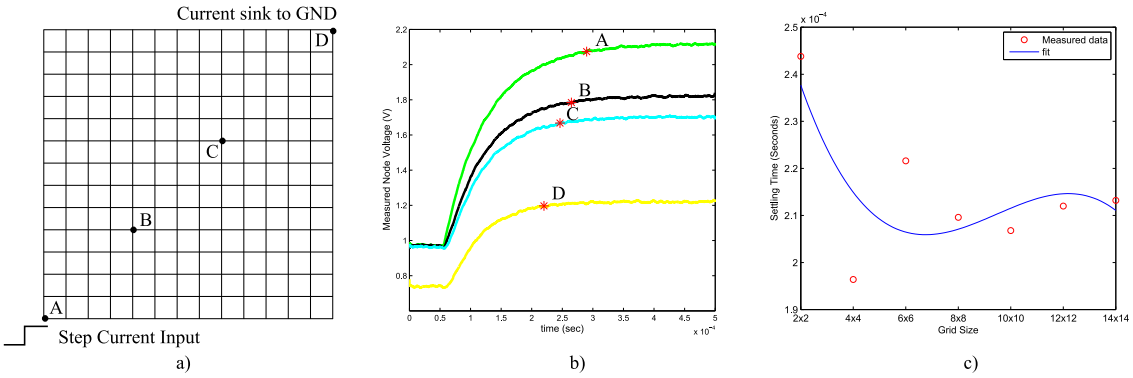


Figure 45. Measured transient responses for node voltages. a) Experiment setup: a step input voltage was asserted on the gate terminal of the input current pFET at node A (1,1). This implemented a step input current to represent the robot's location at this node. A current sink was implemented at node D (14,14) to represent the goal. b) Settling time as a function of position on the 14x14 grid. c) Settling time for the input node as a function of grid size.

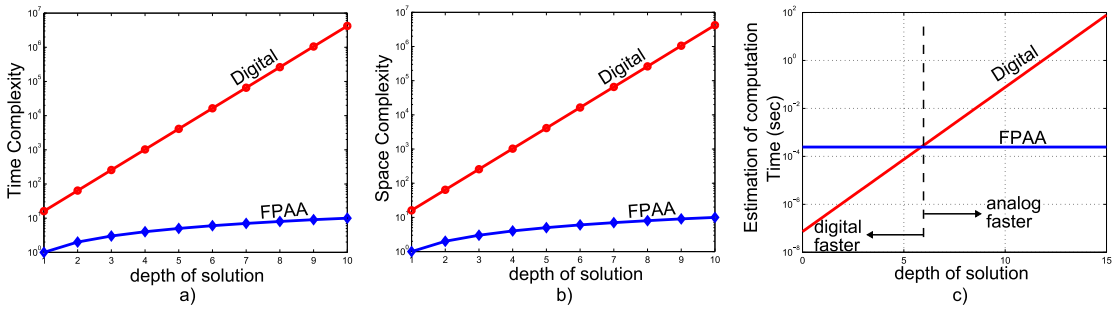


Figure 46. a) Comparing the Time Complexity of the FPAA to BFS. b) Comparing space complexities of the FPAA to BFS. c) Comparing Computation Time of the FPAA to an estimate for BFS.

3.4.2.1.3 Solution access time In the FPAA, once the nodes have settled, the solution is found by reading *b*d* nodes, where *d* is the depth of the goal node, and *b* is the branching factor. The branching factor is 4 for a two dimensional grid where an agent can move up, down, left and right. Since *b* is a constant, the asymptotic complexity of finding a solution is $O(d)$ [130]. For comparison, BFS has Time Complexity of $O\left(b^{d+1}\right)$, where the branching factor $b = 4$, and d is the depth of the solution [1]. Figure 46a compares Analog-to-Digital Time Complexity as a function of solution depth.

3.4.2.2 Space Complexity

Space Complexity is measured in terms of the maximum number of nodes stored in memory [1]. Unlike BFS which must hold in memory every node that is generated, the FPAA

Table 5. RASP29V: Comparing FPAA to BFS (where *b* is the branching factor, *b*=4 for Figure 46), and *d* is the depth of the goal node [1].

Criterion	FPAA	Breadth First
Complete?	No	Yes
Time	$O(d)$	$O\left(b^{d+1}\right)$
Space	$O(d)$	$O\left(b^{d+1}\right)$

algorithm only needs to hold the path in memory. Therefore, the Space Complexity of the FPAA is $O(d)$. BFS worst case Space Complexity is $O\left(b^{d+1}\right)$, where b = 4 and d is the depth of the solution [1]. Figure 46b compares Analog-to-Digital Space Complexity as a function of solution depth. Table 5 summarizes the Time and Space Complexity comparisons between the FPAA and BFS [1].

3.4.2.3 Completeness

In 1985, Khatib [74] was one of the first to combine the ideas of real time path planning and potential fields. One of the drawbacks to this method is that it is *not complete* because local minima in the search space may lead to solutions which do not end in the goal. One of the earliest references to using Laplace's equation for path planning is Connolly's work [118]. This method eliminates the local minima problem of potential fields. Harmonic functions are solutions to Laplace's equations. Harmonic potential fields are explored to eliminate local minima of potential fields, [131, 132, 119]. Tarassenko, et. al. build upon Connolly's work, [118, 48]. First they propose using Neumann boundary conditions (instead of Dirichlet) in an effort to overcome problems associated with the measurements becoming too small to distinguish the best path (bad dynamic range). "Under certain assumptions regarding the boundary conditions, a path planning scheme using harmonic functions is complete up to the approximation of the environment [119]." The FPAA based grid programmer is statistically complete to the extent that our empirical data demonstrates it will find a solution if one exists approximately 95.8% of the time. Also, the FPAA will indicate that a solution is not possible if one does not exist. If a solution does not exist, then there should be very little current draw out of the input pFET. Therefore, one can set a current

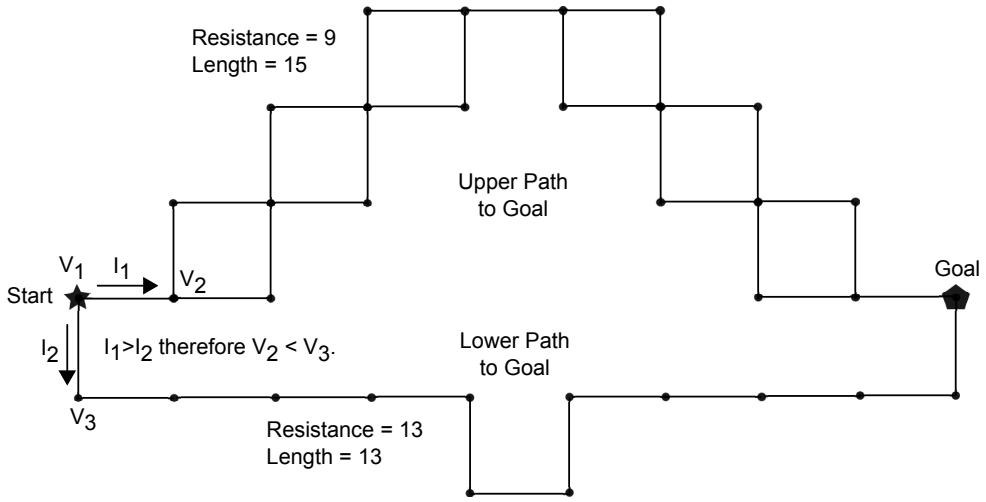


Figure 47. This figure illustrates that using a node's neighbor node voltages to choose the path does not always result in an optimal solution.

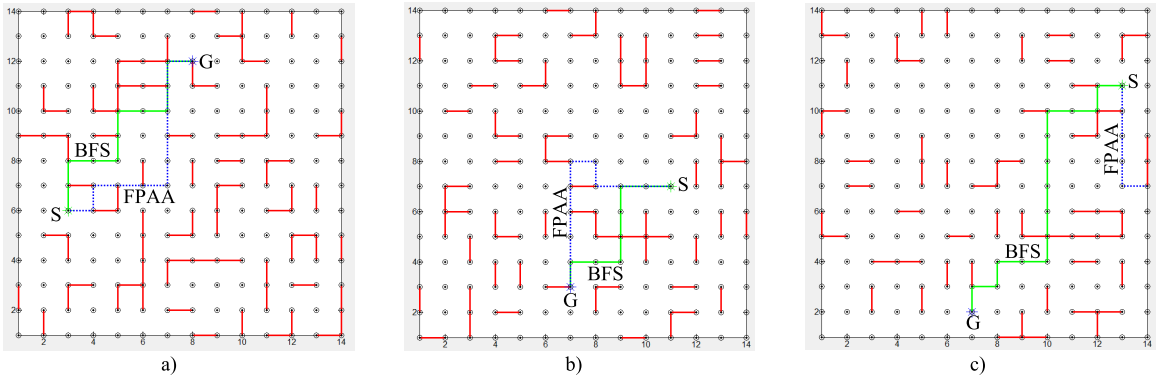


Figure 48. Measured results from FPAA compared to BFS. Red line edges represent obstacles. It is acceptable in this graph to pass *through* two parallel connected edges, but it is not acceptable to move *along* the red connected edges: S = Start and G = Goal. a) Optimal FPAA solution. b) Sub-optimal FPAA solution. c) Incorrect FPAA solution.

threshold, and if this is not exceeded then a solution does not exist.

3.4.2.4 Optimality

The analog planner is not guaranteed to find the optimum solution. Fundamentally, alternate parallel paths can lead the planner to a non-optimal solution. Figure 47 describes a situation where the planner will result in a sub-optimal solution. In this example, there are two branching paths at the start. If one assumes that each of the edges between nodes has the same resistance (say 1 ohm), then one may calculate two equivalent resistances between the start and the goal. The upper path's equivalent resistance is 9 ohms. The lower path's

equivalent resistance is 13 ohms. Ohms law dictates then that I_1 is larger than I_2. If this is the case then V_2 is lower than V_3. Since the algorithm follows the path of greatest current (i.e. largest voltage drop), the selected path would proceed along the upper path to the goal. This is a sub-optimal solution because the shortest path from Start to Goal along the upper path is 15 edges. This is a longer path when compared with the 13 edge length path along the lower path. In the experiments run, however, the FPAA solution was optimal in 20 of the 24 cases (83.3%). Figure 48b shows one of the sub-optimal cases. Finite measurement capability of the measurement circuitry may also lead to sub-optimal solutions.

3.4.2.5 Calculation Time Estimate

Ideally, one would like to compare the actual solution times of the digital and analog solutions and not just operation numbers like Time Complexity. As an estimate, assume that the BFS algorithm is being executed on an a processor such as the ATMEL ARM7TDMI RISC processor operating at 55MHz max clock speed. Further assume that the solution is at the deepest solution of the grid. If one multiplies the BFS Time Complexity number by the inverse of the ARM7 clock then we can have a crude estimate of the digital computation time: $O\left(b^{d+1}\right) * \left(\frac{1}{55Mhz}\right)$ where b = 4. To estimate the computation time of the FPAA, we use the worst case transient settling time from Figure 45c. A comparison plot is shown in Figure 46c. This plot estimates that the FPAA will be quicker at solving plans where the solution depth is greater than 6. This corresponds to the deepest solution of a 4x4 grid.

3.4.2.6 Power Costs

The power used by this system can be split into three categories: grid (map) programming power, solution finding power, and solution extraction power. We claim that the power used by the FPAA to arrive at the solution is much less that that of a digital solution. We can not make appropriate claims however of the grid programming power and solution extraction power, because we have not optimized our system for these problems. The embedded system which interfaces with Matlab uses current on the order of 400mA.

These FPAA planning results can be compared and contrasted with an FPGA implementation of the Breadth-First-Search algorithm in [133]. According to their experiments, mostly all of the measured solution times for the hardware were at least two orders of magnitude faster than their software solutions. (The processor speed for the software solutions is not clear, however). A 10x10 grid solution in FPGA hardware took about 2.35us. A 30x30 grid in FPGA hardware took 28.2us. This is compared to their 10ms software solution. There was a tradeoff, however, with respect to grid size. For smaller grids on the order of 30x30, the software implementation was preferable because the time (although still longer than the hardware implementation) is small, and the implementation is easier in software than in hardware.

3.4.3 Summary

In this chapter, we continued to build upon the idea that Laplace's equation and analog circuits can be used for path planning and are adding to the existing research which combines analog VLSI and robotics [134]. This work is new for a few of reasons: First, it provides an extensive amount of measured data representing different map scenarios from a fabricated AVLSI IC. Second, our analog circuit implementation is different from the existing literature because it is implemented on a reconfigurable analog IC that uses floating-gate transistors which provide, among other things, a non-volatile way to store the environment map. Finally, this work has started to quantify the performance gain for using an analog solution instead of a digital one in terms of Time and Space Complexity.

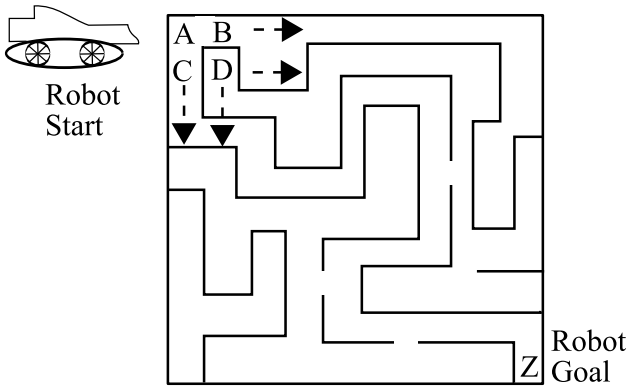
CHAPTER 4

NEURON PATH PLANNING

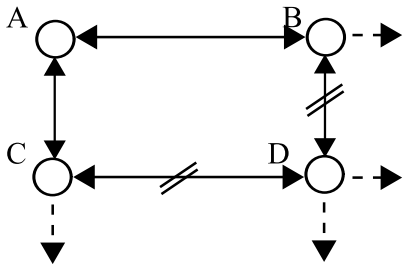
Neuromorphic engineering is an interdisciplinary field which combines concepts from fields such as biology, neuroscience, computer science and engineering [135] [136] [137] [121] [138]. The goal of this field is to design systems that are based on the principles of biological nervous systems. Path planning is a critical task for robots, autonomous vehicles, animated characters, etc. Figure 49 is a cartoon showing the ultimate goal of the problem being addressed in this chapter, namely how to use a neuromorphic approach and neuron array integrated circuit (IC) [10] to plan a path for a Micro Aerial Vehicle (MAV) (or similar power constrained ground or sea robot) through an environment in an effort to conserve its limited battery resources. The IC used for the results in this work, Figure 8c, uses biologically realistic transistor based models which operate as neurons operate, however not necessarily how the brain does path planning. The IC has 100 Neurons and 30,000 synapses. It was constructed in a 0.35 micron process and the die size is 5x5 mm. Floating-gate transistors are used as the synapse elements. The floating-gate synapse transistors are used to create the programmable routing. A synaptic weight is stored by an adjustable charge on the gate of the floating-gate synapse transistor. Multiple synapses are connected to a dendrite by adjusting the weights on the synapse transistors which are connected in parallel to the dendrite wires. Unused synapses are not connected internally to the path planning circuit. The floating-gates for these transistors are not set to conduct. The system uses Address Event Representation (AER) to record spike data from the neurons. Neuron Elements include: soma, dendrite, synapses, and axons. Path planning can be summarized with the following three tasks given that states, actions, an initial state, and a goal state are provided. The robot should:

1. Find a sequence of actions that take the robot from its Initial state to its Goal state

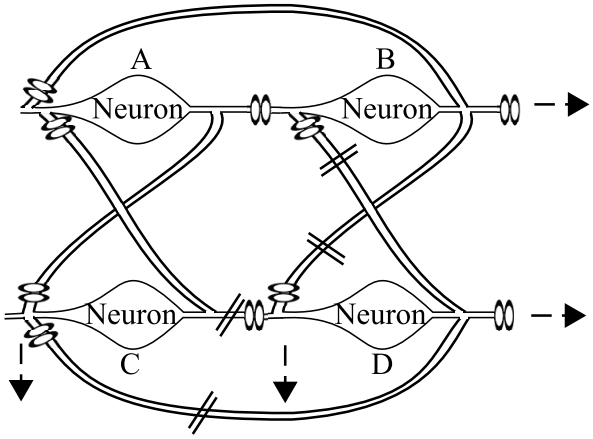
2. Find actions that take the robot from *any* state to the Goal state



a) Discretized locations in maze



b) Grid representation of maze



c) Neuron representation of maze

Figure 49. The goal of this research: to use a reconfigurable neuron Array IC to plan a path for small robot from point A to point Z through an environment. a) Maze environment which is discretized into grid points (shown here as A,B,C,D,...). b) A simplified grid representation of the maze in a. Note that some edge connections are not active between nodes (marked by two hash marks). This represents a wall. c) This work uses bidirectional connected neurons to implement the edges between nodes. As in b, some axon-dendrite connections are not made (marked by two hash marks) for the neurons representing nodes separated by a wall.

3. Decide the *best* action for the robot to take now in order to improve its odds of reaching the Goal

In a comprehensive survey of autonomous rotorcraft unmanned aircraft systems, Kendoul describes path planning methods and algorithms which represent the "most used and practical methods, with a particular focus on works with experimental results [139]." According to this survey, Navigation strategies used in Rotorcraft Unmanned Aircraft Systems (RUAS) include, among others, Road Maps (RM) and Potential Fields (PF) [139].

In RM methods, graphs are constructed using nodes to represent robot positions and edges are used to represent paths between positions. A search algorithm is then often used to plan a path among the nodes [139]. Among other groupings, search algorithms used in path planning can be grouped into *Uninformed* strategies and *Informed* strategies. Informed strategies differ from Uninformed strategies in that they have a method to guide the search to make it more efficient. Breadth-First-Search (BFS) and Dijkstra's algorithm are examples of Uninformed search strategies. BFS is optimal when path costs are equal, Dijkstra's is optimal with non-equal path costs [1]. BFS has also been called *brushfire* [140] , or *grassfire* [21] , since it resembles the way fire progresses in a dry grassland [140] . Technically expressed, the grassfire transform "is simply breadth-first search implemented in the constrained space of an adjacency array [21]." An early grassfire algorithm is called NF1 [141]. NF1 is described as a wavefront expansion algorithm which calculates a navigation function for each point in the workspace [141]. Although this results in a path which is shortest from the robot to the goal, a problem with the NF1 algorithm is that it creates paths that come close to obstacles [141]. Further, the wavefront propagation algorithm can also be considered a specialized version of Dijkstras algorithm that optimizes the number of stages to reach the goal [142]. Informed strategies include best-first search strategies like Greedy best-first search and A* (pronounced A-star) [1]. These methods use a heuristic to guide the search. A* is an extension of Dijkstras algorithm which tries to reduce the number of explored states by using a heuristic [142]. An example of a heuristic in a two

dimensional grid map representation is the straight line distance between a point and the goal.

In PF methods, the geometry of the workspace and obstacles is often first discretized into grids in what is called an *Approximate cell decomposition*. The obstacle grids are programmed to produce a repulsive force and the goal grid represents an attractive force. A force vector is calculated for each free space point in the workspace. This force vector is a sum of the attractive and repulsive forces acting on it. One may follow the force vectors at each point to find the path from each point in the grid to the goal. A problem with this method is local minima. In practice, one RUAS planning system uses Laplace's equation with binary occupancy grid for a potential field approach. Their numerical solution is found using a 1.8GHz Pentium M with 2MB L2 cache. A C implementation of their procedure had a constant runtime of 0.07s for a three dimensional grid (64x64x32) and 0.6s for a (128x128x64) grid [143, 144].

In some sense, wavefront or grassfire planners have characteristics of both RM and PM methods. FPGAs have been used to implement planners based on wavefront or stencil based gradient calculations [116, 12]. The authors describe processing 33 maps with a resolution of 1024x1024 per second on a midsize Virtex-5 FPGA [12]. Analog VLSI has previously been used for path planning, however much of this work involves Laplacian based potential field approaches and not analog VLSI *neuron* based approaches like in this chapter [44, 53, 7, 2, 72].

A motivating reason for using the neuron array IC for path planning is for better Time Complexity (one of four main performance metrics used to compare path planners) and the *potential* for lower power processing capabilities when compared to digital implementations. The Time Complexity advantage is driven by the Neuron IC's capability to propagate signals in parallel. This may be a significant advantage for MAVs, ocean gliders or other robot applications where the power budget for Guidance, Navigation, and Control

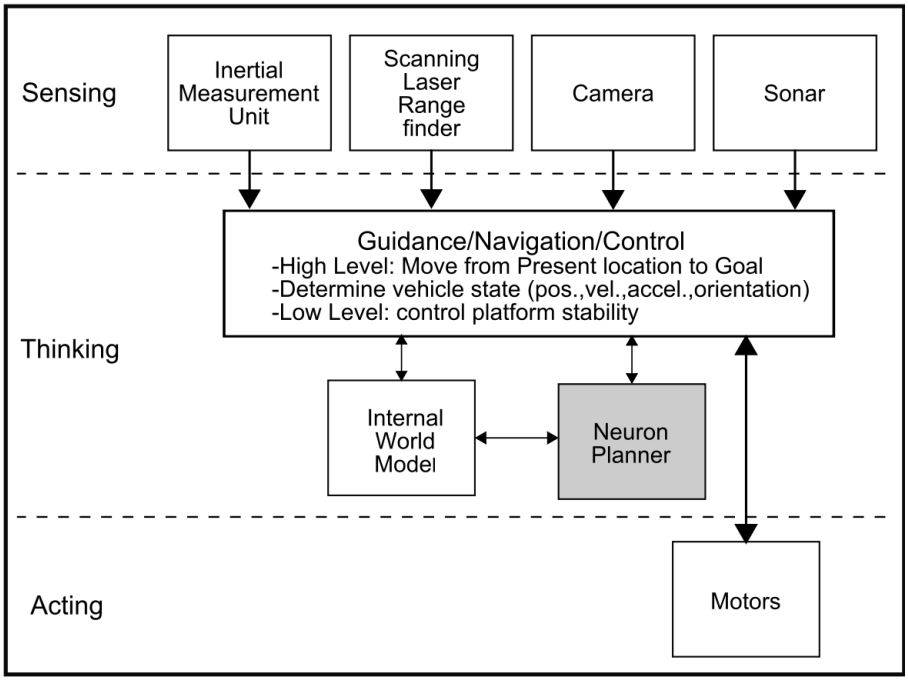


Figure 50. The path planning block is one subsystem needed for an autonomous vehicle operating in an unknown environment. There are three broad categories of subsystems on the vehicle: Sensing, Thinking, and Acting. The neuron IC and planning block fits into the Thinking category [8, 9].

is limited [102, 29]. A key feature of the neuron processor is that it performs the wave-front expansion in parallel. Early discussions for parallelizing the wavefront computation involved having a virtual processor for each point in the grid [13]. Although this "brute force" parallelism could waste power, it was suggested that this was one of the only ways for parallelism [13]. In our neuron implementation, the entire numerical potential field does not need to be calculated in order to find the solution. Once the wavefront reaches the start, the computation is done. One may use best-first-search for finding the path through the wavefront produced gradient. In best-first search, the node "selected for expansion is based on an evaluation function, f(n). Traditionally, the node with the lowest evaluation is selected for expansion, because the evaluation measures distance to the goal [1]." In our case, the evaluation function is extremely accurate and leads us to expand the best node. The heuristic function in our case is the time it took for the wavefront to travel to each of a nodes neighbors.

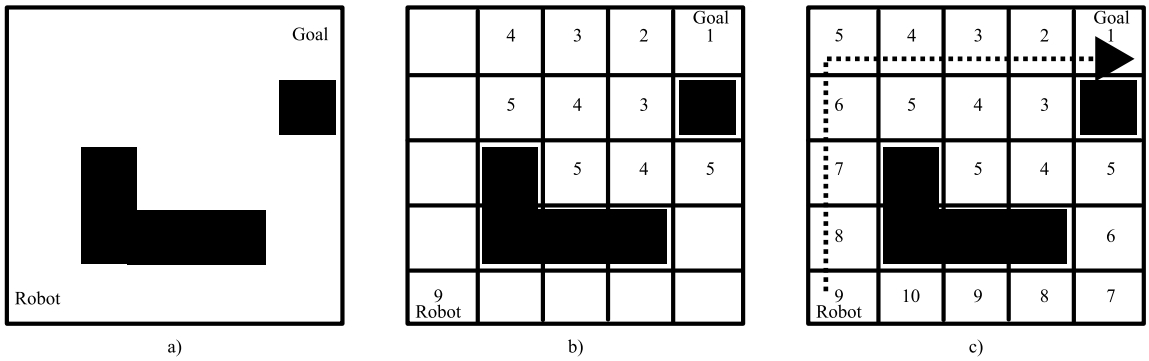


Figure 51. This figure illustrates the basics of the wavefront planner. a) Scene where the robot is trying to reach the goal while avoiding obstacles and traveling in the shortest path. b) The grid is discretized and a wavefront is propagated away from the goal. The number represents a time stamp of when the wavefront reached the square. c) The shortest wavefront reaches the goal in 9 moves.

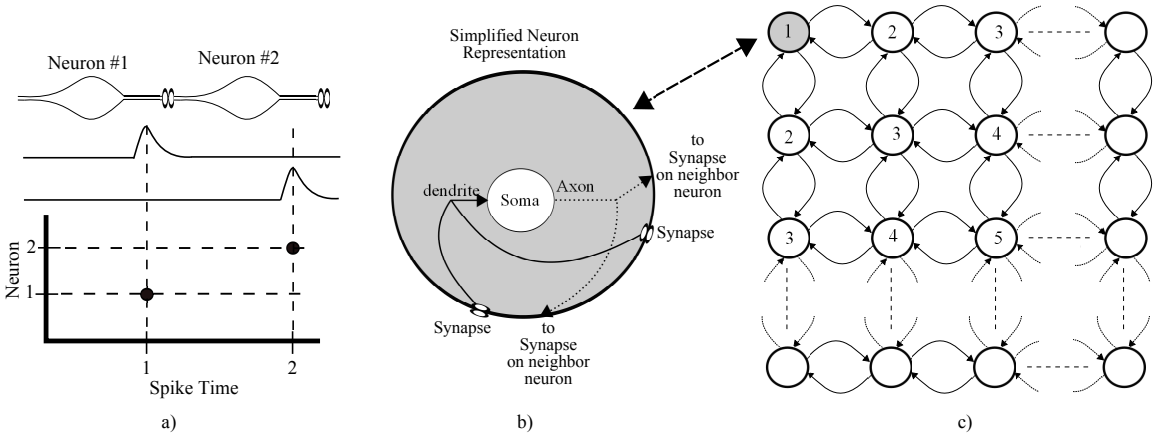


Figure 52. This figure shows how the neurons of a fully connected grid pass spikes. a) A signal is originated in neuron 1 and this causes neuron 2 to fire at a later time. Dots in the raster plot show when the spikes occur. b) The outer large circle represents the map grid location and is implemented using a neuron with soma, dendrite, axon, and synapse components. The dendrites in these neurons are represented by wires. The synapse strengths are set with floating-gate transistors. c) This represents a fully connected grid. The number in the neurons show how the wavefront of a signal initiated in the upper left of the grid propagates. The numbers represent increasing time stamps of the propagating wavefront. The 1 represents time 1, the 2 in the neurons show which neurons fire at time 2, etc.

The path planning block is one of many subsystems needed for an autonomous vehicle operating in an unknown environment [145, 146, 9]. There are three broad categories of subsystems on the vehicle: Sensing, Thinking, and Acting. The neuron IC and planning block fits into the *Thinking* category, Figure 50 [8, 9]. Information on implementing the other subsystems is beyond the scope of this chapter. This work assumes that the other subsystems are able to sense and map the environment and represent the map in an NxM grid (internal world model). The map representation is programmed onto the neuron array IC using an embedded system described in Chapter 2.

In summary, analog path planning is explored because it represents a potential decrease in Time Complexity and a potential power savings. This neuron array path planning IC is useful because this reconfigurable AVLSI IC provides circuit tune-ability and flexibility that custom ASICs often do not provide. The general idea of this planner is a grassfire [141, 21, 147] or wavefront planner [148, 149]. A spike based neuron inspired wavefront planner and simulation results were presented in [45]. This work in this chapter is new because it provides successful path planning results and analysis of a wavefront planner implemented on biologically inspired AVLSI hardware.

The wavefront based setup of the neuron analog planner is presented in Section 4.1. Hardware results and analysis are presented in Section 4.2, and conclusions are made in Section 4.3.

4.1 Wavefront Neuron Analog Planner Setup

Our neuron IC implements a standard wavefront planner using biologically realistic neurons [11]. An early paper presenting a wavefront planner is [150]. In the standard wavefront planner algorithm, the goal node is given a value of *1*, and its neighbors which are not obstacles are given the value of *2*, and all the non-obstacle neighbors of *2* are given the value of *3*, etc. This is continued until the start point is reached. To find the path, the algorithm starts at the start point, and takes steps in the direction of decreasing node value, until the

goal is reached [151, 141]. This process is illustrated in Figure 51.

Neurons which are connected together in the IC pass spikes to their neighbors. Like the standard wavefront planner, this neuron IC initiates a wave at the goal neuron and this in turn excites its non-obstacle neighbors. Time stamps are recorded at the point when each neuron is excited. The time stamps when each neuron fires for the first time are used to back out a solution.

Figure 52 shows how the neurons of a fully connected grid pass spikes. In Figure 52a a signal is originated in neuron 1 and this causes neuron 2 to fire at a latter time. Dots in the raster plot show when the spikes occur. A more detailed view of each of the neurons in our model is shown in Figure 52b. The dendrites in these neurons are represented by wires. Figure 52c shows a fully connected grid. The number in each of the neurons show how the wavefront of a signal initiated in the upper left of the grid propagates. The numbers represent the time stamps indices of when each of the neuron fires.

This technique is similar to the radar path planner presented and simulated in [45]. Our research furthers this technique by implementing the wavefront planner using analog neuron circuits implemented in actual silicon hardware. Using analog neurons to pass signals can be compared to other signal propagation methods in analog such as Resistor-Capacitor integrator delay line circuits and delay lines formed by operational transconductance amplifier (OTA) based follower-integrator circuits, Figure 53 [121] [122].

Figure 55 shows how a maze environment is modeled using the neuron IC. Figure 55a shows the example maze problem. The robot is located at Node 22 and the goal is at Node 77. Figure 55b shows the bidirectional connections between neurons that represent free paths in the maze. Figure 55c shows how the connections are made in the neuron IC. A system called Address Event Representation (AER) is used to record the time at which spikes occur in each neuron [152]. The 100 waveforms represent the 100 neuron outputs, and the grey dots represent synaptic connections. This figure shows how the outputs of the neurons are connected to the dendritic inputs of neurons [10]. Figure 56 shows details of

the solution for Figure 55. A wavefront was initiated at the goal (Node 77) and propagated throughout the neuron grid. Figure 56b shows a raster plot of the solution nodes. Neuron 77 causes neuron 76 to fire, which causes neuron 75 to fire, etc. Figure 56c shows a zoomed raster plot of the solution nodes. A linear fit shows that the wavefront propagation has a nearly constant velocity of 1.091 neurons/ms.

A flow chart of the algorithm solution for Figs 55 and 56 is found in Figure 57. The robot is located at Node 22. The circles connected to Node 22 represent its non-obstacle neighbors. The numbers inside the circle are the timing index numbers from the AER circuitry. Specifically, these numbers capture the time when the neighbors experienced their *first* spike. The neighbor that fired first before Node 22 fired is Node 23. The robot should therefore move to Node 23 next. This represents the next move on the optimal path to the goal. Node 23 has three non-obstacle neighbors. Its neighbor node that fired first is Node 33. Node 33 is then the next path on the way to the goal. This procedure continues until the full path is found.

One of the features of this algorithm is that the entire path solution can be found by only a single excitation at the goal. Figure 59 shows a couple of ways that the planner can be incorporated into a robot system. In method 1, the entire path to the goal is planned each time the robot moves. In method 2, only the *next* location is found. If the robot's environment is not static this second method may reduce the AER post-processing computations after each move.

Regarding the software tools for this IC, we used PyNN [153] to specify the neuron structure of the maze. PyNN is a Python-based network description of the maze environment.

Three specifications of the neurons' settings include: the neuron's refractory period, the velocity of the propagated signal from neuron to neuron, and the synaptic strengths. These three items can be adjusted to affect performance.

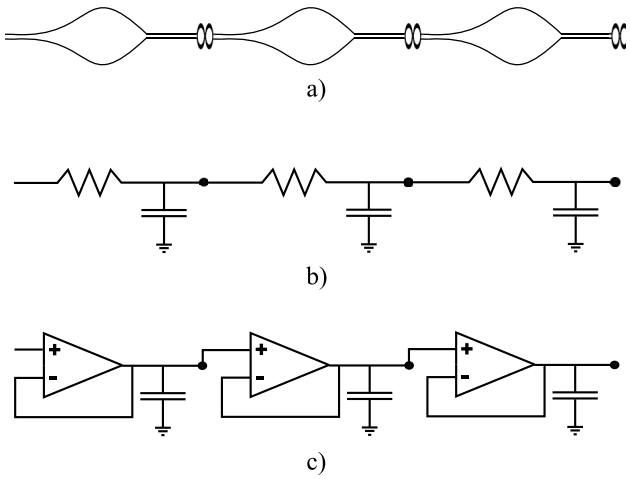


Figure 53. This figure shows how signal propagation velocity can be modeled using various methods: a) neuron, b) diffusive, and c) hyperbolic

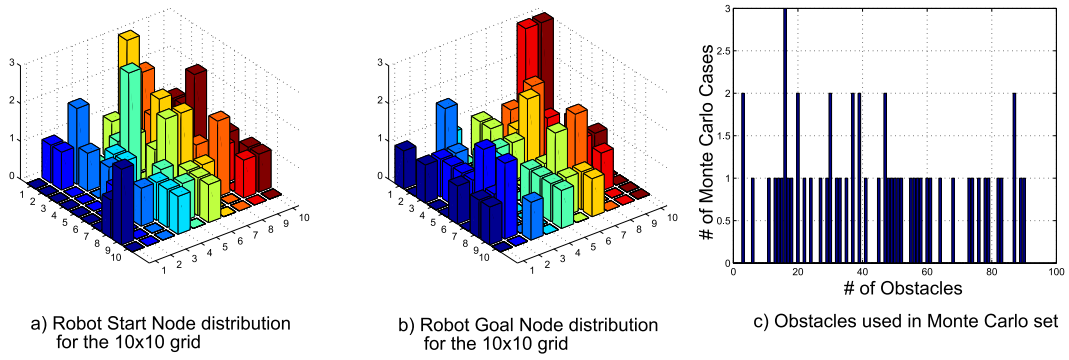


Figure 54. Statistics of the Monte Carlo scenarios used in the experiments. a) Distribution of the Robot start states, b) Distribution of the Robot goal states, and c) Distribution of number of obstacles used in the experiments.

4.2 Neuron IC Hardware Results and Analysis

The neuron IC was used to evaluate 55 different experimental maze cases and was 100% correct. The cases were generated randomly as follows. The number of obstacles in each case was a pseudo random integer value drawn from a discrete uniform distribution in the range from 1 to 90. The location of the obstacles in each case was randomly placed using a pseudo random integer value drawn from a discrete uniform distribution in the range from 1 to the maximum number of edges in the grid, $((nR - 1) \cdot nC) + ((nC - 1) \cdot nR)$, where nR is the number of rows in the grid and nC is the number of columns in the grid. nR and nC were both 10 for our scenarios. The node locations of the starting node and goal nodes are each drawn from a discrete uniform distribution in the range from 1 to the maximum number of nodes in the grid (100 for our cases). The statistics of the path and obstacle scenarios are found in Figure 54.

Path planning problems are typically evaluated with four metrics: Time Complexity, Space Complexity, Completeness, and Optimality [1]. Time Complexity is a measure of the time needed to find a solution [1]. Space Complexity is a measure of the amount of memory needed for the search [1]. Completeness ensures that a solution is found if it exists and also must ensure that that if a solution does not exist then the algorithm terminates and says so. Optimality ensures that the *best* solution is found if it exists. Each of these criteria are discussed in the following sections.

4.2.1 Time Complexity

Qualitatively, Time Complexity is a measure of how long it takes to find a solution [1]. It is often measured by the number of nodes generated when the algorithms are implemented in digital computers [1]. This is a reasonable measurement for a system that sequentially evaluates the nodes of a grid. For systems that can search multiple grid nodes in parallel, such as the Neuron IC planner, one may use a different measurement. In the Neuron IC, if we assume that all edges between nodes have the same weight, then the propagation of the signal is uniform in the grid and effectively all of the nodes on the leading edge of the

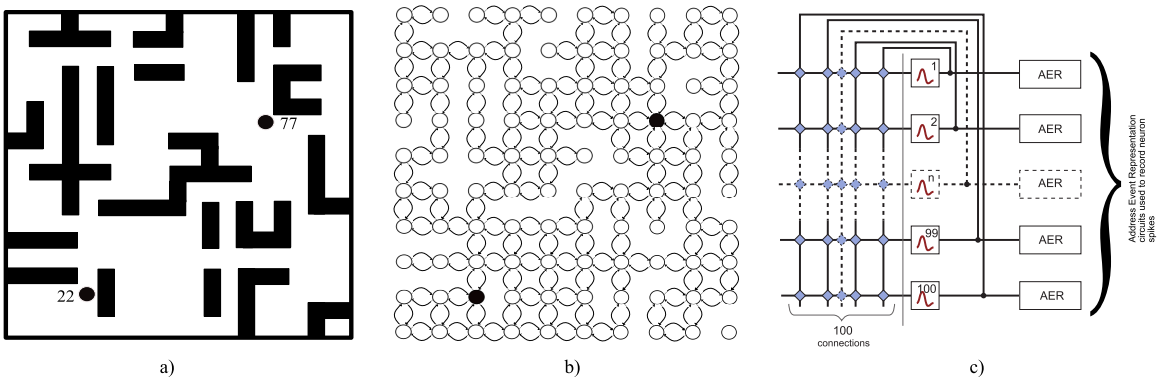


Figure 55. This figure shows how the neurons of a specific maze case are configured to represent free paths and obstacles. a) The maze which will be represented with neurons. The goal is at Node 77, the robot is at Node 22. The IC will plan an optimal path between these two nodes. b) This is how the neurons are configured for the maze case in (a). There are bidirectional paths between neurons where free paths exist and no connections where obstacles exist. c) This shows how the neurons are connected within the IC. Address Event Representation (AER) circuitry enables the circuit to time stamp when each of the neurons fires [10].

wavefront are evaluated at approximately the same time. This allows one to say that the solution time is a linear function of the depth of the solution, *d*. As shown in Figure 56b and Figure 61b, each time the wavefront expands it reaches all of the neighbors at this depth in a time that is a function of the depth and the velocity of propagation. Eq (12) is the Time Complexity estimate for the Neuron IC planner. The estimate is written for an arbitrary solution depth *d*, and arbitrary neuron propagation velocity, *v*.

$$TC = \frac{d}{v} \tag{12}$$

The essence of Time Complexity is not as simple as number of nodes generated with the Neuron Array IC. There are also four items to consider when calculating the total time cost of the Neuron IC planner: Neuron pre-programming time, Environment map programming time, Time to read the solution from the neurons, and Solution computation time. Each of these are addressed below.

4.2.1.1 Neuron Pre-programming Time

Before being used for planning, the neurons need to be programmed. The neuron structures can be pre-programmed onto the IC before the autonomous agent navigates its environment.

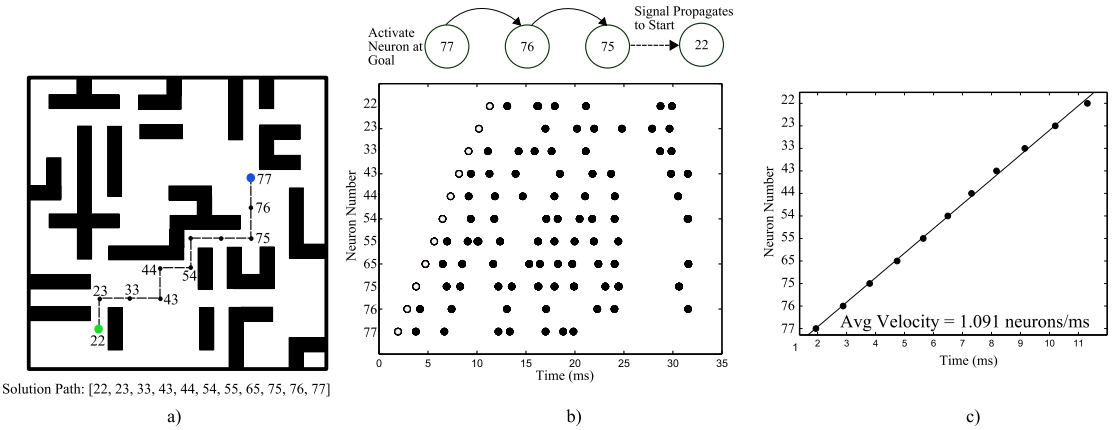


Figure 56. Results from one case that was solved on the Neuron IC. a) This is a grid environment example that the neuron IC solved. The robot is located at Node 22 and the goal is at Node 77. A wavefront is initiated at the goal (Node 77) and propagated throughout the Neuron grid. b) Raster plot of the solution nodes. Neuron 77 causes neuron 76 to fire, which caused neuron 75 to fire, etc. c) Zoomed raster plot of the solution nodes. A linear fit shows that the wavefront propagation has a nearly constant velocity of 1.091 neurons/ms.

This involves programming the biases among other settings. Programming the neurons for a 10x10 grid currently takes on the order of 5-10 minutes. As shown in Figure 58, a current starved inverter [10] circuit is used to produce a gate waveform in the synapse circuit. This gate waveform circuit and some operational transconductance amplifiers (OTAs) used for detecting spikes in the neurons also need to be programmed. The current starved inverter circuit has two floating-gate voltage settings, V_{bp} and V_{bn}. Changing these alters the gate waveform driving the synapse transistor. Details of how these are changed (programmed) can be found in previous papers [11, 10]. The velocity at which the neurons fire is a function of this gate waveform. If the velocities are changed this means modifying the gate waveform programming. Changing one gate waveform requires that all gate wave waveform circuits be erased and re-programmed. This reprogramming process takes on the order of 5-10 minutes. The cap at the gate of the transistor charging $C_{membrane}$ is a coupling capacitor between the processed output of a neuron and the floating-gate transistor device operating as a synapse. The floating-gate of the transistor can be programmed and "can be used to store a weight in a nonvolatile manner, compute a biological EPSP, and demonstrate

biological learning rules [10]." Programming accuracy can be achieved at 9 bits of floating-gate voltage [31].

4.2.1.2 Environment Map Programming Time

In addition to setting up the neurons, the IC also needs to be programmed with the environment map. The environment is mapped onto the array by setting the strengths of the synapses. Programming time for a 10x10 map (i.e. synapses in the array) is about 10 minutes. This assumes that the velocities (i.e. gate waveform circuits) are not changed. This programming time can theoretically be reduced in three ways: 1) by using a digital injection instead of precise injection programming, 2) by doing row parallel injection, or 3) by using precise programming but implementing it on the micro controller.

The velocity is set to about 1ms/neuron, but this can be adjusted to a point. Predicted jitter on the velocity programming capability is estimated to be +-1us. This is conservative estimate based on a scale factor of gate delay jitter.

4.2.1.3 Time to Read the Solution From the Neurons

The AER system records the neuron spike events. The system can operate at about 1 event per micro second. This is the limitation of the AER system. With our current neuron velocity of 1ms/spike, the AER system could therefore keep up with approximately 1000 neurons firing before it starts to lose events.

4.2.1.4 Solution Computation Time

As in [148], the search of the raster plot runs in linear time with respect to path length, and the computation time is proportional to the number of free cells in the environment. The solution time for the next move is $(pathLength) \cdot v$ ms, where v is about 1 ms in this system. If one is finding the entire path, then the time is slightly more.

4.2.2 Space Complexity

Space Complexity is measured in terms of the maximum number of nodes stored in memory [1]. As was said in Section 4.1, the neuron IC can operate in two modes as it plans the

optimal path to the goal. The first method plans the entire path to the goal, and in Method 2, only the next move to the goal is planned. These two operating strategies have different Space Complexity requirements. Furthermore, there are different memory estimates depending on the refractory period programmed into the neurons. Section 4.1 mentioned that the refractory period of the neurons is adjustable in this system. The refractory period is the time delay during which the neuron will not fire. There are different Space Complexity estimates depending on the refractory period programmed into the neurons. The following section will address the details of a long and short refractory period where a long refractory period is defined to be a time much greater than the total time for the wavefront to propagate from the goal to the start.

First, Method 1 is addressed. Method 1 finds the full solution from the start to the goal. This requires that the entire wavefront fanout be recorded in order to find the full solution. If one assumes that there is a long refractory period, then one can estimate an upper bound on the number of spikes generated as follows: As shown in Figure 61a, each time the wavefront expands it reaches $4d$ new neighbors. This is where d is the depth of the solution. Assuming that each grid has a branching factor of 4 (i.e. the autonomous agent can, in the absence of an obstacle, move up, down, left, or right), one may calculate the total number of nodes enclosed by the wavefront as follows.

$$SC_{long\_refractory} = 4 \cdot 1 + 4 \cdot 2 + 4 \cdot 3 + \cdots + 4 \cdot d \tag{13}$$

$$= 4\,(1 + 2 + 3 + \cdots + d) \tag{14}$$

$$= 4\left(\frac{d^2 + d}{2}\right) \tag{15}$$

$$= 2\left(d^2 + d\right) \tag{16}$$

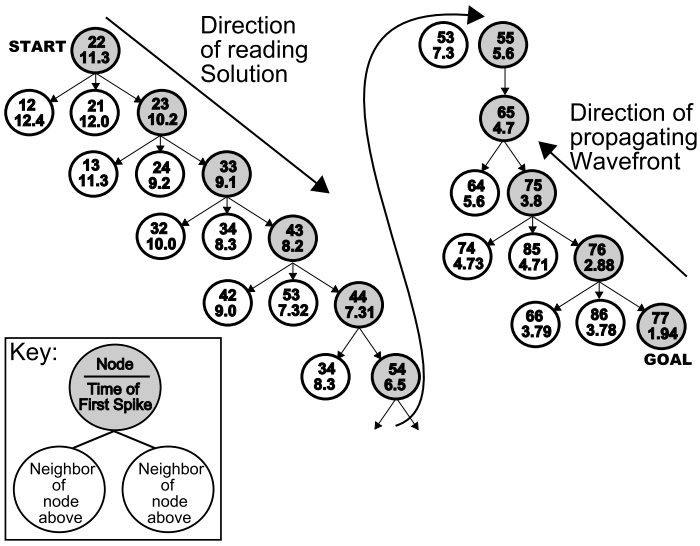


Figure 57. This shows how the solution for the maze in Figure 56 is backed out of the AER spike timing information. The rows of circles represent non-obstacle neighbors. The numbers represent the time that each node first spiked. The solution is found as follows. Node 22 has three non-obstacle neighbors (as represented by the three circles). The neighbor which fired first is Node 23 (It fired at 10.2 ms which is earlier than the other two node firings.). Thus, one may conclude that Node 23 caused Node 22 to fire, and it is selected as a node on the optimum path (in gray).

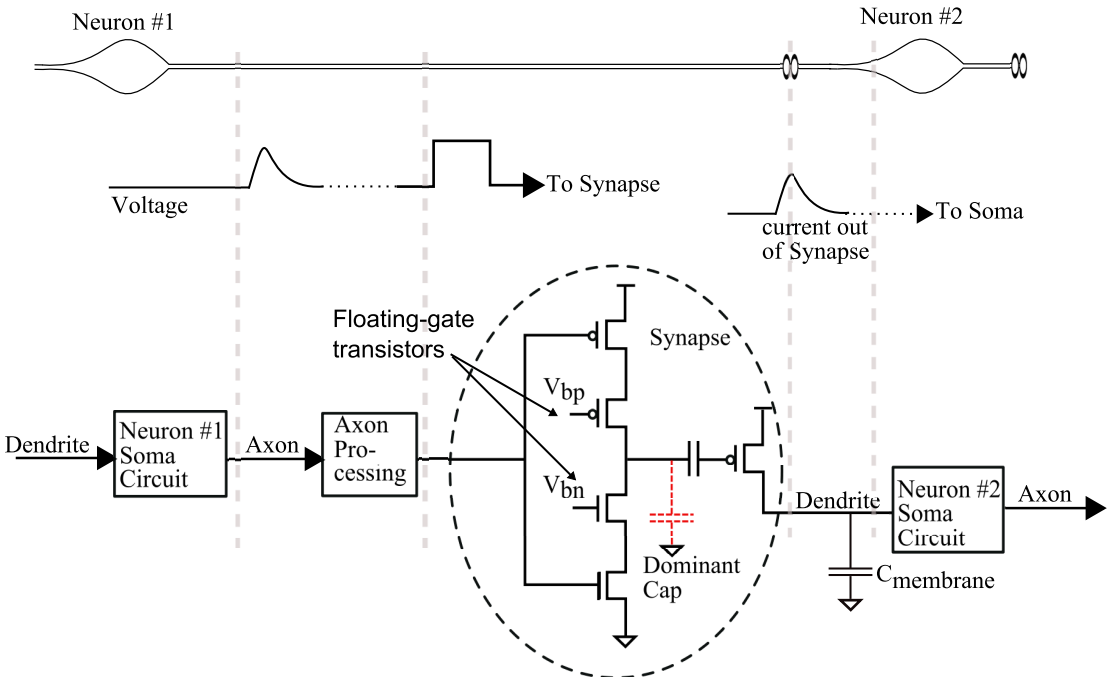


Figure 58. This figure shows the dominant capacitance which limits the velocity (and efficiency) of the neuron propagation. The capacitance is on the output of the current starved inverter. The velocity of the neuron firing is related to the time it takes to charge and discharge this capacitance. Therefore, the velocity of the system is proportional to the power and efficiency of the system. V_{bp} and V_{bn} are floating-gate transistors used to set the gate waveform driving the synapse transistor. Details of how these are changed (programmed) can be found in previous papers [11, 10].

$$= O\left(d^2\right) \tag{17}$$

Equation (17) is the upper bound for the number of nodes generated for a neuron system with a long refractory period. For the other extreme case when there is little or no refractory period for the neurons then one may make a very conservative estimate by assuming that all of the neurons enclosed by the wavefront fire each time the wavefront expands. This upper bound may be estimated as follows. At the first wavefront expansion one may apply (16) and record the number of nodes (18):

$$2\left(1^2 + 1\right) \tag{18}$$

When the second wavefront propagates, one may add all these firing neurons *and* assume that all of the first wavefront's neurons fired again so the new total number of nodes generated is (19).

$$2\left(1^2 + 1\right) + 2\left(2^2 + 2\right) \tag{19}$$

The number of nodes generated can be written for an arbitrary solution depth d as follows:

$$\begin{aligned} &SC_{short\_refractory} = \\ &\quad 2\left[\left(1^2 + 1\right) + \left(2^2 + 2\right) + \cdots + \left(d^2 + d\right)\right] \end{aligned} \tag{20}$$

Rearranging terms:

$$2\left(1^2 + 2^2 + \cdots + d^2\right) + 2\left(1 + 2 + \cdots + d\right) \tag{21}$$

Using Faulhaber's formula to further simplify (21):

$$= 2\left(\frac{d^3 + 3d^2 + d}{6}\right) + 2\left(\frac{d^2 + d}{2}\right) \tag{22}$$

$$= \frac{1}{3}d^3 + 2d^2 + \frac{4}{3}d \tag{23}$$

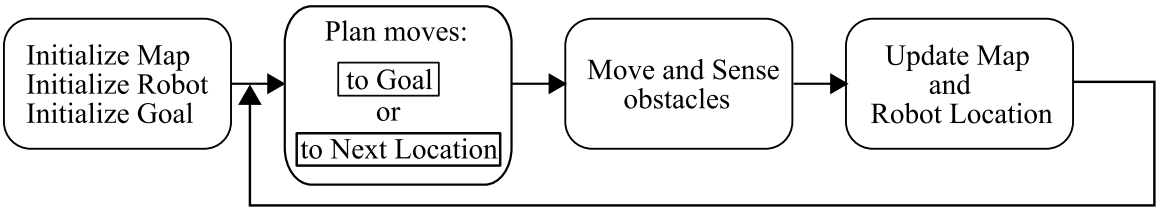


Figure 59. This shows the system view of path planning in an unknown environment with this neuron IC.

$$= O\left(d^3\right) \tag{24}$$

One may therefore claim that (24) is the upper bound for the number of nodes generated for a system with neurons with a short refractory period.

If one assumes that there is a long refractory period (i.e. a refractory period >> the solution time), and also assumes that each grid has a branching factor of 4 (i.e. the autonomous agent can, in the absence of an obstacle, move up, down, left, or right), one may use (17) as an estimate for the Space Complexity. Likewise, if still using Method 1, but assuming a short refractory time, one may express the Space Complexity as (24).

There is a maximum number of nodes required in memory in order to find the solution. The memory system could hold two items for each node: 1) a time stamp for when that node first fired and 2) a list of accessible neighbors. In an *MxN* grid map with no obstacles, this would require approximately *MxN* memory locations for the timing information and $4 \cdot (MxN)$ memory locations for the neighbor information.

In Method 2, if one assumes a branching factor of 4, then the memory requirement is $O(4)$. That is, when planning the next optimal move to the goal, the Neuron IC algorithm only needs to store data for when the four neighbor grid cells are reached. This number is the same for a long or short refractory period.

4.2.3 Completeness

A *complete* planner will either always find a solution or inform the user that there is no solution. This neuron IC planner is complete if the goal is at some finite depth, *d*. If there

Table 6. Comparing Neuron IC planner to digital methods, where d is the depth of the solution in the search tree.

Criterion	Neuron IC	wavefront [142, 154, 140, 155]
Time Complexity	$\frac{d}{v}$	$O\left(d^2\right)$
Space Complexity (# of nodes stored in memory)	Method 1/w long refractory: $O\left(d^2\right)$ $\leq O\left(MxN\right)$ Method 1/w short refractory: $O\left(d^3\right)$ Method 2: O(4)	$O\left(d^2\right)$
Resolution Complete? (assuming finite depth d)	Yes	Yes
Resolution Optimal? (assuming equal path length)	Yes	Yes

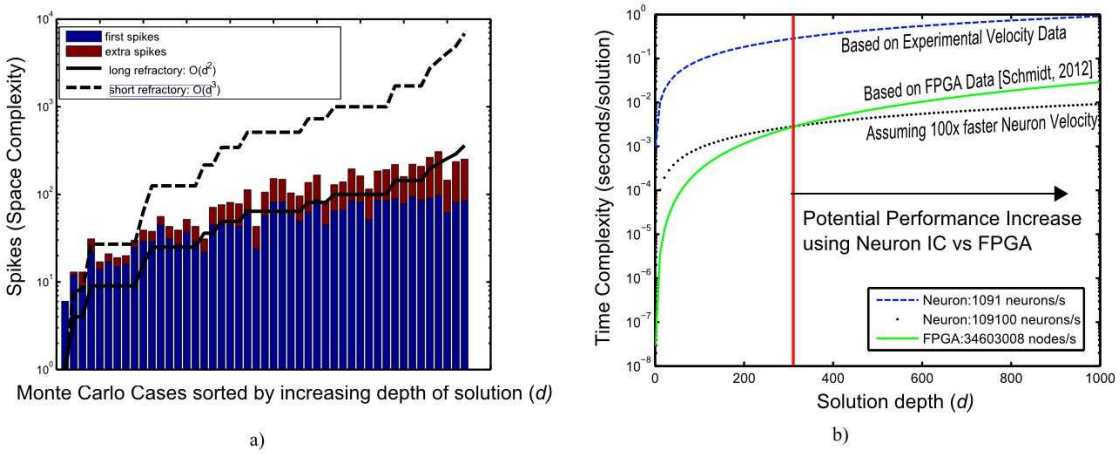
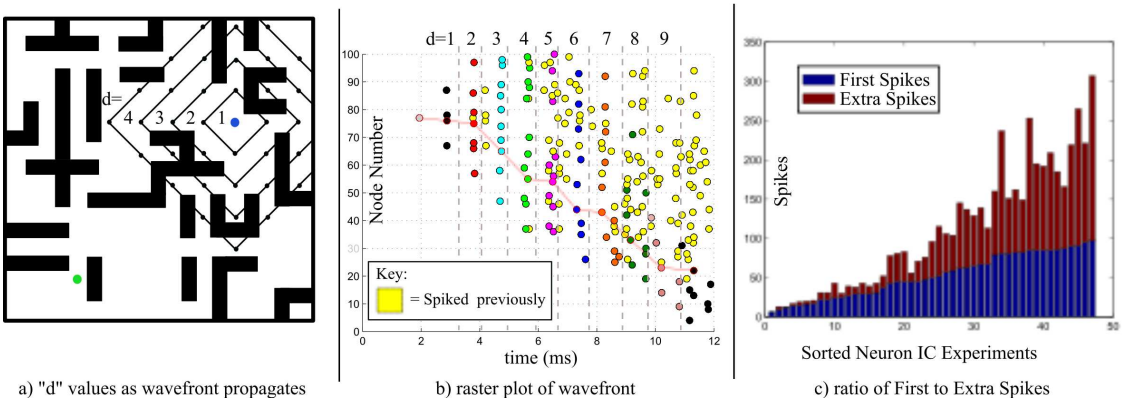


Figure 60. Space and Time Complexity for the Neuron IC. a) Space Complexity: This shows experimental Neuron IC data and how it compares to the Space Complexity models in Table 6. Based on this curve, one may assume that the refractory period in the experimental data is closer to being a long time than short. b) Time Complexity: This curve compares our Neuron IC performance to a state of the art FPGA implementation of Aker's wavefront algorithm [12]. The top line represents the Time Complexity for the Neuron IC with a signal propagation time of 1091 neurons/second. If this is increased by a factor of 100, then it is estimated that the Time Complexity of the Neuron IC will outperform the FPGA implementation when the solution depth is greater than approximately 315.



a) "d" values as wavefront propagates b) raster plot of wavefront c) ratio of First to Extra Spikes

Figure 61. Wavefront propagation in the Neuron IC and the Space Complexity. a) This depicts the wavefront emanating from the goal. Each wave is represented at a depth *d* in the raster plot in b. This represents a numerical potential field with a local minimum at the goal [13]. b) Raster plot showing all the events captured by AER (up until the solution) for the maze case in a.c) This chart shows the number of spikes for our set of experiments. This directly correlates to the time and Space Complexity for our Neuron IC. Since the refractory period of the neurons is less than the solution time, neurons have a chance to fire more than once. This shows that for the programmed refractory period in these 47 experimental cases, the number of *extra* spikes was approximately twice that of the number of initial spikes. (Note: 55 Monte Carlo cases were randomly generated. Based on the start, goal, and obstacle conditions, only 47 of these have a possible solution.)

is a solution, then one can calculate the maximum time, t_{max}, for finding it. If no solution is found by this time, then the planner can return a *No Solution* flag. Assuming a 1ms/neuron velocity and a 10x10 grid space of 100 nodes, the system would time out at 100 ms (max path length) as a bound on the case of no solution.

4.2.4 Optimality

A detailed optimality proof developed by Dr. Stephen Brink, can be found in [156].

4.2.5 Neuron IC Implementation vs. Digital

The Time Complexity of the Neuron IC planner can be compared to digital implementations of the wavefront algorithm. For digital implementations, the worst case Time Complexity is calculated as a function of the number of cached points or grid cells [154]. For a solution of depth *d* the number of cached points can be estimated using the derivation for (17). Similarly, the Space Complexity can be calculated as a function of the number of cached points or grid cells [154]. For a solution of depth *d* the number of cached points for the Space Complexity can be estimated using the derivation for (17) also. A side by side

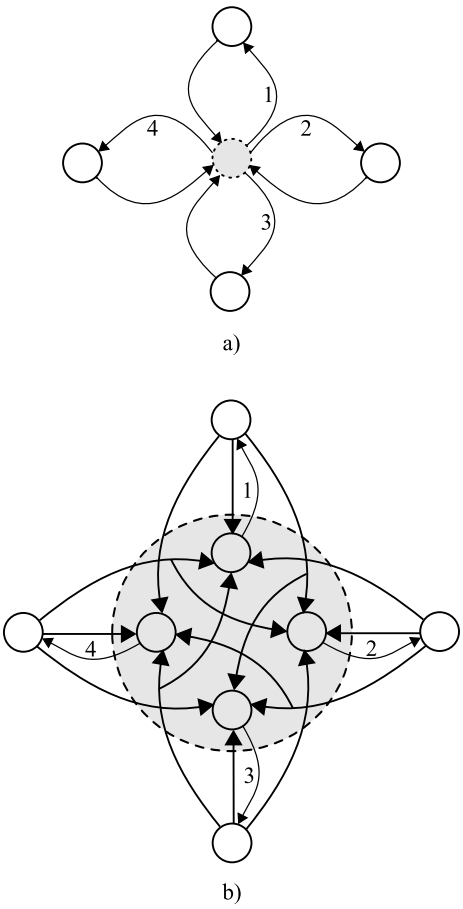


Figure 62. This figure shows how the actual implementation of an asymmetrical weighted node is implemented using multiple neurons in the Neuron IC. a) This is the desired weighting configuration for the node. The gray center node has different cost weights associated with propagating a wave to each of its four neighbors. b) The gray node in the center of (a) is implemented in the Neuron IC using four neurons. Each incoming edge must excite all four of the neurons which compose the center node.

comparison of the Neuron implementation to a digital implementation is shown in Table 6. There are potential tradeoffs for the two implementations. The Time Complexity of the Neuron planner may be a win if the signal propagation velocity is small. However the Space Complexity may not be as good with the Neuron planner if the neurons are programmed with a short refractory time. A state-of-the-art FPGA implementation of a wavefront based planner is found in [12]. Based on their performance numbers, one may calculate that a 1024x1024 grid is processed in 1/33 seconds. This allows one to estimate that the FPGA can process 34603008 nodes/second. Using this processing speed and d^2 as the estimated number of cached node cells for a given solution depth, one may estimate the Time Complexity of the FPGA implementation as a function of solution depth. Figure 60 compares the Time Complexity of our Neuron IC using the experimental velocity information to the estimated Time Complexity of the FPGA implementation. It can be seen that if the neuron velocity is increased by a factor of 100, then the Neuron IC should exhibit a performance improvement over the FPGA implementation.

4.2.6 Experiments with Non-uniform and Asymmetrical Edge Weights

The neuron structure allows one to develop sophisticated graphs with varied edge weights between nodes of the grid. Two specific cases are presented. First, asymmetric edge costs are assigned to describe cases which have a certain cost to travel a path in one direction, but a different cost to travel the same path but in the opposite direction. The application of this feature can translate to real world problems involving hills, traffic patterns, etc. Second, cases are presented where the nodes near an obstacle are given higher costs to visit these nodes. This is in an effort to keep the autonomous agent at a safe distance from obstacles. This grid weighting can also be used to differentiate among terrains such as sand, ice, gravel, or smooth pavement. These two features of the Neuron wavefront planner make it special because the normal wavefront approach cannot handle varying terrain types or uncertainty in the world's state [151]. Figure 62 shows how the hardware implementation of an asymmetrical weighted node is implemented using multiple neurons in the Neuron

IC.

Figure 63 shows a situation where asymmetric edge costs are assigned to describe cases which have a certain cost to travel a path in one direction, but a different cost to travel the same path but in the opposite direction. Consider the Golden Gate Bridge in San Francisco. Tolls are collected only in the southbound direction (the path *into* San Francisco). In this simple, but illustrative example, the asymmetrical traveling paths are compared to paths between two regions which have toll booths. A toll is collected only in one direction in each path. Figure 63a shows the two paths between nodes 1 and 100. There are two paths that lead to and from Nodes 1 and 100. The architecture is designed so that it costs more to travel from Node 1 to 100 using Path B instead of Path A. Similarly, it costs more to travel from Node 100 to 1 using Path A instead of Path B. The node points with the stars in the grid have two neurons each to create these nodes. This is highlighted by the dashed circles surrounding the neuron representations. The grid implementing the cartoon in the Neuron IC is shown in Figure 63b. As described in Figure 62, the nodes representing the toll booth intersections are implemented with multiple neurons. In Path A, the toll booth intersection nodes are formed using Neurons 28 and 37, and in Path B, the toll booth intersection nodes are formed using Neurons 64 and 73. In Path A, the directional sequence of nodes [...38, 28, 27...] is a toll free wavefront sequence because each of these neurons has edge weights of 1 between them. A raster plot of the Neuron IC wavefront experimental data confirms this in the shaded section of Figure 64a since the propagation time between Nodes 38 and 28 as well as the propagation time between Nodes 28 and 27 are both approximately 1ms. In Path A, the directional sequence of nodes [...27, 37, 38,...] incurs a toll penalty. A raster plot of the wavefront experimental data also confirms this in the shaded section of Figure 65a since the propagation time between Nodes 37 and 38 is slightly over 4ms (there is some non-ideality here because the toll booth neurons were programmed for a nominal delay of 5ms).

Figure 66 shows experimental hardware results demonstrating additional advantages of

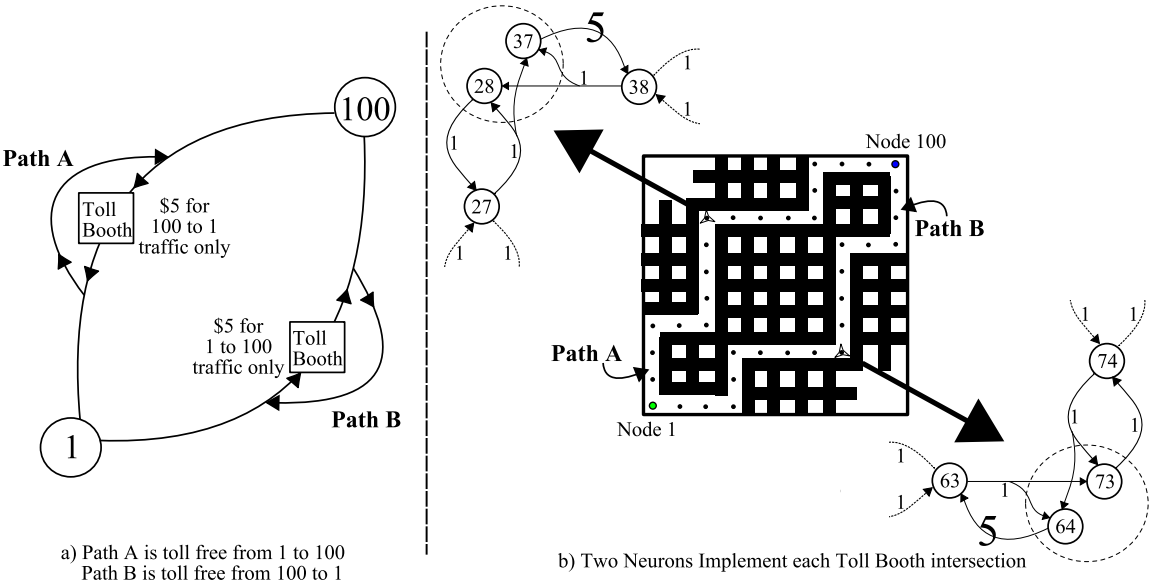


Figure 63. This figure highlights how neurons can be paired to create an asymmetric edge weight architecture. a) This cartoon illustrates the "cost" associated with the directionality of two paths. b) This maze represents (a) and is implemented on the Neuron IC.

non-uniform edge weights in the planner. Wavefront methods generate optimal paths with regard to length, but they tend to "hug the sides of obstacles [151, 141]." One solution to this problem is by "growing" the size of the obstacle [151]. Instead of growing the obstacles, the paths near an obstacle are penalized in the Neuron IC. This is implemented by programming the Neuron IC such that the nodes nearest the obstacles are assigned a higher cost to visit. This extra cost decreases for nodes further away from obstacles. These results demonstrate that weighting the edges between neurons differently can push the autonomous agent away from obstacles and generate a path solution taking into consideration not only path length, but also proximity to obstacles. The edge weights are set in the Neuron IC by adjusting the gate waveforms which drive the synapse transistors (Figure 58). A higher weight cost correlates to a slower response of the synapse and therefore a slower propagation delay time.

Two different weight experiments are shown in Figs 66a and c. In these experiments a ring of nodes around each of the obstacles is assigned a larger weight. The rings are denoted by shaded regions in Figs 66a and c. The non-shaded regions have neurons that are

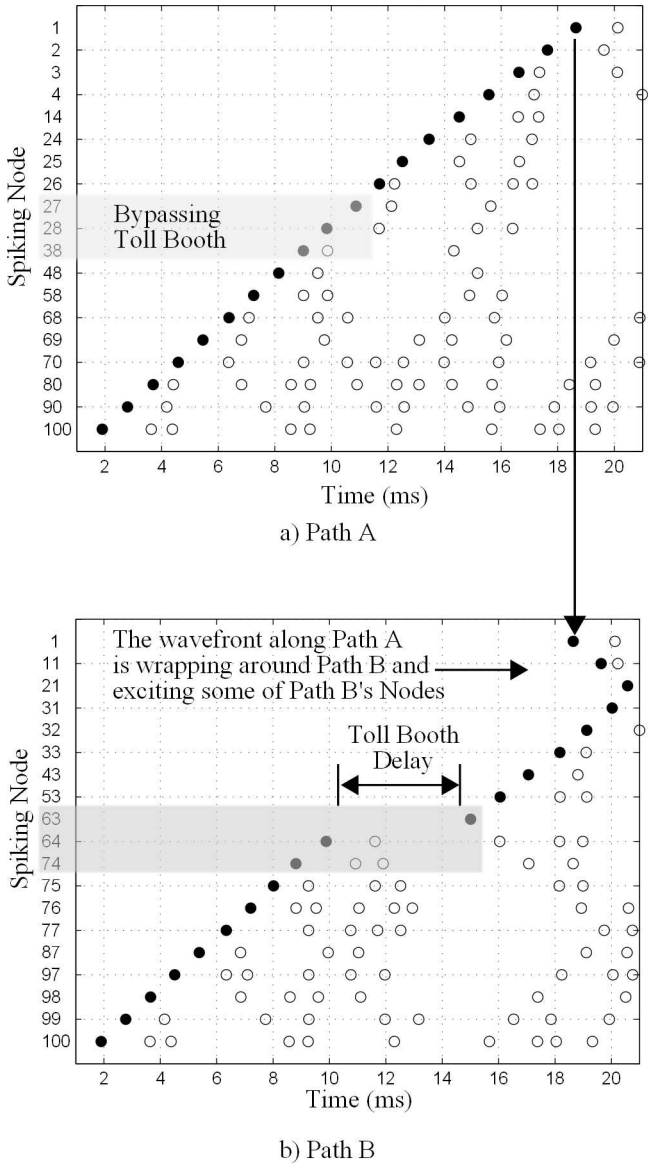


Figure 64. This figure is measured Neuron IC data for the experimental setup in Figure 63. For this experiment the autonomous agent is planning a path from Nodes 1 to 100. Path A was chosen by the Neuron IC. a) Raster plot showing the sequence of nodes for Path A (in black). There is approximately a 1ms delay between neuron firings. This shows that the toll booth was not crossed. b) Raster plot showing the sequence of nodes for Path B (in black). Notice the approximate 4ms delay between Nodes 63 and 53. This shows that the toll booth *was* crossed.

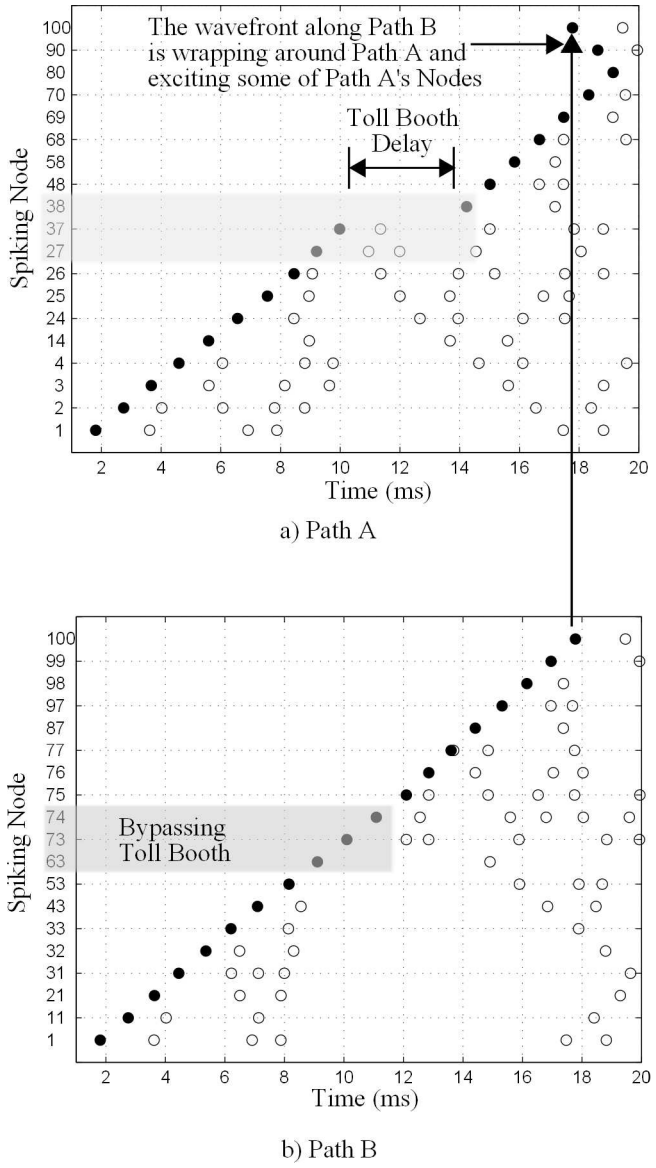


Figure 65. This figure is measured Neuron IC data for the experimental setup in Figure 63. For this experiment the autonomous agent is planning a path from Nodes 100 to 1. Path B was chosen by the Neuron IC. a) Raster plot showing the sequence of nodes for Path A (in black). Notice the approximate 4ms delay between Nodes 37 and 38. This shows that the toll booth *was* crossed. b) Raster plot showing the sequence of nodes for Path B (in black). There is approximately a 1ms delay between neuron firings. This shows that the toll booth was not crossed.

programmed to have a 1ms nominal propagation time between neurons. The shaded areas have neurons that are programmed to have a 2 or 2.5ms nominal delay between neurons for the experiments in Figs 66a and c respectively. At the boundary between the shaded and non-shaded regions, if a wave is propagating *into* the shaded area it will incur the larger cost of the shaded region. However, if a wave is propagating *out of* the shaded area it will incur the lower time cost of the non-shaded area. The cost is thus dictated by the region *into* which the wavefront is propagating. Figure 66a and b give results where the shaded region is programmed for a 2ms delay. The nominal time cost for a wave to propagate through the shaded area between the start to the goal is 12ms. This is in contrast to the nominal time of 13ms for the wavefront to propagate around the obstacle. The Neuron IC planner chose the path shown by the arrows in Figure 66a. The data points in Figure 66b show the measured delays incurred from propagating the wavefront on each of the chosen path's edges. The measurements show some non-ideality in the system. The first five data points should ideally each be 2ms delays, and the last two data points should ideally each be 1ms delays. Figs 66c and d give results where the shaded region is programmed for a 2.5ms delay. The nominal time cost for a wave to propagate through the shaded area between the start and the goal is 14.5ms. This is in contrast to the nominal time of 13ms for the wavefront to propagate around the obstacle. The Neuron IC planner chose the longer but least time cost path shown by the arrows in Figure 66c. The data points in Figure 66d show the measured delays incurred from propagating the wavefront on each of the path's edges. Again, there is some non-ideality in the system as shown in the measurements as all of the data points should ideally each be 1ms delays. These two experiments demonstrate that the neurons can be programmed to weights which create paths from a start to a goal that take into account both proximity to obstacles and path length.

4.2.7 Scalability

"To construct a navigation function that may be useful in mobile robotics, a high-resolution (e.g., 50 to 100 points per axis) grid is usually required [142]." Concerning grid scalability,

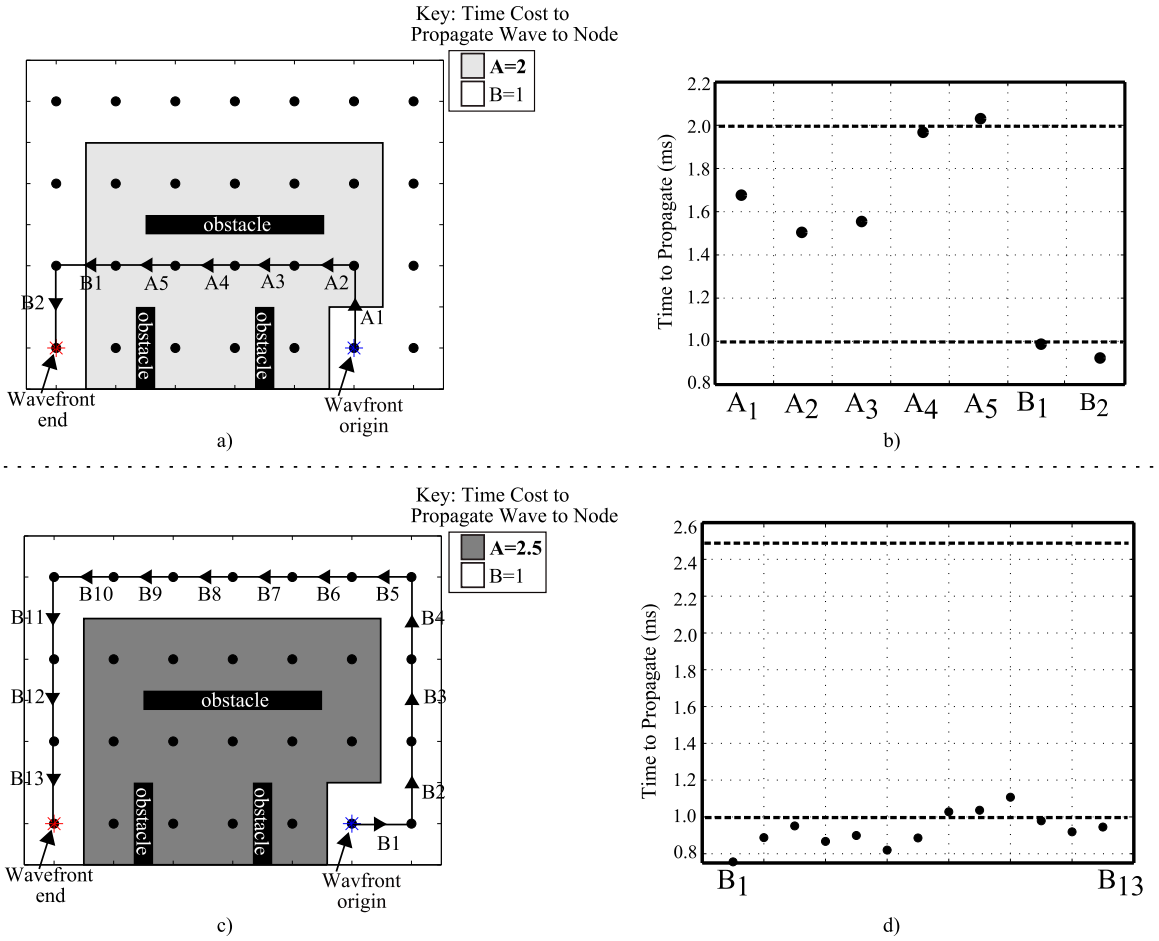


Figure 66. These experimental Neuron IC results demonstrate that weighting the edges between neurons differently can push the autonomous agent away from obstacles and generate a path solution taking into consideration not only path length, but also proximity to obstacles. a) A ring of nodes around each obstacle is given a weight of 2 to enter each node. The best cost path is chosen to be the shortest in length, but also the one that comes closest to the obstacles for the longest amount of time. b) Propagation time between nodes for the edges selected for the best path in the experiment in (a). c) A ring of nodes around each obstacle is given a weight of 2.5 to enter each node. The best cost path in this experiment, in contrast to (a), is not the shortest path. This path, however, avoids getting close to the obstacles. d) Propagation time between nodes for the edges selected for the best path in the experiment in (c).

this neuron planner should be able to be expanded to this scaling or a much larger grid. This is enabled largely because the repeater nature of the neurons enables the propagated signal to continue without reducing. This is in contrast to resistive based path planning methods where the signal at the nodes decreases as the grid gets larger [48]. The synaptic array in the IC is approximately 3 mm^2 in area; therefore one can imagine in a single reticle size chip in the same process to enable an array of one million synapses and thousands of neurons on a single IC. These numbers could increase substantially as one moves from the 350nm process to a more modern IC processes [10]. These numbers of neuron elements would allow one to achieve the 50 to 100 points per axis resolution required for fielding a planning system. Figure 67 shows how multiple neuron ICs can be connected together in order to expand the grids size. In the current form, one could connect two neuron ICs together and use the AER system to pass signals between the two. Also, it is possible to design a new Neuron IC which would have more neurons for larger grids.

4.2.8 Power Costs

The power used by this system can be split into four categories: Neuron pre-programming, Environment map programming, wavefront propagation and neuron read-out, and analysis. We claim that the power used by the FPAA to arrive at the solution is much less than that of a digital solution. We cannot make appropriate claims however of the pre-programming, environment map programming and analysis power because we have not optimized our embedded system for these problems. The embedded system which interfaces with Matlab and the Neuron IC uses current on the order of 400mA.

4.3 Summary

In a comprehensive survey of autonomous rotorcraft unmanned aircraft systems, Kendoul describes path planning methods and algorithms which represent the “most used and practical methods, with a particular focus on works with experimental results [139].” These

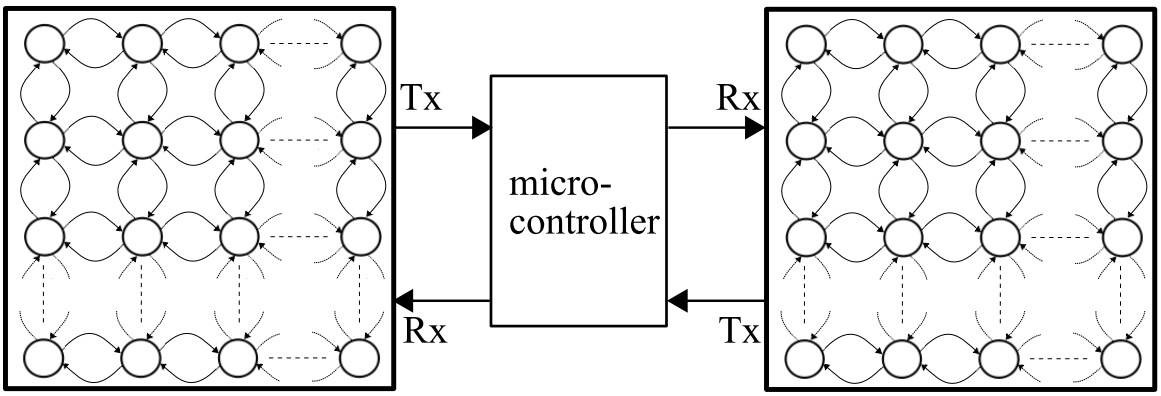


Figure 67. Two neuron ICs can be connected via the Tx/Rx pins in order to expand the grid size.

methods include Road Map and Potential Field approaches. Dijkstra's algorithm is commonly used to plan a path in Road Map methods [139]. A specialized form of Dijkstra's algorithm is the wavefront method [142] , so it is reasonable to say we are using a state-of-the-art method.

This chapter continues to build upon the idea that analog circuits, and specifically biologically inspired neuron circuits, can be used for path planning and are adding to the existing research which combines analog VLSI and robotics [134]. This chapter has presented approximately 50 Monte Carlo path planning results for 10x10 grids. Although not as large as the 50 to 100 points per axis grid as is usually required [142], this modest grid demonstrates experimentally that a Neuron IC can be used to propagate a wavefront for planning. This provides an advantage because the wavefront is propagated in parallel. This chapter also characterized the Space Complexity (memory requirement) of this Neuron IC method and validated the model with experimental data. This work is new for a couple of reasons: First, it provides an extensive amount of measured data representing different map scenarios from a fabricated AVLSI neuron IC. Second, our analog circuit implementation is unique because it is implemented on a floating-gate based reconfigurable neuron analog IC. Finally, this work has started to quantify the performance gain for using an analog solution instead of a digital one in terms of Time Complexity.

CHAPTER 5

CONTROL WITH FPAAS

5.1 The Mobile Manipulator and FPAA

A mobile manipulation robotic platform solving the Towers of Hanoi puzzle is described in this chapter. This work is based on a collaboration with David Lenz, Sebastian Hilsenbeck and Smriti Chopra, and they should be credited with the digital software design and implementation which made this chapter possible [8]. This work is described with the possibility of leveraging this work in two domains. First, the mobile manipulator could be developed into a system that could interact with children or adults in a *turn taking* scenario. Second, this platform can be used to investigate feedback control systems implemented with reconfigurable analog electronics. Robot control software called *Player* is used as the main software for this system, [127, 157]. *Player*, running on a laptop, is the brains of the system. It receives sensor input from an overhead camera for localization and then commands the robot as desired. *Player* is operated in two robot environment modes. The first mode is *Player* interacting with a real robot in the real world. The second mode is *Player* interacting with a simulated robot in a 3D simulated environment with dynamics. This 3D environment is called *Gazebo*.

The *Player* software has the ability to interact with the FPAA reconfigurable analog electronics system, Figure 9. In this configuration, the FPAA could be characterized as a *Feedback Control Co-processor* for the robot's navigation system. Path planning is another demonstrated use of an FPAA in robotics [7].

The mobile manipulator system, using a Pioneer robot and arm [125], is demonstrated solving the classic Tower of Hanoi problem, Figure 68. In this puzzle, a tower of disks is created by stacking disks on top of each other. One of the rules is that only smaller disks may be placed on larger disks. This version assumes there are three possible locations for the tower's location. The tower starts in one of these locations. The goal is to move the

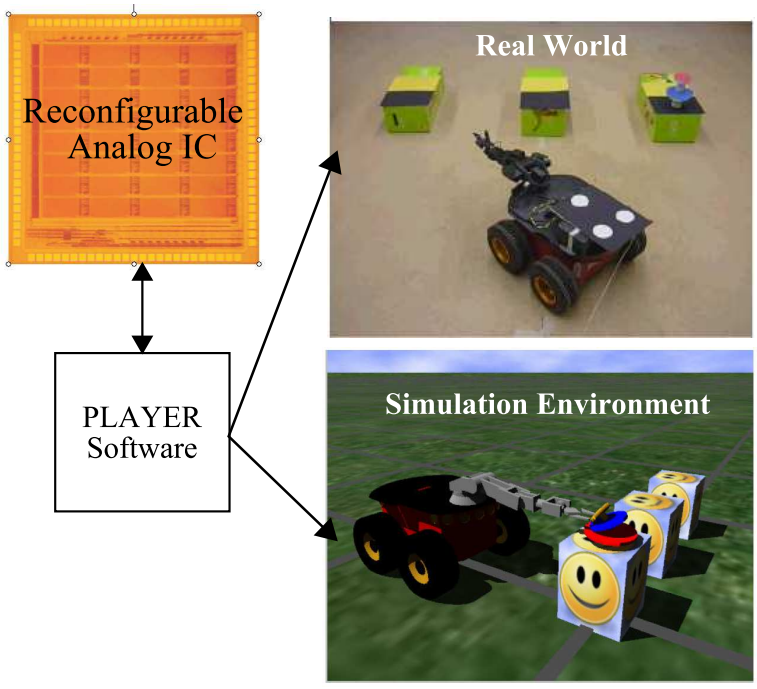


Figure 68. This is the big picture of the system: a client software called *Player* interacts with either the real world or a simulated world and solves the classic Towers of Hanoi puzzle. Additionally, the software has the ability to interact with a reconfigurable analog co-processor [8].

tower from one location to another location.

This robotic manipulator has three main tasks: Sensing, Thinking, and Acting. The *Sensing* task involves an overhead camera as the primary sensor. Image processing tasks for the Towers of Hanoi problem include segmenting the disks from the background and identifying their size and position. The *Thinking* tasks include creating a sequence of legal actions for moving the disks so that the goal is achieved (path planning), as well as turning these high level commands into low level control functions. The *Acting* tasks include commanding the Pioneer robot's forward/reverse velocity and rotation, as well as commanding an attached five degree of freedom (DOF) Pioneer manipulator arm to move the pieces.

Section 5.2 discusses related work, Section 5.3 describes architecture for Sensing, Thinking, and Acting, Section 5.4 compares differences between simulation and real world operation, and Section 5.5 is a closing summary.

5.2 Related Work

5.2.1 Playing with Robots

Robots have been used in the past for games such as chess [158], or as a therapy aid [159]. Robots have also been used to help children with disabilities [160, 161].This mobile manipulator could be extended for use in future work such as *turn taking*, [162]. A simple non-mobile manipulator is described for solving the Towers of Hanoi problem in [163]. This was part of a Robotics Education Lab at CMU. Humans used a mobile web interface to instruct a PR2 how to solve the towers of Hanoi problem in [164]. A video of a PR2 and many other robots solving the Towers of Hanoi may be found on the internet.

5.2.2 Analog Control

A number of recent papers have been written regarding using reconfigurable analog circuits called Field Programmable Analog Arrays (FPAA) for low level control. This work and [165] are based around custom FPAAs, but many are based on the switch-capacitor Anadigm IC design [166, 167, 168]. General references concerning PID controllers are [169, 170, 171, 172]. Background for using Operational Transconductance Amplifiers (OTAs) for PID control is found in [173, 174]. Finally, although this robotic system is accessible and easy upgraded and serviced, this is not always the case for all robotic platforms. Other FPAAs are being explored to allow flexibility in sensing and control circuits of space systems [175, 176]. The FPAA in this chapter is typically different than other reconfigurable analog circuits because it uses floating-gate transistors as the switch matrix.

5.3 Architecture for Sensing, Thinking, and Acting

One of the goals of the architecture is to give the robot a high level of autonomy. The robot's a priori knowledge consists of the following:

- A list of potential disk colors
- An initial estimate of pole positions

- The height of the disks

The system block diagram in Figure 74 provides a high level view of the robot's Navigation system and also shows how it interfaces with the planner, vision sensor, and robot hardware. The Sensing, Thinking, and Acting portions of this block diagram are individually addressed in the remainder of this section.

5.3.1 Sensing

Vision is the primary sensor in this system. It sends information to the Tracker sub-block. It assumes that there is an overhead camera available to image the robot, poles, and disks at all times. "OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision [177]." It has been integrated into the control program for image processing tasks. Figure 79a shows an example image from the overhead camera modeled in Gazebo, and Figure 79b shows a view from the real overhead camera. Working with the Tracker, this image system is able to successfully segment images using color features and is able to extract colored circles from images.

5.3.2 Thinking

The section of the robot's system block diagram that describes *thinking*, Figure 74, consists of four main tasks: Navigation, Planning, Tracking, and maintaining the internal World Model. A high level state machine description is found in Figure 69. The first state in Figure 69 is "Get Initial Configuration." In this step, the system determines the number and color of the disks and the initial positions. The a priori information that helps this process is that it is assumed that the disk colors come from a known set of colors in a color list.

The Planner's task is to identify a sequence of actions that will accomplish the goal of moving the disks from their starting position to the goal position. A previously existing Towers of Hanoi planner was integrated into this system, [178]. A plan has the following form:

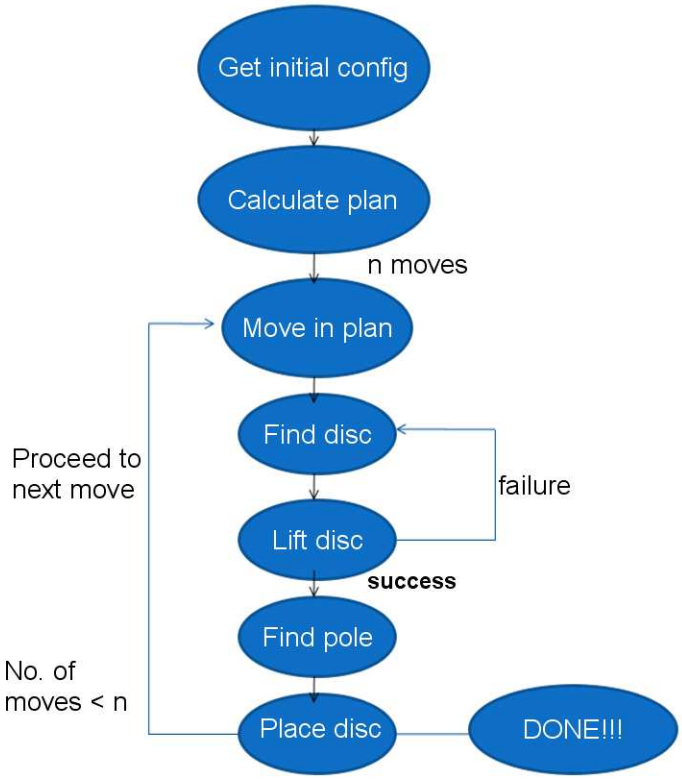


Figure 69. This figure shows a high level flowchart of the Thinking tasks [8].

1. Take the disk on pole 1 and place it on pole 3
2. Take the disk on pole 1 and place it on pole 2
3. Take the disk on pole 3 and place it on pole 2
4. ...

The Navigation block's task is to convert high level plans to low level commands. Proportional-Derivative closed loop control systems are used to control the robot's angle and forward/reverse position. A block diagram of a Proportional-Integral-Derivative closed loop control system is found in Figure 70. The system was operated using the Digital Controller, but this figure also shows a diagram of how the FPAA based analog PID controller could be integrated into the loop. Ideally, the PID output signal would be sent directly to the plant and not use the A/D and D/A functions.

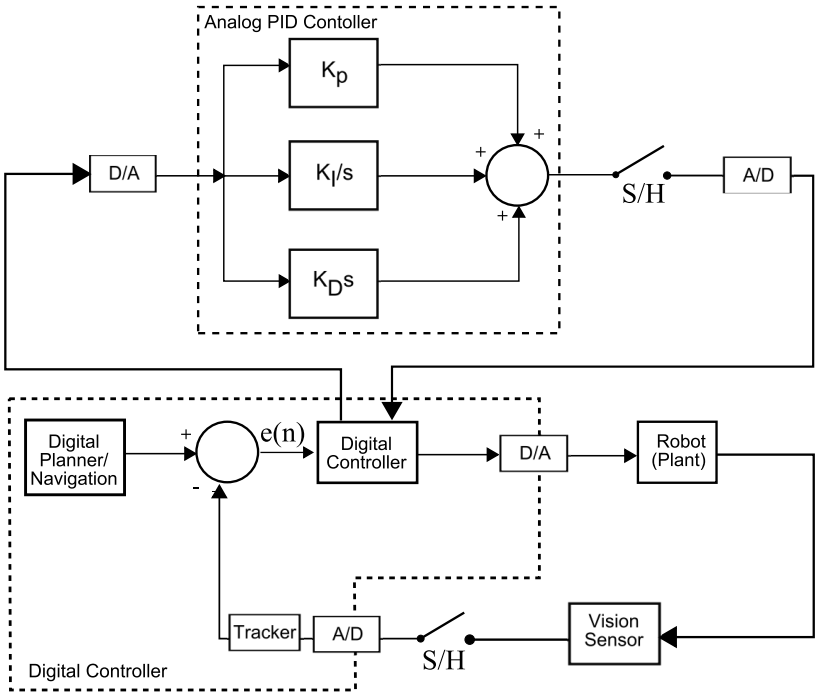


Figure 70. This figure illustrates an example of how an analog co-processor PID controller could be merged with the digital controller for initial testing. The control system implemented for this project uses a Digital Proportional-Derivative closed loop control system to control the robot's position and orientation [8].

The Tracker has three main image processing tasks: to determine the 1) Disk poses, 2) Robot poses, and 3) Pole poses. The tracker uses colors to identify objects. To track the disks, first they are segmented from the background with thresholding in the HSV color space. A "blobfinder" is then applied to the segmented image [126]. The blobs are then filtered based on size to determine if they are too large or too small. Finally, the blob's features such as position, area, and standard deviation are calculated, and this information is returned to the Navigation routine. This is illustrated in Figure 71. The same process is used to track the poles (boxes) on which the disks sit, except that before the blobfinder is applied the segmented image undergoes erosion and dilation to remove the eyes and mouth of the smiley on the boxes in the simulation. (This process was not used with the real hardware because uniform colored black boxes were used for the poles.) Finally, Robot pose is determined by using a triangle formed by three white dots added to the back of the Pioneer robot. These dots are segmented by the tracker and the robot's pose is calculated.

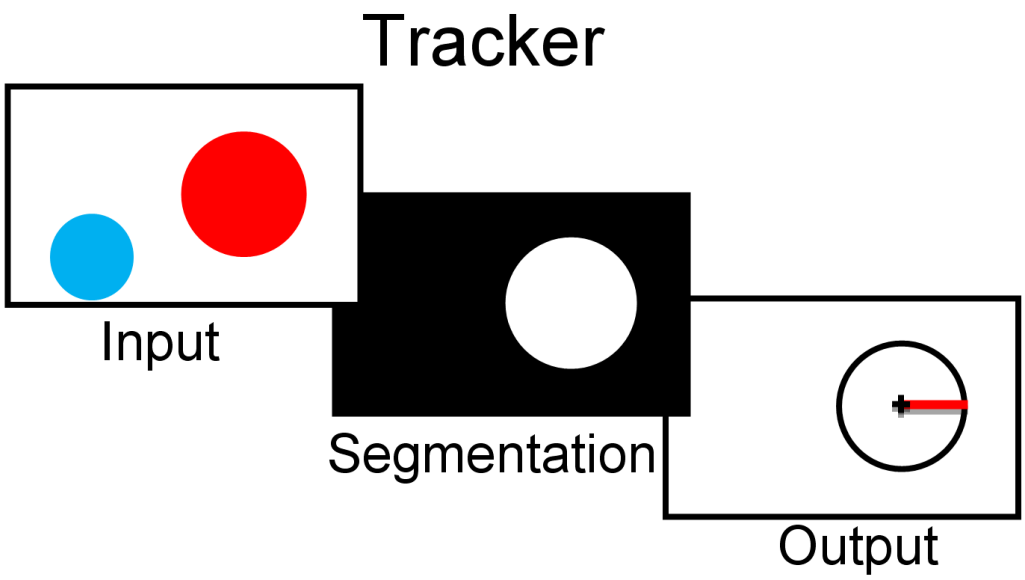


Figure 71. This figure illustrates that the Tracker first segments the image based on color (in this example it was asked to track the red disk). It then calculates the radius of the disks [8].

All calculations are in camera coordinates. An internal World Model is also maintained by the robot. This World Model contains three items:

- List of disks (with each disk's color, position, and radius)
- List of positions of the poles
- Color list

The overall strategy for executing a high level command is shown by the state machine in Figure 73. The robot uses the closed loop controller when rotating to the disk or goal and when moving to the disk or goal.

This software/hardware platform offers a unique capability to integrate our FPAA system into this robot for control. Figure 74 shows how the FPAA might be integrated into the system block diagram. The FPAA contains many OTAs. Figure 72a shows how OTAs can be used to implement a PID controller [14]. Figure 72a builds upon the OTA PID model in [14] by adding parasitic capacitances that are inherent when routing circuits on an FPAA. The current out of an OTA is a function of its transconductance gain, G_m, and the difference between the positive and negative terminals, (26) [122]. Ideally, the current into the

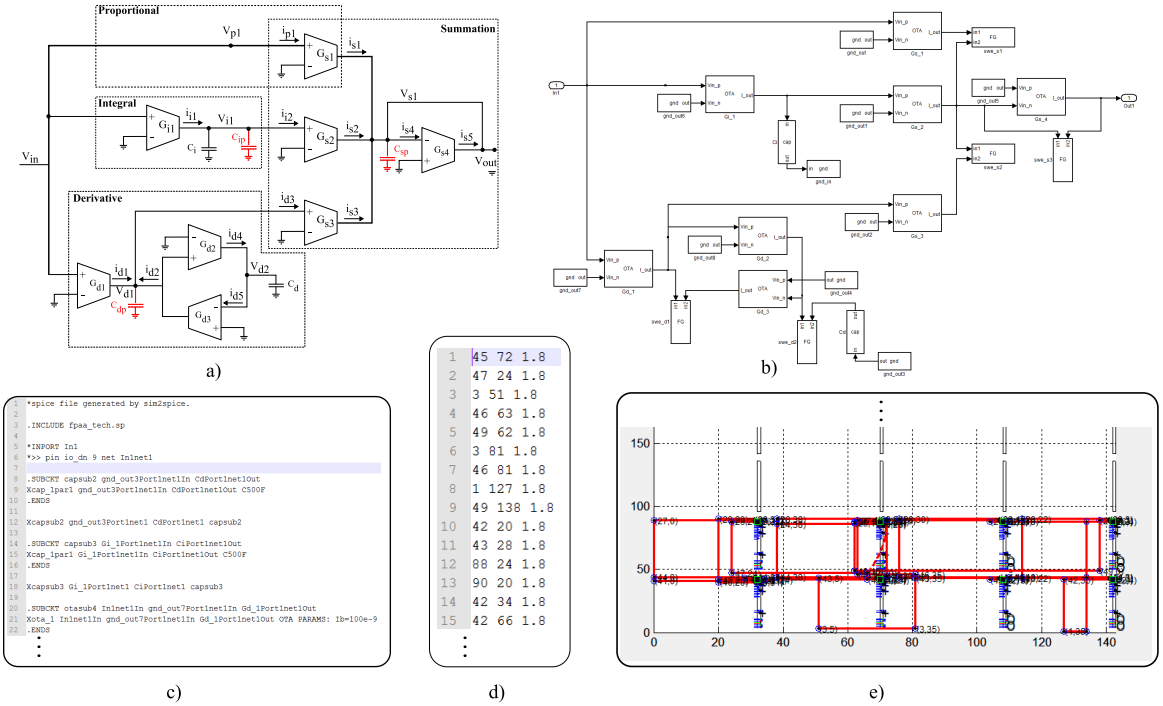


Figure 72. Design Flow for an OTA based PID controller. a) OTA based PID controller based on [14]. Unlike [28], this model includes parasitic capacitances that are a part of an actual implementation and affect performance. b) Simulink Block Diagram of controller. c) SPICE list generated by *Sim2Spice* tool. d) FPAA switch list generated by GRASPER tool. e) RAT Figure showing switch list routing on RASP 2.8a IC [8].

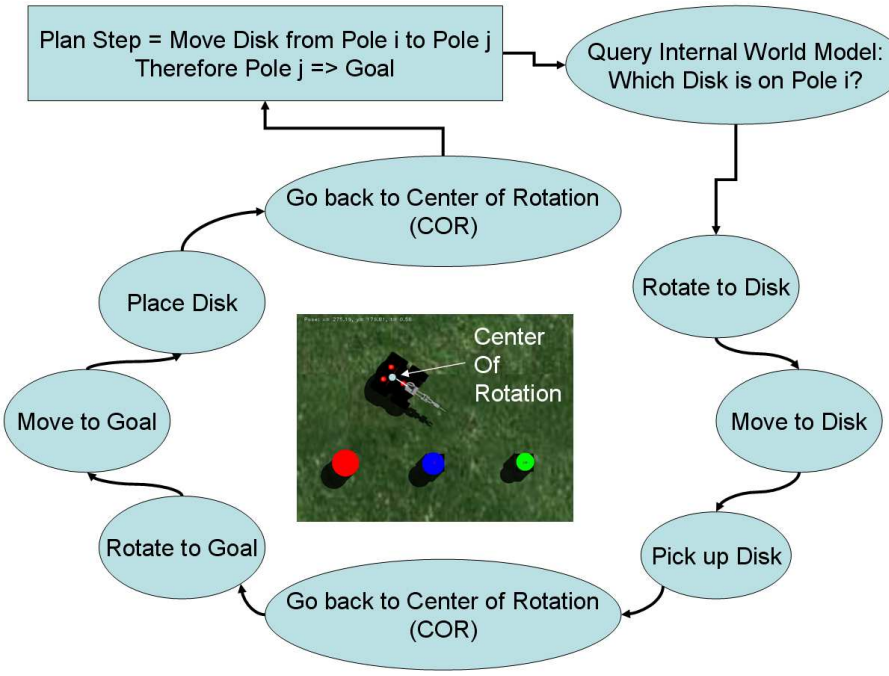


Figure 73. This figure illustrates the overall guidance and control strategy. The robot will perform this loop for each high level command in the planning sequence [8].

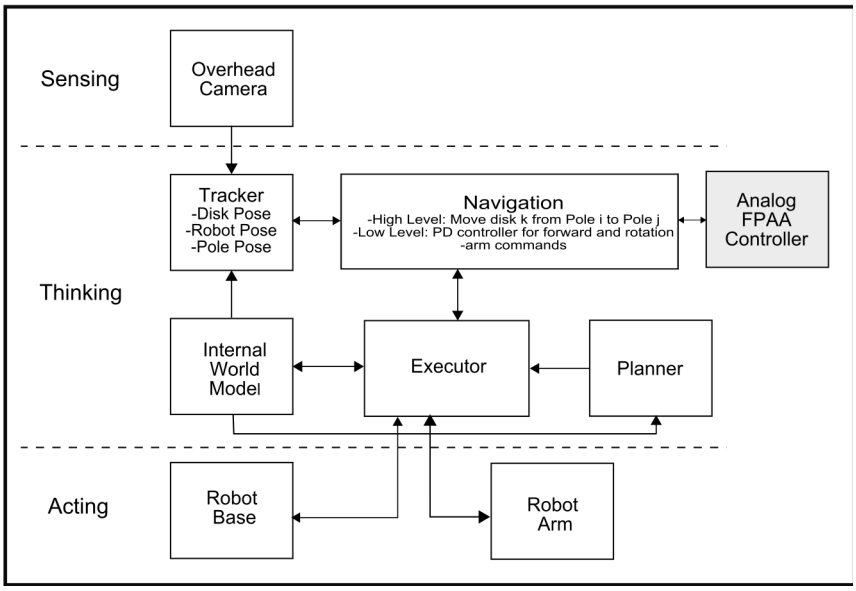


Figure 74. High level control System Block Diagram: this figure shows how the sensing, thinking, and acting systems are combined and where the analog co-processor fits into the larger robot system [8].

positive and negative terminals of an OTA is zero. In subthreshold operation, the output current of an OTA is shown in (25) [122].

$$I_{out} = I_{bias} \tanh\left(\frac{\kappa}{2U_t}\left(V_p - V_n\right)\right) \tag{25}$$

For small values, $\tanh(x) \approx x$, and G_m, the so called *transconductnace* of the amplifier, is the slope of the *tanh* curve at the origin.

$$I_{out} = G_m\left(V_p - V_n\right) \tag{26}$$

Where G_m is calculated to be:

$$G_m = \frac{\partial I_{out}}{\partial V_{in}} = I_{bias}\frac{\kappa}{2U_t} \tag{27}$$

Therefore, one may adjust an OTA's *transconductnace* by adjusting the bias current, I_{bias}. Using the notation from Figure 72a, the PID gains K_P, K_I, and K_D in Figure 70 for an OTA based controller are as follows. The intermediate *Proportional* voltage term is:

$$V_{p1}(s) = V_{in}(s) \tag{28}$$

The intermediate *Integral* voltage term, taking into account integral circuit parasitic capacitance, C_{ip}, is:

$$V_{i1}(s) = \frac{G_{i1}}{C_i + C_{ip}} \frac{1}{s} V_{in}(s) \tag{29}$$

The intermediate *Derivative* voltage term, taking into account derivative circuit parasitic capacitance, C_{dp}, is:

$$V_{d1}(s) = \frac{G_{d1}}{C_{dp}s + \frac{G_{d2}G_{d3}}{C_d s}} V_{in}(s) \tag{30}$$

The individual PID currents are added using four OTAs. Taking into account summation circuit parasitic capacitance, C_{sp}, the equation is:

$$G_{s1}V_{p1} + G_{s2}V_{i1} + G_{s3}V_{d1} + I_{out} = C_{sp}\frac{dV_{out}}{dt} \tag{31}$$

$$I_{out} = -G_{s4}V_{out} \tag{32}$$

Substituting (32) in to (31), taking the Laplace transform, and simplifying yields a transfer function for the analog PID controller with parasitic capacitances:

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{1}{\left(C_{sp}s + G_{s4}\right)} \left(\begin{array}{l} G_{s1} \\ + \frac{G_{s2}G_{i1}}{C_i + C_{ip}} \cdot \frac{1}{s} \\ + \frac{G_{s3}G_{d1}}{C_{dp}s + \frac{G_{d2}G_{d3}}{C_d s}} \end{array} \right) \tag{33}$$

Table 7 compares the PID gain terms with and without parasitic capacitances. The terms with parasitics reduce to the ideal under DC conditions. As described in (27) The PID gains can be tuned by adjusting the OTA bias currents. The bias currents are tuned on the FPAA by adjusting the charge on the floating-gate of an OTA's "tail-current" transistor.

Table 7. Analog Proportional Integral Derivative Controller design with and without parasitic capacitances

Gain Term	Ideal	Realistic with parasitic capacitance
Proportional (K_P)	$\frac{G_{s1}}{G_{s4}}$	$\frac{G_{s1}}{(C_{sp}s+G_{s4})}$
Integral (K_I)	$\frac{G_{s2}G_{i1}}{G_{s4}C_i}$	$\frac{G_{s2}G_{i1}}{(C_{sp}s+G_{s4})(C_i+C_{ip})}$
Derivative (K_D)	$\frac{G_{d1}G_{s3}C_d}{G_{d2}G_{d3}G_{s4}}$	$\frac{G_{s3}G_{d1}}{(C_{sp}s+G_{s4})(C_{dp}s^2+\frac{G_{d2}G_{d3}}{C_d})}$

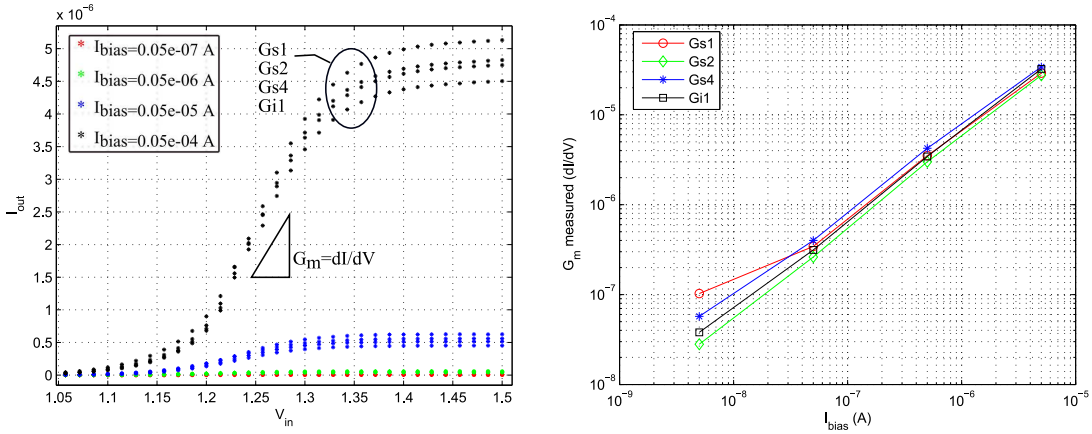


Figure 75. Characterizing selected OTAs in Figure 72a. a) Tanh curve as in (25). b) This figure illustrates that changing the I_{bias} changes the G_m. Note that there is not much variability among OTAs when I_{bias} increases.

Figure 75 shows the results of measuring and characterizing selected OTAs on the FPAA which affect the Proportional and Integral gains. The OTAs correspond to G_{s1}, G_{s2}, G_{s4} and G_{i1} in Figure 72a. Figure 75a shows that the measured data follows the *tanh* curve described in (25). Figure 75b illustrates the how changing the I_{bias} of the OTA changes the G_m. Note that in this experimental data there is not much variability among OTAs when I_{bias} increases.

Experimental data was gathered from the FPAA to characterize the parasitic capacitances affecting the Proportional and Integral gains (see C_{ip} and C_{sp} in Table 7). Bode plots of the Proportional and Integral gain terms are found in Figs 76 and 77 respectively. To generate these plots, V_{out} was measured when a 100mV (peak-to-peak) V_{in} was applied with varying frequency. The magnitude and phase responses of the experimental data is plotted with blue circles. The transfer function for the Proportional and Integral gains in Table 7

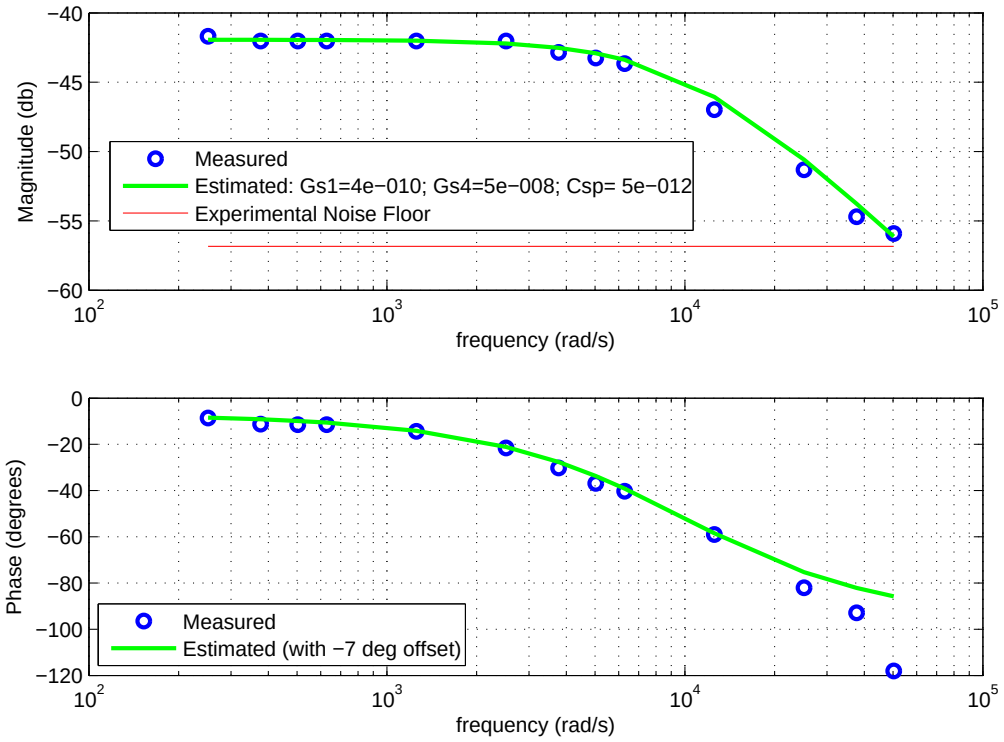


Figure 76. Bode Plot of the Analog Proportional Controller. The gains G_{s1}, G_{s4}, and the parasitic capacitance C_{sp} is estimated and used to plot a curve over the experimental data. An I_{bias} = 0.05e-07 Amps was programmed for both G_{s1} and G_{s4} OTAs.

were plotted over the experimental data. An I_{bias} = 0.05e-07 Amps was programmed for all OTAs for this experiment. Estimates of G_{s1}, G_{s4}, and parasitic capacitance C_{sp} were made using the experimental data in Figure 76. Estimates of G_{s2}, G_{i1}, and parasitic capacitance (C_i + C_{ip}) were made using the experimental data in Figure 77 and the estimates from the experimental data in Figure 76. Estimates of G_{d1}, G_{d2}, G_{d3}, G_{s3}, and parasitic capacitances C_d and C_{dp} were made using the experimental data in Figure 78.

5.3.3 Acting

Action takes place in the robot frame. The system has control of the Pioneer robot's forward/reverse velocity and also its angular velocity. Regarding the robot arm, the arm joint angles are commanded from the control program. Existing low level arm control routines were used. Images of the robot Acting (grasping) a disk are found in Figure 68. Inverse

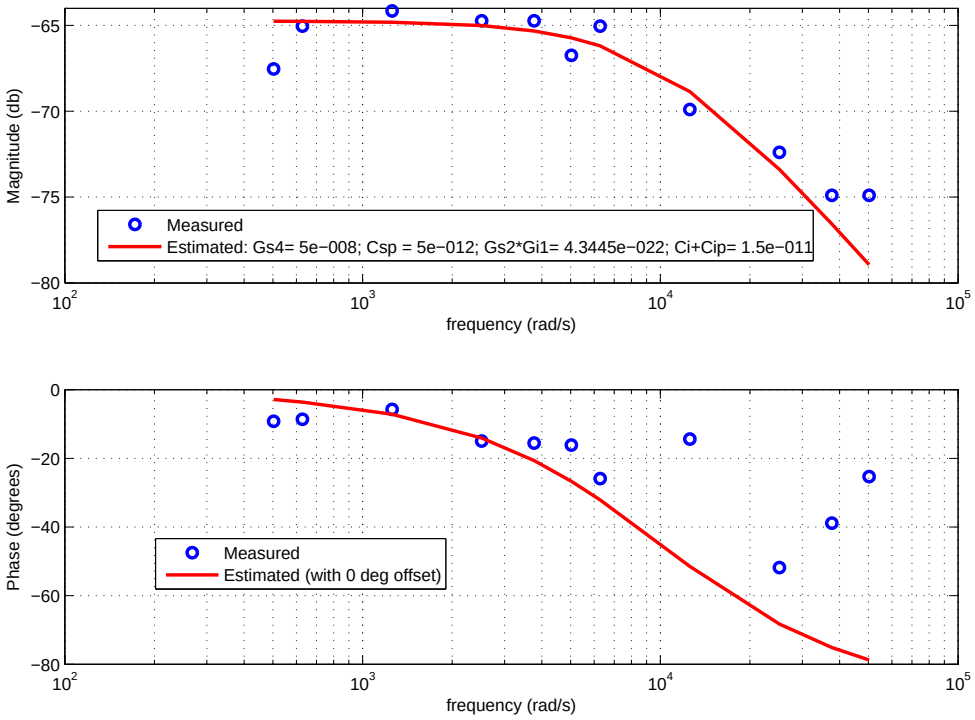


Figure 77. Bode Plot of the Analog Integral Controller. The gains G_{s2}*G_{i1} and the parasitic capacitance (C_i+C_{ip}) are estimated and used to plot a curve over the experimental data. An I_{bias} = 0.05e-07 Amps was programmed for G_{i1}, G_{s2}, and G_{s4} OTAs. The curve being fit here is actually the transfer function for the Integral only controller multiplied by *s*.

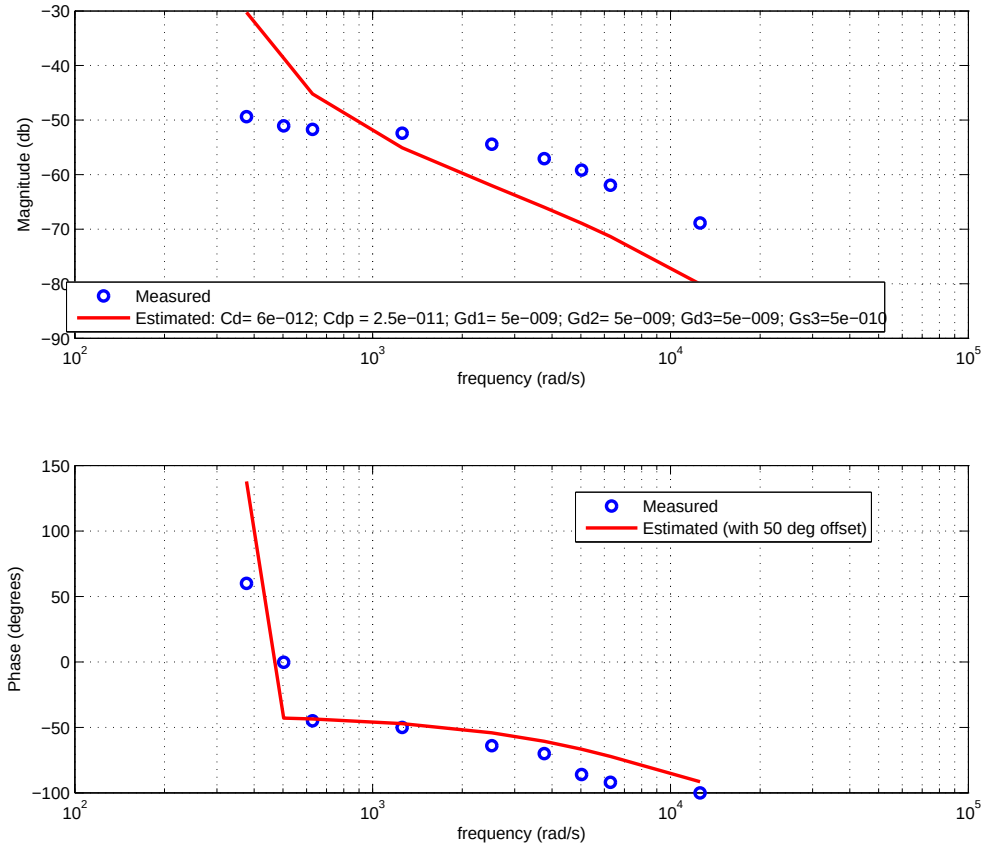


Figure 78. Bode Plot of the Analog Derivative Controller. The gains G_{d1}, G_{d2}, G_{d3}, G_{s3} and the parasitic capacitances C_d and C_{dp} are estimated and used to plot a curve over the experimental data. An I_{bias} = 0.05e-07 Amps was programmed for the OTAs. The curve being fit here is actually the transfer function for the Derivative only controller multiplied by *s*.

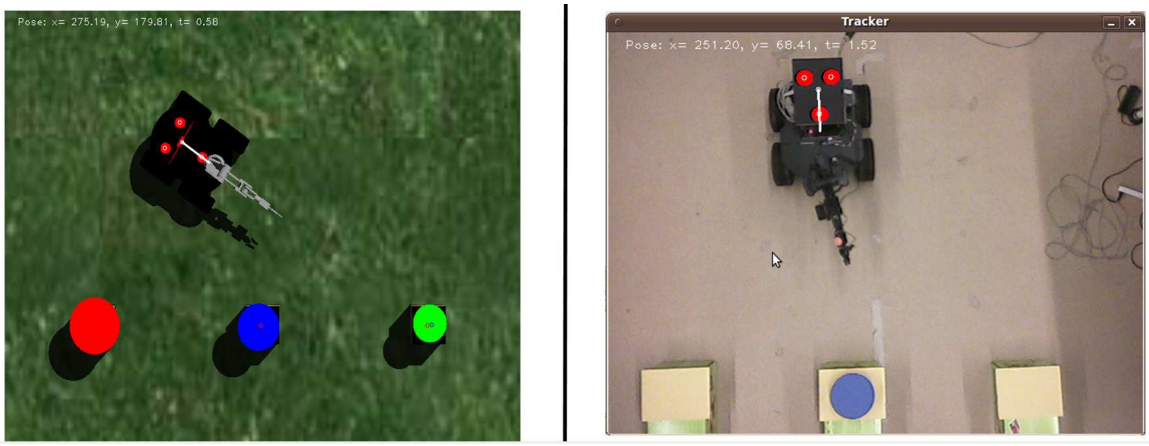


Figure 79. This figure compares the tracker images from the overhead camera in the simulation to real life overhead camera hardware.

kinematics are used for two joints so that the end effector has a desired height, and the gripper is parallel to the ground. The height of the disk is problem specific and is hardcoded in this routine.

5.4 Hardware Implementation

The next step, after successfully completing the problem in the Player/Gazebo simulation, was to try the algorithms on a real robot, Figure 68. The robot successfully completed a two disk Towers of Hanoi problem. A Logitec model V-UBV49 Webcam was used for the camera. It was mounted to a pole on the ceiling of the lab. Figure 79 shows a comparison between the Gazebo simulation camera image and the actual image from the Logitec webcam.

There were some notable differences between the simulation and real world environments. Regarding sensing, in the simulation environment one can specify perfect illumination and ideal color values. This is not the case in a real world lab environment. In the lab one has to contend with shadows and broader color range values. The coded range for color values had to be changed for the real world control code. The hardware also behaves differently in the simulation vs. real world. The Proportional and Derivative gains (K_P, K_D) for the closed loop control system in the real hardware needed to be modified from their

simulation values.

Figure 72 shows the hardware/software design flow concept for implementing an OTA based PID controller on an FPAA. Figure 72a shows the desired circuit. Figure 72b shows the equivalent Matlab Simulink model. Figure 72c Shows the Spice level model automatically generated from the Simulink model, Figure 72d shows the low level switch list for programming the FPAA, and finally, Figure 72e shows a picture of the utilization of the FPAA IC by plotting the switch list.

5.5 Summary

This chapter presented a mobile manipulator that solves the classic Towers of Hanoi problem. The effectiveness of the Player/Gazebo simulation to real hardware design cycle was demonstrated. The process of identifying what needed to be changed to make the simulation control software work on real hardware was educational. This may lead the authors to consider during the simulation phase of a project how certain aspects of the design can be parameterized to best facilitate the transition from simulation to real hardware. Future work may consider using a camera mounted near the end effector to aid in grasping. Turn taking can be explored where the robot moves a disk and then the human moves a disk for interactive game play. Finally, the FPAA can be fully integrated into the platform for low-level control.

CHAPTER 6

IMAGE PROCESSING

This chapter describes an image processing system being designed for an off-the-shelf quadrotor. A mixed signal approach to the embedded image processing is presented. Constraints on image processing computation include: power efficiency, short term storage (i.e. part of an image), and movement of data. This algorithm performs a type of convolution on the image and also subsamples it. "The computational approach is similar to some computational attention mechanisms [179]. Computing pre-attention fields are typically large convolution and subsample filters corresponding to agregated (sic) cortical receiptive (sic) fields [180]." This processing has been referred to as "center-surround" processing and results in feature maps [181].

6.1 FPAA Based Image Processing Algorithm

The analog computation uses a data flow architecture that merges memory and computation [180]. A mixed signal processing algorithm has been developed that uses analog elements to process and subsample an image. A non-overlapping *n* by *n* kernel is used in the subsampling process. To illustrate the output of the system, consider Figure 80, where a 24 by 24 pixel image is reduced to a *3* by *3* matrix through the image processing algorithm and an *8* by *8* kernel. Two key innovations of this architecture are: first, the vector-matrix multiply computation is performed by analog elements (floating-gate transistors), and secondly partially processed data is stored on analog elements. The reason for doing this is to reduce the time needed to process data because fewer time intensive digital memory access steps are needed.

6.1.1 Subsampling Algorithm

The subsampling operation transforms an *n* by *n* block of the image into a single scalar value, *x*. Although not a strict requirement, the system is presented where each of these

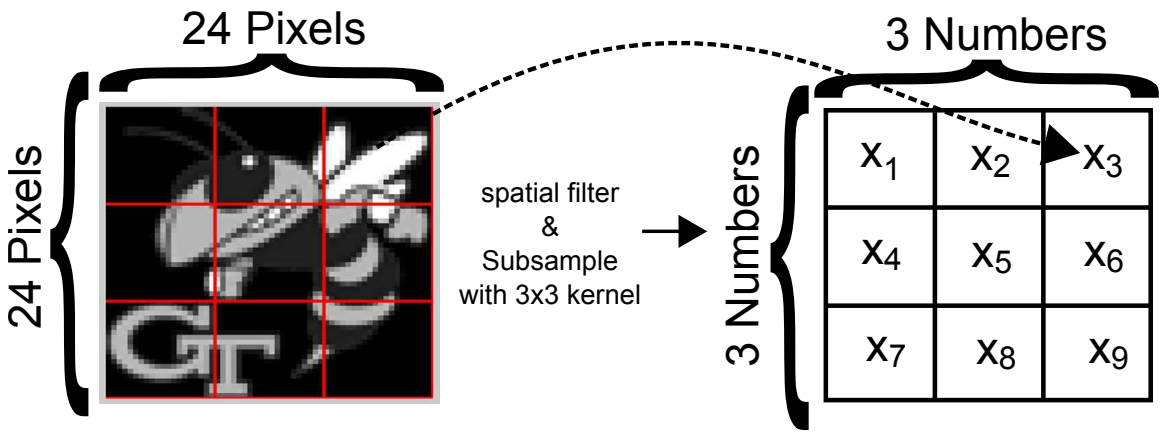


Figure 80. Big picture of the subsampling operation of the mixed signal image processing algorithm.

blocks are non-overlapping.

Let a block of image data be represented by (34),

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \tag{34}$$

and the image processing kernel is represented by (35).

$$\mathbf{B} = \begin{bmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,n} \end{bmatrix} \tag{35}$$

Let **C** be a vector which represents intermediate processing, (36), where the individual terms are computed as in (37).

$$\mathbf{C} = [c_1, \cdots, c_n] \tag{36}$$

$$c_i = \sum_{j=1}^{n} a_{i,j} b_{i,j} \tag{37}$$

The final subsampling scalar value, (38), is produced by summing the terms of (36).

$$x = \sum_{q=1}^{n} c_q \tag{38}$$

This subsampling and convolution process is illustrated in Figure 81. A motivating

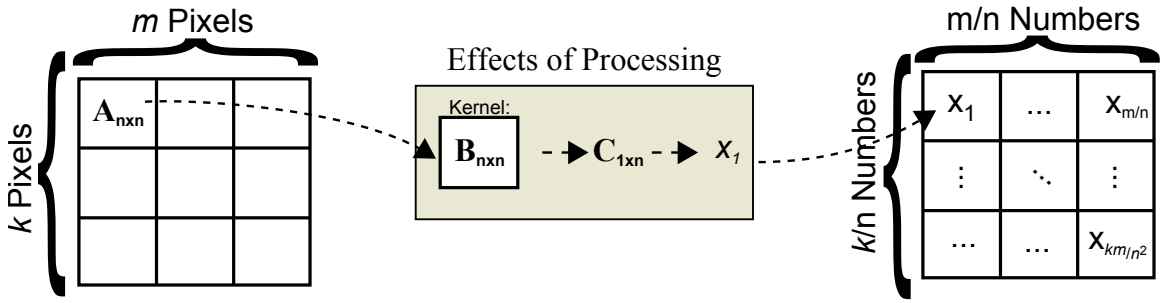


Figure 81. The equations showing the effects of the image processing and subsampling.

reason for this image processing process is to be able to track objects with a camera. To this end, it is important to identify an appropriate kernel and understand the conditions under which this convolution process and kernel can perform this task. Digital simulations show that a Laplacian of Gaussian (LoG) kernel can identify an object with sufficient high frequency (edge) components. Figure 82 shows Matlab results of using the algorithm with a 15x15 LoG kernel function (39). The Laplacian, (3), in addition to being useful for path planning, is also useful for finding the edges of an image. Convolving a Laplacian kernel with an image finds regions of rapid intensity change, and edges represent regions of rapid intensity change [182, 183]. A problem with Laplacian kernels is that their performance is reduced by noise. To combat this, one may smooth the image first (low pass filter) and then apply the Laplacian kernel. It is possible to combine the smoothing and Laplacian kernel into a single kernel, the LoG kernel (39). Figure 81a shows a 120x160 image which was first re-sized to 480x640. Figure 81b shows the image sub-sampled into a grid of 32x42 blocks. Figure 81c shows the LoG kernel. To create this kernel, first (39) was used with $x = -7:7, y = -7:7, \sigma = 1.4$. Next the kernel was normalized by the minimum value. Finally, this was multiplied by 40 to arrive at the figure [182, 183]. Figure 81d is the result of applying the LoG kernel to the image in Figure 81a. This identified edges. Figure 83 shows additional Matlab results of using the algorithm with a 15x15 LoG kernel function (39). This illustrates that the LoG kernel can be used to identify objects of interest if the objects contain enough edge (or high frequency components).

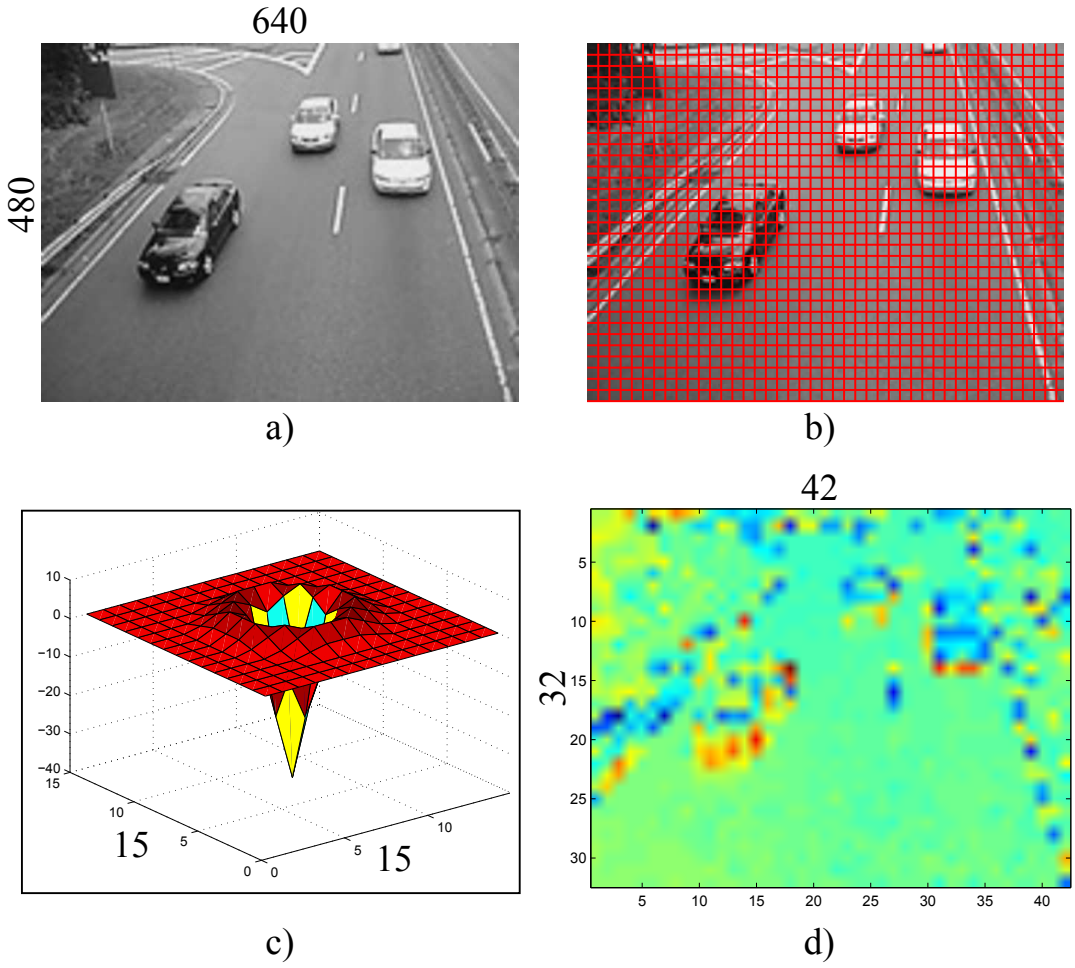


Figure 82. Matlab simulation of image processing: a) 480x640 Image, b) Non-overlapping blocks to be processed, c) 15x15 Laplacian of Gaussian (LoG) kernel with $\sigma = 1.4$ used for processing, and d) 32x42 processed and subsampled image output.

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{39}$$

6.1.2 Circuit Architecture for the Subsampling Algorithm

The section above shows the equations and effects of the subsampling algorithm, but the mechanics of this process are that each block is not computed one at a time. Instead, parts of *m*/*n* blocks are processed sequentially until all of them are finished. One of the innovations of this algorithm is that the intermediate values for these *m*/*n* blocks are saved with an analog memory (a capacitor). Once all of the processing for the *m*/*n* blocks is

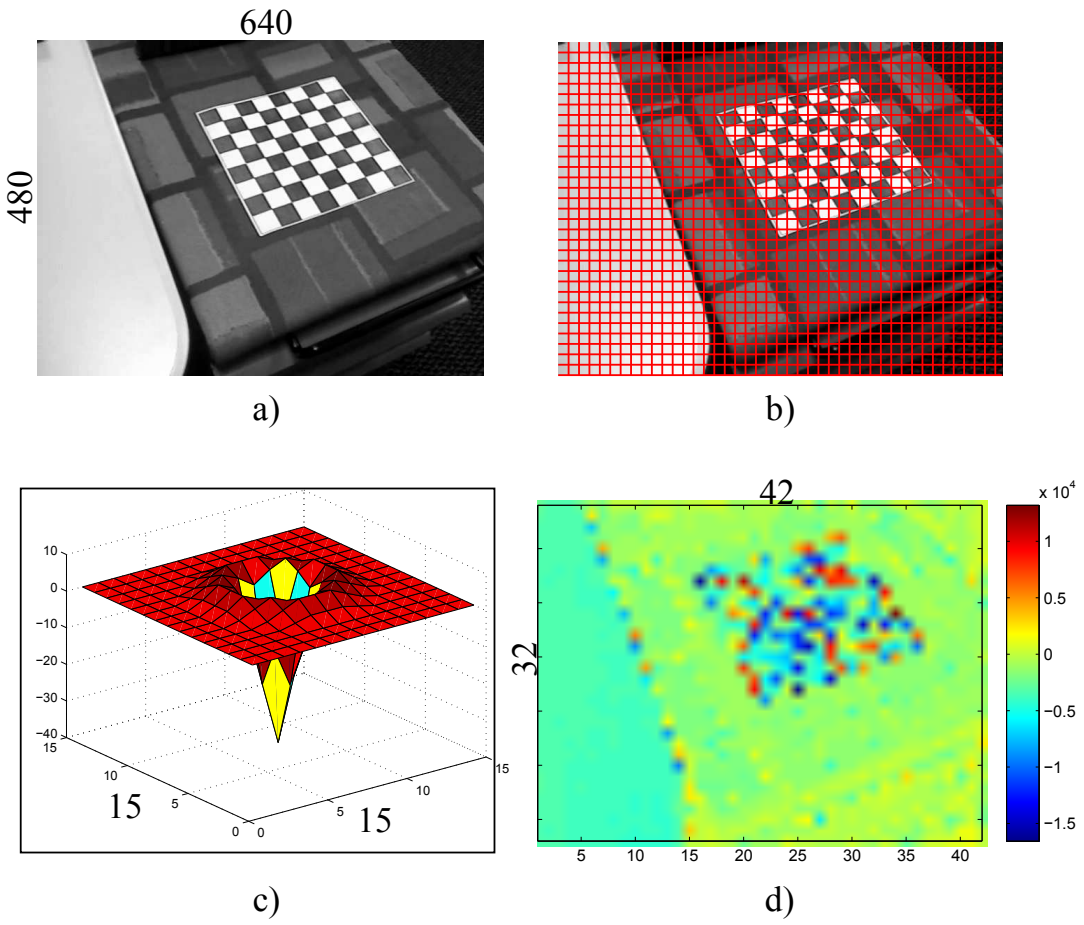


Figure 83. Matlab simulation of image processing: a) 480x640 Image, b) Non-overlapping blocks to be processed, c) 15x15 Laplacian of Gaussian (LoG) kernel with $\sigma = 1.4$ used for processing, and d) 32x42 processed and subsampled image output.

finished, these *m/n* scalar values can either be converted to digital values and stored in digital memory or transferred to another analog memory. Figure 84 shows a diagram of the image processing architecture.

Figures 85 and 86 describes the signal flow for processing the first row of Figure 80. The pixel data from the imager is streamed in serially. Once it streams in all of the first row it moves to the second row, etc. The first *n* pixels of the first row are multiplied with an analog multiplier and summed as charge on a capacitor. Then the next *n* pixels are multiplied and stored on another capacitor. This continues until the first row is processed (Figure 85). The second row of data is processed in a similar way, Figure 86. At the end of this process each of the three capacitors in this example contains a voltage which represents a convolved and subsampled pixel. These values would then be converted to a digital value using an Analog to Digital Converter (ADC) and sent to digital memory or to a microprocessor. The capacitors would be cleared and the process would repeat for the second row of blocks.

The architecture uses floating-gate transistors (see Section 2.5) for the weights. Sub-threshold operation of the transistors is also part of the design. Equations describing this region of transistor operation are described in the following section.

6.1.2.1 Subthreshold Transistors

Most of the transistors used to create the image processing circuits are operating in the subthreshold region of operation. One of the advantages of this region of operation is that the transistors are conducting less current (and hence, using less power) than in above threshold operation.

When the channel (i.e. area under the gate of a transistor) is in weak inversion then one can say that it is operating in *subthreshold* [122]. Weak inversion happens when a positive voltage is applied to the gate, and “This charge repels the holes in the substrate and leaves behind negatively-charged ions, that balance out the gate charge. The MOSFET operates in the subthreshold regime when the positive charge on the gate is almost balanced by the

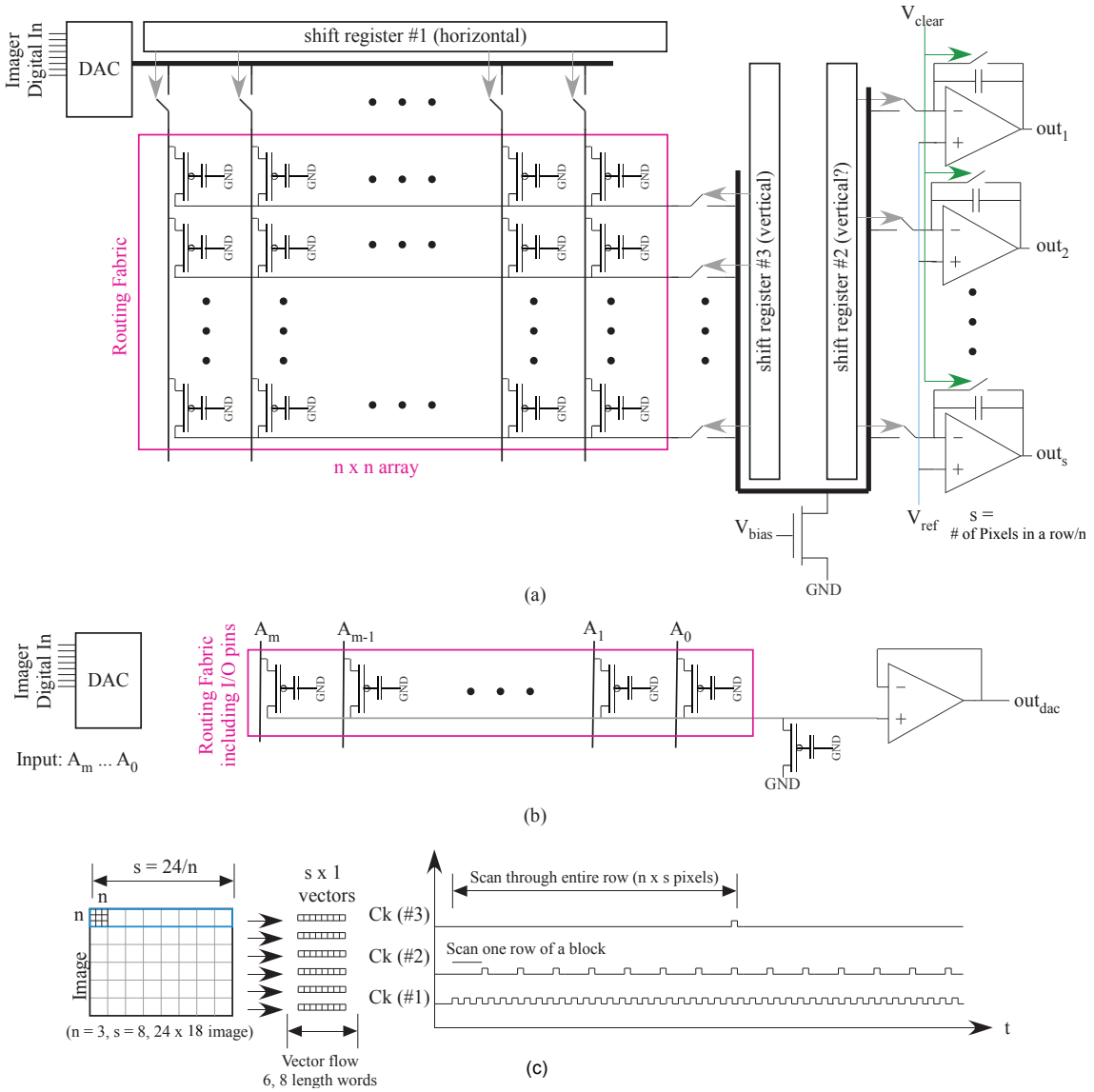


Figure 84. FPAA image processing signal flow architecture. a) This shows the entire architecture. The vector matrix multiplication is performed by the n by n array of weighted floating-gates. The shift registers, when clocked as in c, route the signals through the proper kernel matrix weight and then to the proper analog memory element. b) This is the Digital to Analog Converter (DAC) schematic which converts m+1 bit digital voltages to log compressed values. c) This shows the registers' clock timing for an example 24x18 image and a 3x3 kernel.

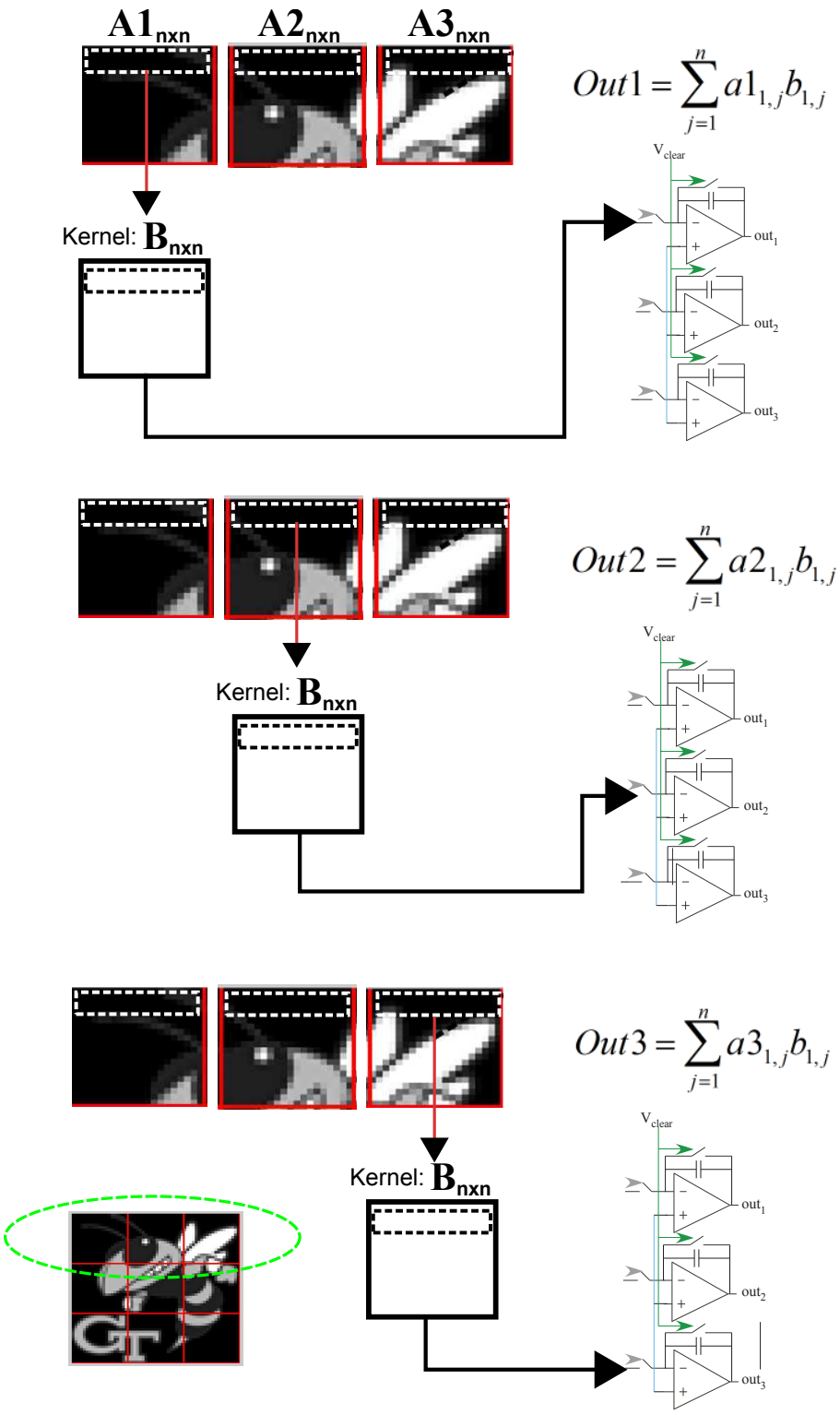


Figure 85. The input data is streamed in from the imager one row at a time. Parts of each subblock are processed in series. This shows processing row ONE of blocks (1,1) (1,2) and (1,3).

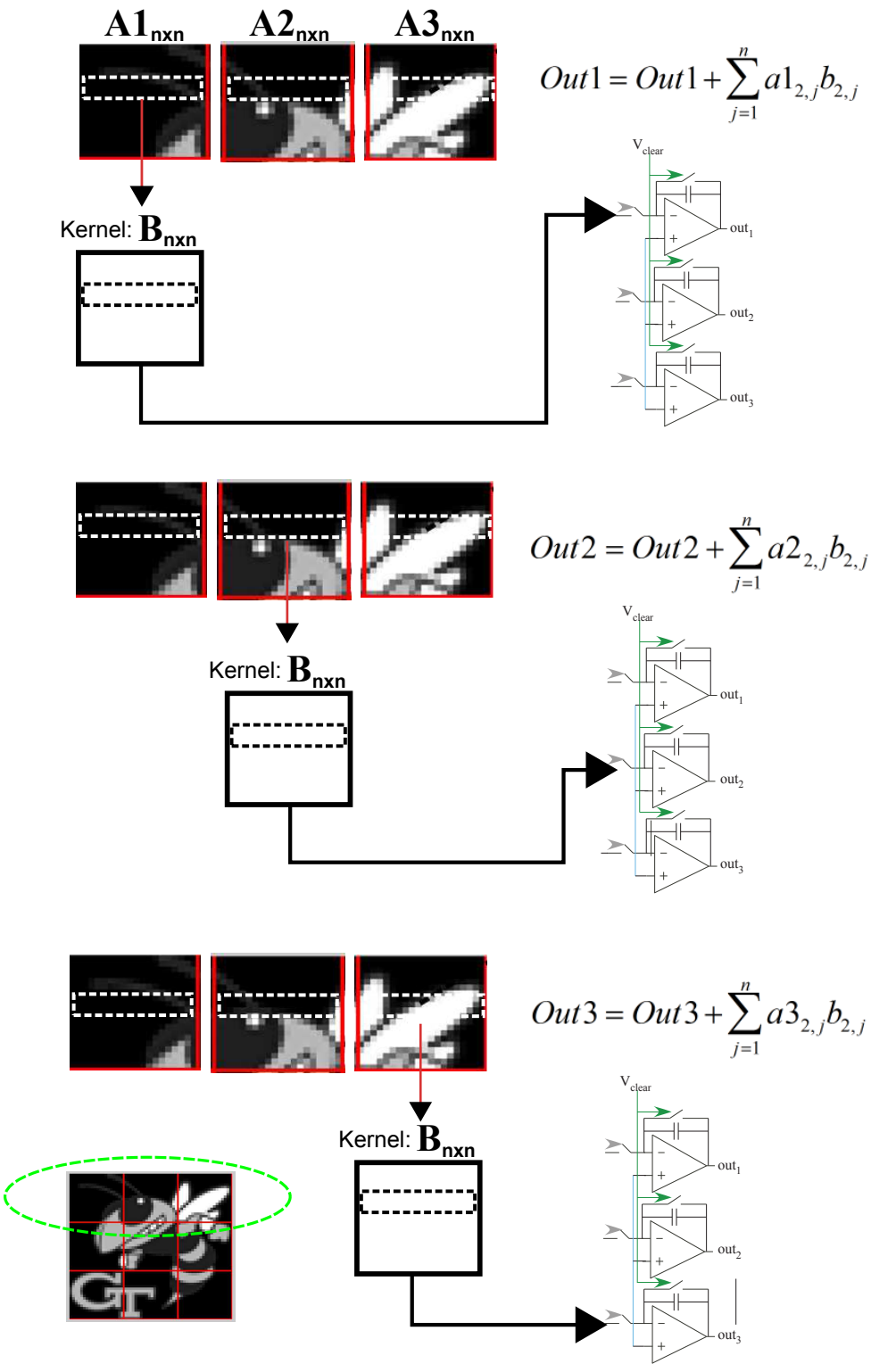


Figure 86. This illustrates the next step after Figure 85. This shows processing row TWO of blocks (1,1) (1,2) and (1,3).

negatively-charged depletion region underneath the gate. There is also a very thin layer of electrons beneath the gate (the *inversion layer*). In subthreshold, we ignore the charge from the inversion layer because it is almost negligible compared with the depletion charge [122]." Assuming drift current is zero (because one assumes the potential in the channel, surface potential, is zero): "The electron concentrations at the two ends of the channel depend on the energy barrier that the electrons encounter. This barrier is determined by the voltage difference between the surface potential and the applied voltages V_s and V_d...[122]."

The current in a pFET transistor in subthreshold operation may be described as a sum of the forward and reverse current in the transistor:

$$I = I_f - I_r \tag{40}$$

Expanding (40):

$$I = I_0 \left[e^{\frac{\kappa(V_w - V_g) - (V_w - V_s)}{U_T}} - e^{\frac{\kappa(V_w - V_g) - (V_w - V_d)}{U_T}} \right] \tag{41}$$

Where in (41), I_0 is a constant representing *preexponential factors* [121, 122], and V_w is the bulk voltage (n-well), usually Vdd [122]:

$$I_0 = q\frac{W}{L}tD_nN_1e^{\frac{\psi_0}{U_t}} \tag{42}$$

And κ is a constant representing the "capacitive coupling ratio from gate to channel [122]":

$$\kappa = \frac{C_{ox}}{C_{ox} + C_d} \tag{43}$$

And U_t is the *thermal voltage* [122]:

$$U_T = \frac{kT}{q} \tag{44}$$

If V_g, V_d, and V_s are referenced to the bulk voltage [122]:

$$V_g \equiv V_g - V_w \tag{45}$$

$$V_d \equiv V_d - V_w \tag{46}$$

$$V_s \equiv V_s - V_w \tag{47}$$

then (41) can be re-expressed as:

$$I = I_0 \left(e^{\frac{-\kappa V_g + V_s}{U_T}} - e^{\frac{-\kappa V_g + V_d}{U_T}} \right) \tag{48}$$

and further re-expressed as:

$$I = I_o e^{\frac{-\kappa V_g + V_s}{U_T}} \left[1 - e^{\frac{V_{ds}}{U_T}} \right] \tag{49}$$

Substituting $\frac{V_{ds}}{U_T} = \frac{-V_{sd}}{U_T}$ into (49) gives a compact form of the current in a subthreshold-operating pFET.

$$I = I_o e^{\frac{-\kappa V_g + V_s}{U_T}} \left[1 - e^{\frac{-V_{sd}}{U_T}} \right] \tag{50}$$

If the drain to source voltage is larger than a certain value (51), then we say the transistor is operating in *saturation* and can simplify (7) to (52).

$$V_{sd} > 4U_T \tag{51}$$

$$I = I_0 e^{(-\kappa V_g + V_s)/U_T} \tag{52}$$

An nFET transistor in subthreshold operation may be described as (53) or (54).

$$I = I_o \left[e^{\frac{\kappa V_g - V_s}{U_T}} - e^{\frac{\kappa V_g - V_d}{U_T}} \right] \tag{53}$$

$$I = I_o e^{\frac{\kappa V_g - V_s}{U_T}} \left[1 - e^{\frac{-V_{ds}}{U_T}}\right] \tag{54}$$

6.1.2.2 Power Analysis

The power analysis for the analog system can be broken down into different sections of the algorithm such as: Digital to Analog Conversion (DAC), matrix multiplication and addition analog computation, Analog to Digital Conversion (ADC), memory read-write, and digital computation. The analog numbers can then be compared to ones from an all digital approach. Before continuing with the analog power analysis, an all digital computation can be addressed as follows [180]. First, assume a 640x480 VGA image (i.e. 307,200 pixels) operating at 60 Frames Per Second (FPS). This results in 18,432,000 pixels per second. Second, assume that the system requires approximately one Multiply and ACcumulate (MAC) per pixel. Based on the number of pixels and the fps, this system would require approximately 20 Million MAC/s, with sufficient digital resolution for summation of large number of values. If one assumes the digital system requires approximately 4 memory read/write's per operation then this results in approximately 80M MIPS (Million Instructions Per Second) of computation. If using an Atmel ARM9 SC9 processor (TSMC 130nm) with 262 DMIPS (Dhrystone MIPS) [184], then one may estimate that this image computation is possible with 1/3 of the full speed ARM9 processor.

Now the power analysis of the analog system is addressed. Figure 84a shows the circuit architecture. An integral part of this is the timing diagram in Figure 84c which controls the switching of the three registers. This switching routes the signals from the DAC through the matrix weight and onto the correct integrating capacitor. Figure 87 simplifies this diagram to show the signal routing for a single pixel value. The *m* bits of digital data go through a source follower circuit and buffer. This signal is weighted by a floating-gate transistor and then integrated onto a capacitor. To help explain the signal flow, the *a*, *b*, and *c* labels correspond to the same letters used to describe the algorithm in (34) through (38).

Figure 88 shows the main DAC, MAC, and temporary storage circuits of the analog

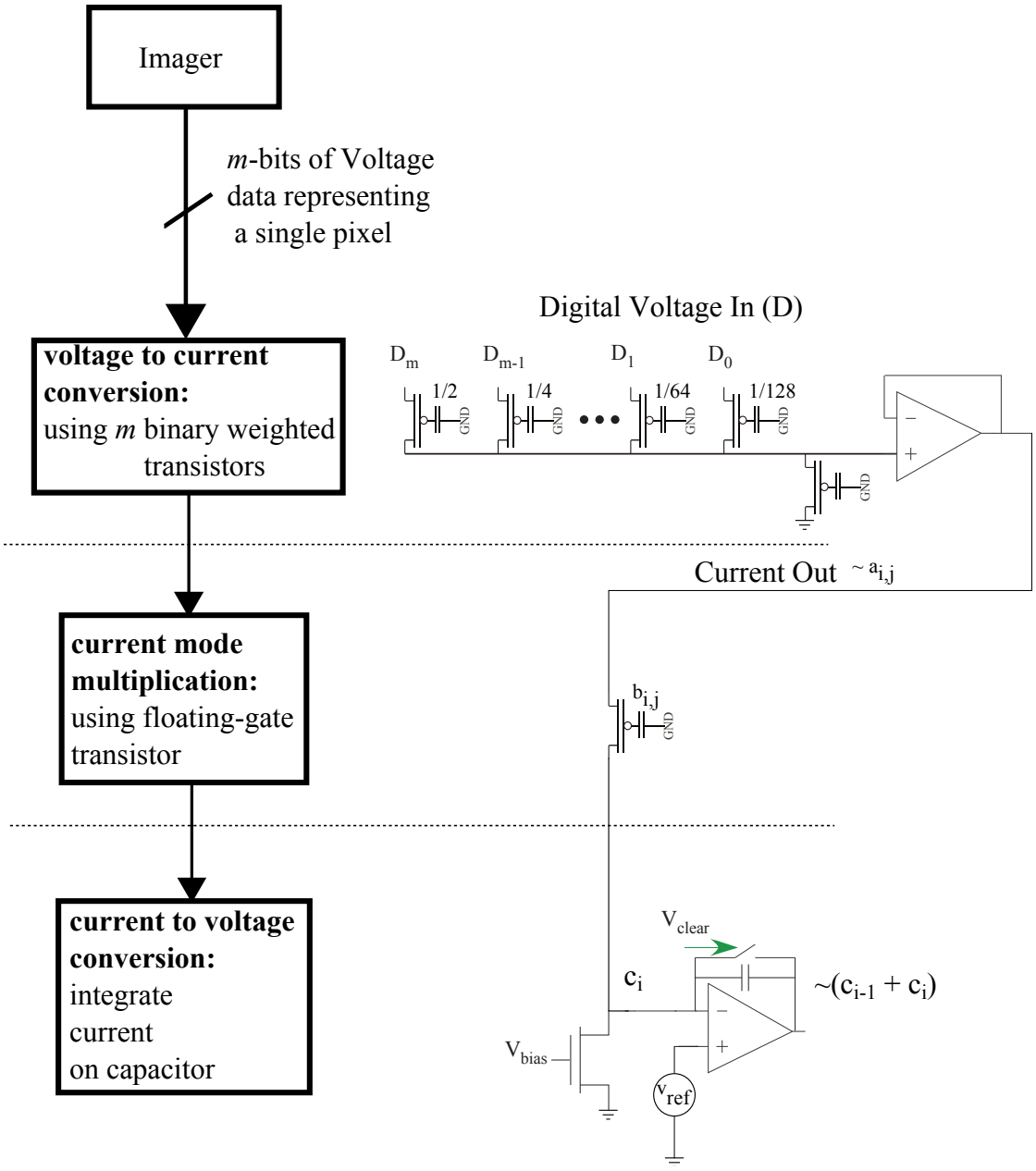


Figure 87. Signal Flow for processing a single pixel.

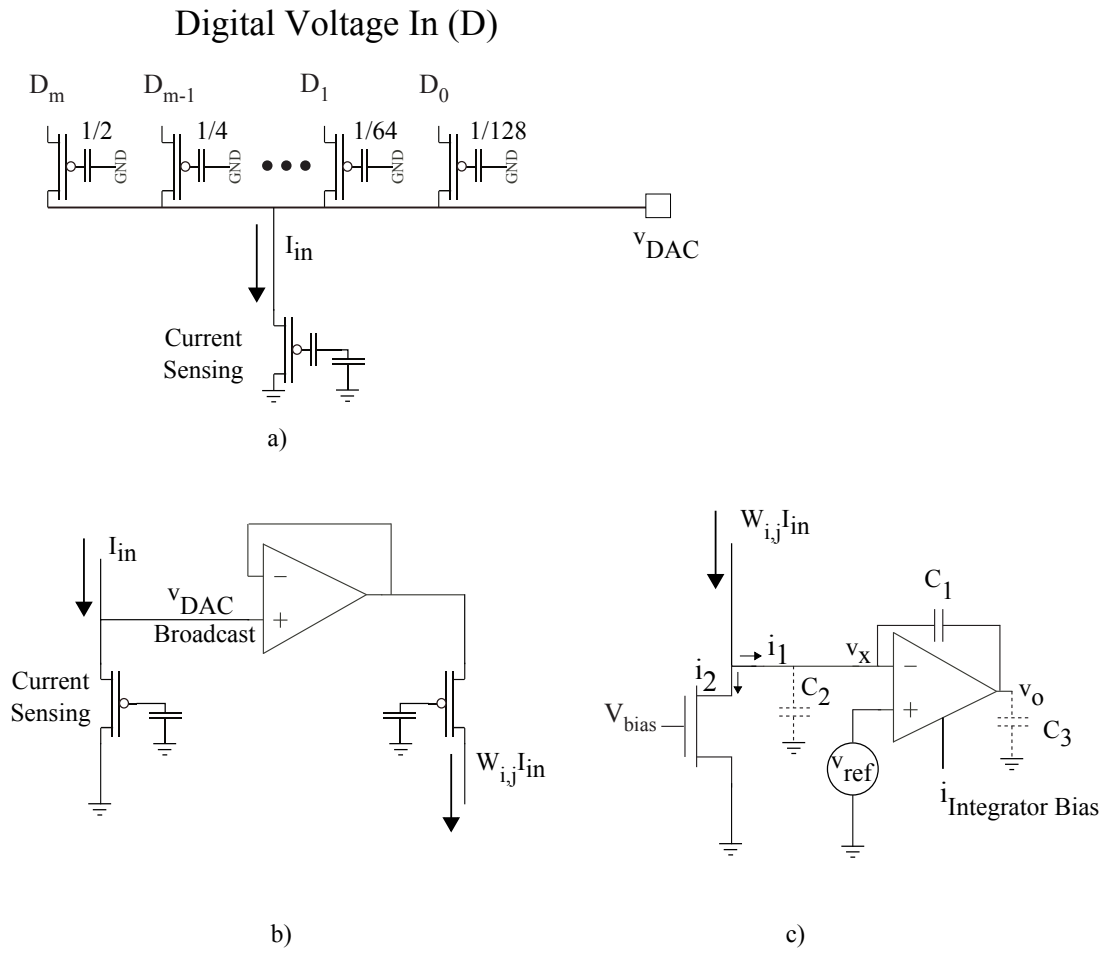


Figure 88. The main circuit components which perform the analog image processing are: a) Source follower circuit which performs Digital to Analog Conversion current sensing, b) Source-coupled topology for a weighted current mirror [15], and c) Storage circuit for weighted current. The integrator circuit which holds the intermediate processed values. C_2 and C_3 are parasitic capacitances.

image processing system. Each of these can be analyzed separately to obtain equations for current and power (assuming an operating frequency). Figure 88a shows a source follower circuit, operating in the subthreshold domain [122], which converts *m* digital bits into an analog voltage [15]. Source followers are typically used as impedance converters which convert a weak voltage signal into a stronger driven signal [122]. In typical source followers, the bias (i.e. sensing) transistor is operated in saturation (52). The standard equation for a two transistor source follower is (55), where V_g is the input.

$$V_{out} = U_T \log\left(\frac{I_b}{I_0}\right) + \kappa V_g \tag{55}$$

When more than one bits are feeding into the source follower, one can express the equation as (56).

$$I_{01}e^{\frac{\left(-\kappa V_{g1}+V_s\right)}{U_T}} + I_{02}e^{\frac{\left(-\kappa V_{g2}+V_s\right)}{U_T}} = I_b \tag{56}$$

For two bits, the equation for the DAC output is (57).

$$V_{DAC} = U_T \log\left(\frac{I_b}{I_{0x}}\right) + U_T \log\left[\left(e^{\frac{-\kappa_1 V_{g1}}{U_T}} + e^{\frac{-\kappa_2 V_{g2}}{U_T}}\right)^{-1}\right] \tag{57}$$

The power used by the source follower is based on the current setting of the bias, or "sensing" transistor.

Figure 88b shows a floating-gate source-coupled current mirror [15]. Schlottmann and Hasler describe two implementations of floating-gate source-coupled current mirrors in [15]. The type used in Figure 88b is what they refer to as a "Buffered input state" as opposed to their "Log-amp input stage." "Source coupling involves forcing the input current into the source of the sensing FET, then buffering the source voltage to the output stages [15]." The bandwidth of the buffered input state source coupled current mirror is (58) [15].

$$\omega = \frac{g_m}{C}; \qquad \text{where} \qquad g_m = \frac{I_b}{U_T} \tag{58}$$

The weight for the multiplication in the analog circuit is determined by the floating-gate voltages on the sensing pFET and the mirrored pFET. The equation for the weight is (59) [185, 15].

$$W \equiv \frac{I_{out}}{I_{in}} = e^{\frac{\kappa\left(V_{fg,out}-V_{fg,in}\right)}{U_T}} \tag{59}$$

The OTA in Figure 88b is configured in a Unity-Gain Follower, or "buffer" configuration. It is used to buffer the DAC voltage. In this feedback configuration, the OTA is used to supply the necessary current in order to maintain V_{DAC} at the output of the OTA. An OTA has a high input impedance (ideally infinite) so it is also used to isolate the source follower

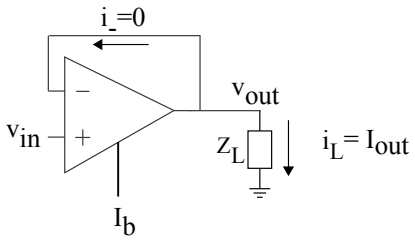


Figure 89. The unity gain follower configuration of an Operational Transconductance Amplifier (OTA).

circuit from the multiply and accumulate circuit which follows. “In contrast to the source follower...which is also used as an impedance converter, the unity-gain follower does not introduce a large voltage offset [122].” The OTA’s current output can be modeled with a *tanh* curve (60) [121], and the input-output transfer function can be calculated as follows. Given the buffer in Figure 89 and a load, Z_L, the following equations can be stated. The static output current of an OTA is found in (60) [186]. This equation is a *macromodel* of the amplifier because it is based on the output terminal’s characteristics [17]. In a five transistor implementation of an OTA, this assumes: 1) all MOSFETs are in saturation, and 2) the differential pair is operated below threshold [122].

$$I_{out} = I_B \tanh\left(\frac{\kappa}{2U_T}(V_{in} - V_{out})\right) \tag{60}$$

$$I_{out} = \frac{V_{out}}{Z_L} \tag{61}$$

Linearizing the OTA current output with the constraints in (68) and combining (61) and (60):

$$\frac{1}{Z_L}V_{out} = \frac{I_B\kappa}{2U_T}(V_{in} - V_{out}) \tag{62}$$

$$\frac{V_{out}}{V_{in}} = \frac{\frac{I_B\kappa}{2U_T}}{\frac{1}{Z_L} + \frac{I_B\kappa}{2U_T}} \tag{63}$$

If the load, Z_L, is a purely resistive load, and one lets it approach infinity, one arrives at the unity gain transfer function, (64).

$$\lim_{R_L \to \infty} \frac{\frac{I_B \kappa}{2U_T}}{\frac{1}{R_L} + \frac{I_B \kappa}{2U_T}} = 1 \tag{64}$$

Figure 88c shows the temporary storage circuit. The memory storage element is a key feature in this image processing architecture. An integrating amplifier [187] is used to hold and sum the intermediate values for each block's processing. Using Kirchhoff's Current Law (KCL), three equations can be written to describe the currents at the output of the amplifier, and at the inverting terminal, (65),(66),(67). C_2 and C_3 represent parasitic capacitances.

$$I_o = I_B \tanh\left(\frac{\kappa}{2U_T}\left(V_x - V_{ref}\right)\right) \tag{65}$$

$$I_o = C_3 \frac{dV_o}{dt} + C_1 \frac{d\left(V_o - V_x\right)}{dt} \tag{66}$$

$$I_{in} = C_2 \frac{dV_x}{dt} + C_1 \frac{d\left(V_x - V_o\right)}{dt} \tag{67}$$

One may linearize (65) to (69) if (68) holds, and the other three equations may be re-arranged too (70) (71) .

$$\left|V_x - V_{ref}\right| < \frac{2U_T}{\kappa} \tag{68}$$

$$I_o = \frac{I_B \kappa}{2U_T}\left(V_x - V_{ref}\right) \tag{69}$$

$$I_o = \left(C_3 + C_1\right)\frac{dV_o}{dt} - C_1 \frac{dV_x}{dt} \tag{70}$$

$$I_{in} = \left(C_2 + C_1\right)\frac{dV_x}{dt} - C_1 \frac{dV_o}{dt} \tag{71}$$

Substituting (69) into (70) leaves one with two equations and two unknowns, V_o and V_x:

$$I_{in} = (C_2 + C_1)\frac{dV_x}{dt} - C_1\frac{dV_o}{dt} \tag{72}$$

$$\frac{I_B\kappa}{2U_T}\left(V_x - V_{ref}\right) = (C_3 + C_1)\frac{dV_o}{dt} - C_1\frac{dV_x}{dt} \tag{73}$$

Further simplifying yields the following two equations:

$$C_1\frac{dV_x}{dt} + \frac{I_B\kappa}{2U_T}V_x - \frac{I_B\kappa}{2U_T}V_{ref} = (C_3 + C_1)\frac{dV_o}{dt} \tag{74}$$

$$\frac{dV_o}{dt} = \frac{I_{in}}{-C_1} + \frac{(C_2 + C_1)}{C_1}\frac{dV_x}{dt} \tag{75}$$

Equation (75) can be substituted into (74) to give the equation representing the intermediate processed value which is stored on the capacitor:

$$\left(C_1 - \frac{(C_3 + C_1)(C_2 + C_1)}{C_1}\right)\frac{dV_x}{dt} + \frac{I_B\kappa}{2U_T}V_x - \left(\frac{I_B\kappa}{2U_T}V_{ref} - \frac{(C_3 + C_1)I_{in}}{C_1}\right) = 0 \tag{76}$$

Eq (76) is a first-order linear differential equation of the form (77), which has a solution of the form (78):

$$a\frac{dx}{dt} + bx - c = 0 \tag{77}$$

$$x(t) = k_1 e^{-\frac{bt}{a}} + \frac{c}{b} \tag{78}$$

Four-quadrant multiplication is desired in this circuit. One way to accomplish this with analog vector matrix multiplication (VMM) is to use differential signals and adopt a convention where the signed signal is the difference between two positive currents [15]. The architecture in this chapter does not use differential signals but instead uses a bias nFET (Figure 88c) to help allow the system to produce negative weights. The nFET is biased to

be in saturation. The idea is that it can be set such that for positive weights the currents integrate onto the summing capacitor, and for negative weights charge is pulled off of the capacitor. The key is, similar to [15], to establish signals which are small changes around a bias current. Similarly, the positive and negative weights will need to be represented by deltas around a bias weight.

Now that the DAC, MAC, and intermediate storage circuits have been described one may analyze the system power of these elements. (For now, let us ignore the ADC and further digital processing stages.) The power-delay product, (79), is used as a power performance metric, where speed comes at the expense of power [17, 15].

$$P_\tau = IV\tau \tag{79}$$

In digital circuits, τ represents the propagation delay. In this analog circuit, it represents the delay caused by resistors and capacitors in the circuit and is inversely related to the maximum operating frequency of the architecture. In finding the maximum operating frequency, one analyzes the bandwidth of the circuit to find the -3dB point. The "standard recipe" for computing the bandwidth is (from [16]):

1. Derive the input-output transfer function in terms of s (use node equations)
2. Set $s = j\omega$
3. Find the magnitude of the result of step 3
4. Set the magnitude = $1/\sqrt{2}$ of the "midband" value
5. solve for ω

As an approximation to the above steps, one may also estimate the bandwidth using *The Method of Open-Circuit Time Constants* [16, 17]. A high frequency MOSFET model is shown in Figure 90 [16, 17]. If the input stage of Figure 87 (i.e. the buffered input stage) is the dominant factor in the frequency response, then one can use it to estimate the

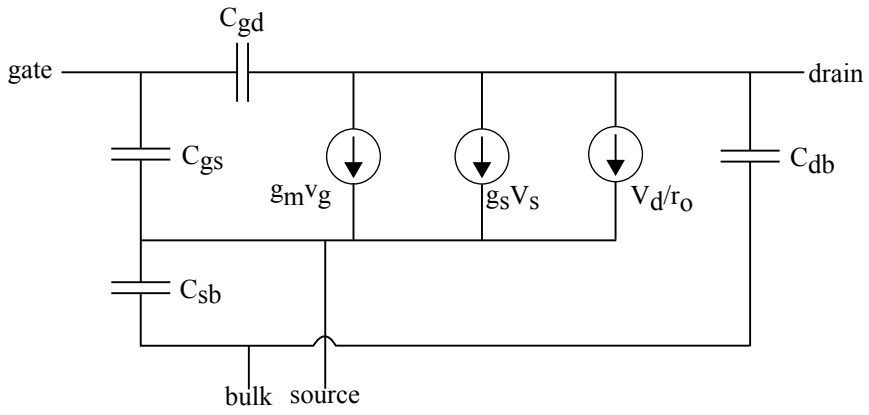


Figure 90. High frequency MOSFET model [16, 17].

operating power [15, 188]. Following the analysis in [15, 188], one may find the -3 dB frequency of the OTA from (58).

$$\tau = \frac{U_T C}{I_b} \tag{80}$$

$$f_{-3dB} = \frac{1}{2\pi\tau} = \frac{1}{2\pi}\frac{I_b}{U_T C} \tag{81}$$

The currents cancel out in the power-delay product calculation (82) and show that one can reduce the power in this system by reducing the capacitance and/or reducing the capacitance seen by the buffer. "The inverse of this product can also roughly be considered the computation per unit power [15]."

$$P_\tau = V_{dd} U_T C \tag{82}$$

If one includes a factor of two to account for the power used by the output stage (the current integrator), the power is estimated to be (83).

$$P_\tau = 2V_{dd} U_T C \tag{83}$$

A summary of estimated performance metrics is shown in Table 8. If the input stage is changed from a Buffered input stage to a Log-amp input stage then we would expect to

Table 8. Estimated Performance Parameters for VMM with Inputs Coming in Serially; with Buffered Input stage using: $C = 1.6pF$**,** $V_{dd} = 2.4$ **V,** $U_T = 0.026$

Property	Expression	I=100pA	I=1nA	I=10nA
Bandwidth (f)	$\frac{I}{2\pi U_T C}$	383Hz	3.83kHz	38.3kHz
Power (P)	$2IV_{dd}$	0.48nW	4.8nW	48nW
MMAC/μW	$\frac{10^{-12}}{9.6\pi U_T C}$	0.80	0.80	0.80

Table 9. Estimated Performance Parameters for VMM with Inputs Coming in Serially; with Log-amp Input stage using: $C = 1.6pF$**,** $V_{dd} = 2.4$ **V,** $U_T = 0.026$**,** $A = 165$

Property	Expression	I=100pA	I=1nA	I=10nA
Bandwidth (f)	$\frac{IA}{2\pi U_T C}$	63KHz	631kHz	6313kHz
Power (P)	$2IV_{dd}$	0.48nW	4.8nW	48nW
MMAC/μW	$\frac{10^{-12}A}{9.6\pi U_T C}$	132	132	132

see increased performance. According to [15], the gain stage of the Log-amp input stage configuration increases the bandwidth and therefore lowers the power-delay product, which increases computation per unit power. Estimated performance metrics for this system are shown in Table 9.

The power numbers for this analog system can be compared to the power numbers in Figure 2. If 1 MAC in 100pJ (10MMAC/mW) then one may make the following predictions for the computations per Joule for the buffered input stage and Log-amp input stage systems, (84), (85), respectively.

$$\frac{.8MMAC}{\mu W}\frac{10^6\mu W}{1W}\frac{1GMAC}{10^3MMAC} = \frac{800GMAC}{W}\frac{W}{J/S} \approx \frac{800GComputations}{J} \tag{84}$$

$$\frac{132MMAC}{\mu W}\frac{10^6\mu W}{1W}\frac{1GMAC}{10^3MMAC} = \frac{132000GMAC}{W}\frac{W}{J/S} \approx \frac{132000GComputations}{J} \tag{85}$$

6.2 Robotic Testbed Development

This section addresses the embedded systems developed for the visual processing robotic application. Two FPAA embedded systems are described. First, a modular board system built around the RASP 2.9V is described. Second, the latest RASP IC, the RASP 3.0 IC,

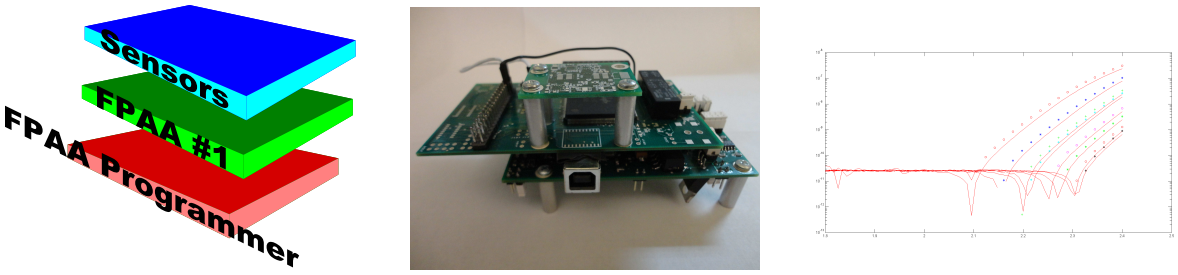
and its embedded system are described.

6.2.1 RASP 2.9V Modular Board System

A flying robot platform is a combination of three main items: the robot hardware, avionics, and sensors. This work's analog-digital hardware development is tied to the avionics and sensors packages. I developed a modular avionics system, Figure 91, in which different printed circuit board (PCB) modules have been made for specific tasks. These modules are integrated together as needed. Module (1) is a small FPAA board with a camera and motor driver IC, Module (2) is a microprocessor and power board for programming and powering the FPAA, and Module (3), designed by UC Berkeley [18], is a sensor and wireless transceiver module. This modular board system was developed for use with the RASP 2.9V FPAA IC. Further work on this system has been suspended in favor of a more powerful RASP 3.0 FPAA IC system.

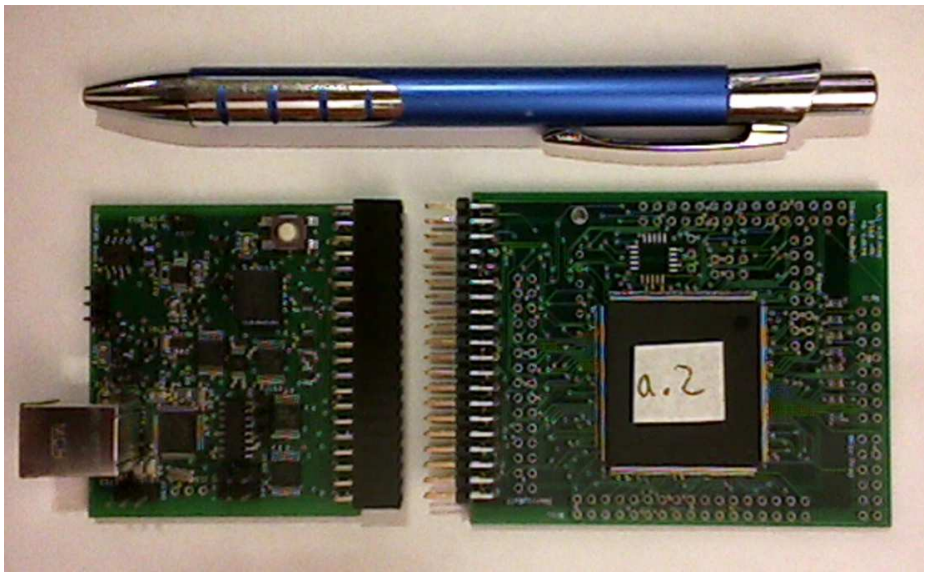
6.2.2 RASP 3.0 System

The latest FPAA developed by the CADSP Lab at Georgia Tech is the RASP 3.0. It is constructed in the 350nm CMOS process. The die measures 7mm by 12mm, Figure 93. When compared with previous iterations of the RASP IC, two distinguishing features of this IC are as follows. First, a Texas Instruments MSP430 compatible microprocessor has been



a) Board Stackup b) Side c) Measurements from System

Figure 91. a) Block diagram of Electronic Module stackup for avionics system. b) Side view of fabricated boards: stand alone FPAA programmer board (bottom), FPAA/camera board (middle), and UC Berkeley sensor board (top) [18]. c) Measured data from the FPAA programmer/Rasp 2.9V FPAA board stackup. Successful results from characterizing a source follower setup similar to Figure 88a for converting digital bits to current.



a) b)

Figure 92. a) RASP 3.0 Control Board (about 4 square inches). Note the USB connection on the top left; 40 pin header on the right to plug into the RASP 3.0 IC board. b) RASP 3.0a IC Board (about 7.5 square inches) contains the RASP 3.0a IC which is a smaller version of the RASP 3.0a.

integrated into the same die as the FPAA. Second, an analog memory has also been integrated into the die. These two features help allow the FPAA to process images efficiently. Having the microprocessor on the die means that the capacitive load associated with transmitting signals between the FPAA and microprocessor is reduced. The makes for a faster system. Having an analog memory on the die also allows the image processing system to save image information without the time cost of transferring data to the microprocessor's memory.

The RASP 3.0 Control Board, Figure 92a, is USB powered. An FTDI brand IC is used for serial communication. A high level system block diagram for this board and how it interacts with the RASP 3.0 is found in Figure 94.

6.2.2.1 Hardware Results

Some ititial results from characterizing a source follower setup used for the DAC in the RASP 2.9 system are found in Figure 91c. Each of these curves represents current contributions from each of eight transistors. The transistors have their gate voltages programmed

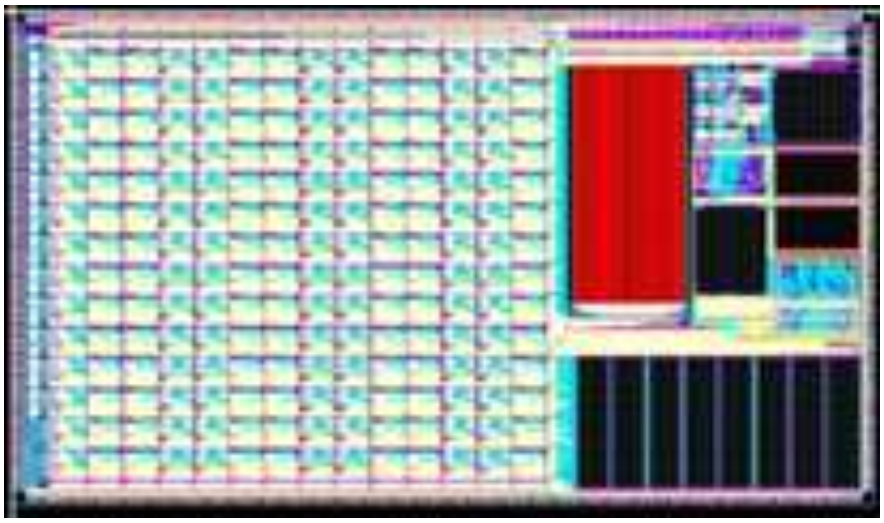


Figure 93. A picture of the RASP 3.0 IC layout.

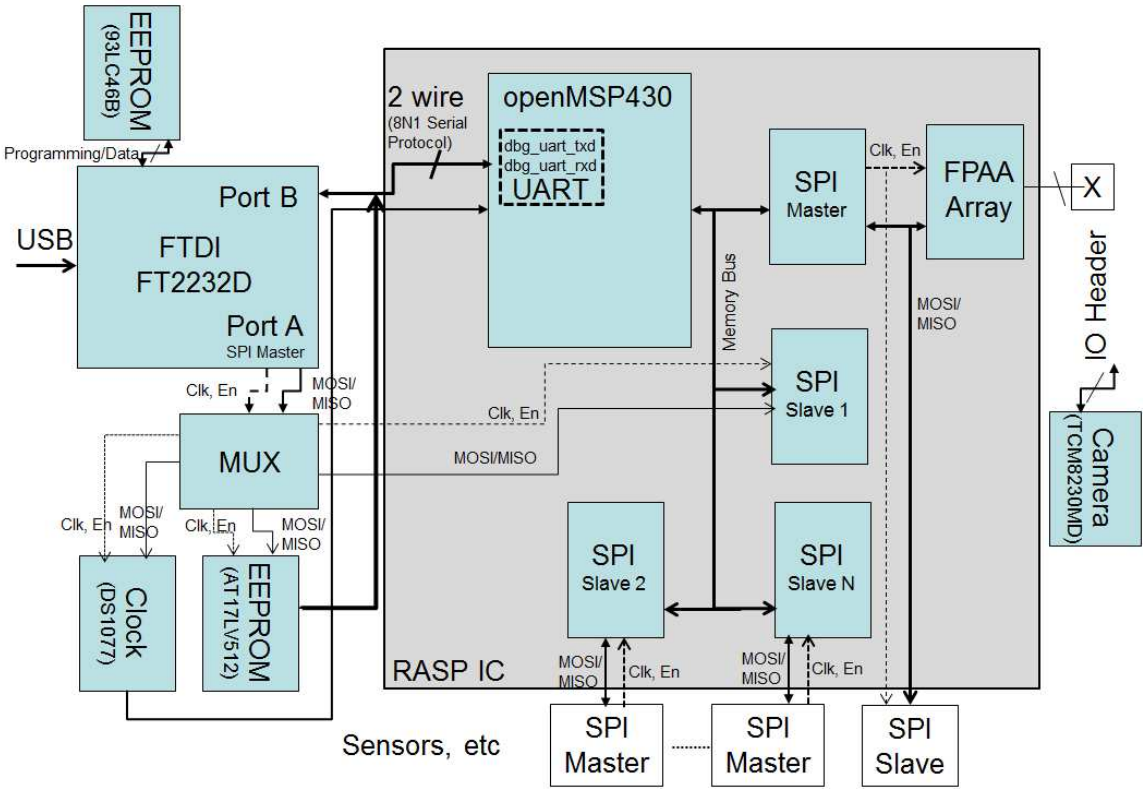


Figure 94. System block diagram for the interface between the RASP 3.0 control board and the RASP 3.0 IC.

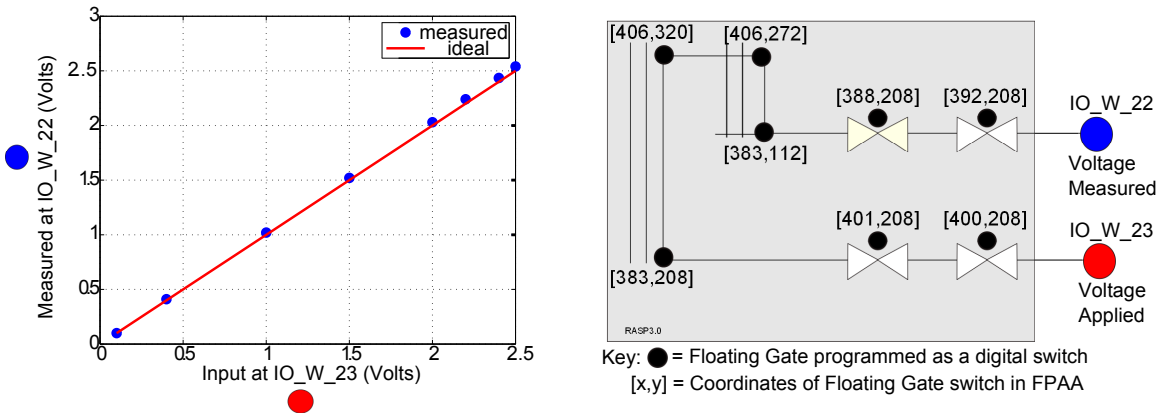
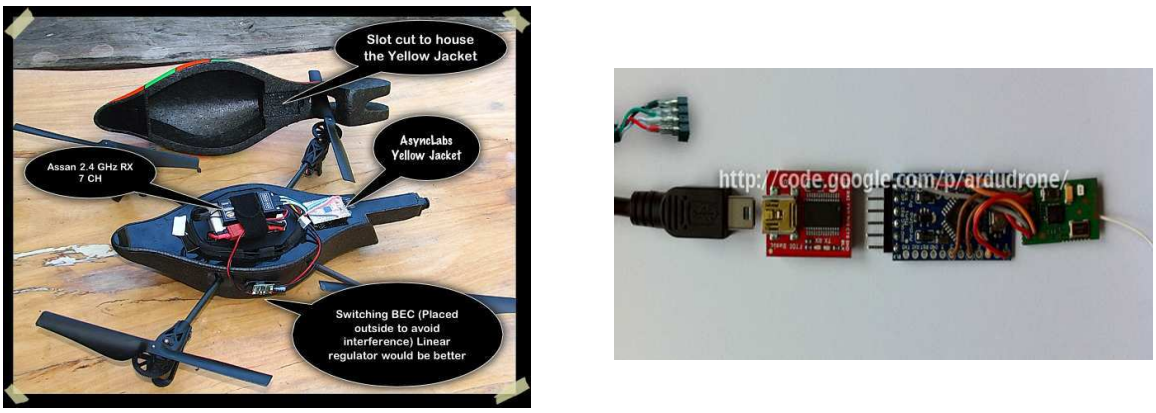


Figure 95. Measured Data from RASP 3.0. Loopback Switch test: connecting [I/O W 22] to [I/O W 23] using 8 digitally programed floating-gate switches. Input was applied at one I/O pin, passed through the FPAA, and measured on another I/O pin.

to binary weighted values. The full RASP 3.0 system will include routing software that allows one to design a system in Scicos and Xcos (two open-source Matlab and Simulink type programs, respectively.) The testing of the embedded system and RASP 3.0 IC is currently being done. We have been able to demonstrate a loop-back-switch test which involves applying an input on one I/O pin, passing the signal through the FPAA, and then back out on another I/O pin. The internal routing used 8 floating-gate switches to make the internal connection between I/O pins. We performed this experiment and the results are found in Figure 95. Figure 95a shows the measured results from hardware. The red line represents the ideal curve. If one applies a voltage A at the input, one ideally should measure voltage A at the output. The blue circles represent the measured data and show that our system is performing well. Figure 95b shows the coordinates of the floating-gate switches that were used to perform this test. The bow-tie looking blocks represent transmission-gate switches that are controlled by floating-gates. This simple test demonstrates basic functionality of the microprocessor and also our ability to program digital (i.e. fully conducting) floating-gate transistors.

6.2.2.2 Robot Integration

Future work involves integrating the FPAA board into a real robot. The ICE lab at Georgia Tech has recently purchased a Parrot AR.Drone quadrotor helicopter for this purpose,



a) b)

Figure 96. a) RC Controlled AR.Drone [19]. b) Ardudrone interface board which enables access into the AR.Drone control system [20].

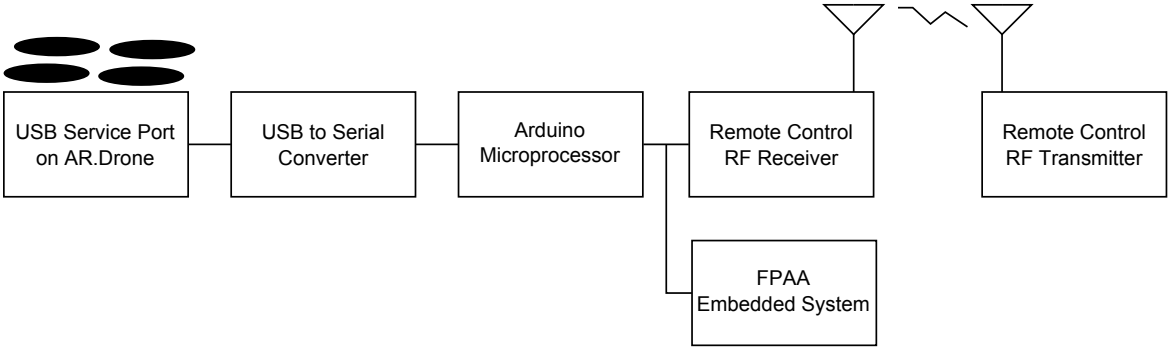


Figure 97. The FPAA could piggy-back onto the Ardudrone interface system to enable access into the AR.Drone control system [20]. The original Radio Control system of the Ardudrone could be maintained to use as a safety override system during testing.

Figure 96a. An open source embedded system called *Ardudrone* [20] has been identified as a possible resource for integrating the FPAA into the drone. The Ardudrone system interfaces to the AR.Drone through the drone's USB port. The Ardudrone system is designed to allow an RC controller (as used in RC model airplanes) to control the AR.Drone. An RF receiver on the drone receives the RF control signals, passes them to an Arduino for processing, and then these are passed to the drone via its USB port. Figure 97 shows how the FPAA Image processing system can be integrated into this existing control architecture. The RC control link could be maintained to use as a safety override during testing phases.

CHAPTER 7

CONCLUSIONS

This dissertation showed how to map path planning and image processing algorithms onto new analog signal processing systems based on floating-gate based reconfigurable analog hardware. Hardware results were shown, and benefits and limitations were described. Analog path planning with floating-gates and neurons was demonstrated in two chapters, and an analog computation based image processor was also described. One of the computational capability benefits of the path planning system is the Time Complexity win of the neuron based path planner, $\frac{d}{v}$, compared to $O\left(d^2\right)$ for a digital wavefront planner (where d is the depth of the solution and v is the propagation velocity of the wavefront in the neuron IC). One of the major benefits of the image processing system is the estimated performance of 800 to 132000 Giga Computations/Joule, which is significantly higher than the approximately 10 Giga Computations/Joule efficiency wall trend in Figure 2. One of the limitations shown is that the resistive grid solution is not always optimal.

7.1 Chapter Reviews

Chapter 2 presented three hardware and software infrastructures used with a family of floating-gate based FPAAs being developed at Georgia Tech. These compact and portable systems provide the user with a comprehensive set of tools for custom analog circuit design and implementation. The infrastructure includes the FPAA IC, microcontrollers for interfacing the FPAA with the user's computer, and Matlab and targeting software. The FPAA hardware can communicate with Matlab over a USB connection. When connected to a computer, the USB connection also provides the FPAA hardware's power. Some of the software tools include three major systems: a Matlab Simulink FPAA program, a SPICE to FPAA compiler called GRASPER, and a visualization tool called RAT. This chapter also

presented a description of a floating-gate transistor because this is the key enabling technology that allows the FPAA to program arbitrary circuits (and also implement resistive elements).

Chapter 3 presented path planning using resistive grids implemented on two different FPAA ICs: the RASP 2.8a and the RASP 2.9V [31, 73]. The resistive grids elements were implemented with floating-gate transistors. The general idea is similar to the well known potential field approach to path planning [74] in that the robot's location is the high point of an energy surface, the goal is at the low point, and the path to goal follows the gradient. This chapter presented hardware results using reconfigurable AVLSI circuits developed at Georgia Tech to implement a path planning algorithm. Experimental results were presented for a large number of environment scenarios. Also, an experimental result of interfacing the FPAA with a Pioneer robot was described.

Chapter 4 presented hardware results for a neuromorphic approach to path planning using a neuron array IC. The algorithm was explained, and experimental results were presented showing 100% correct and optimal performance for a large number of randomized maze environment scenarios. Based on neuron signal propagation speed, neuron IC path planning may offer a computational advantage over state-of-the-art wavefront planners implemented on FPGAs. Analytical Time and Space Complexity metrics were developed in this section for a Neuron IC planner, and these were verified against experimental data. Optimality and Completeness were also addressed. The neuron structure allows one to develop sophisticated graphs with varied edge weights between nodes of the grid. Two interesting cases were presented. First, asymmetric edge costs were assigned to describe cases which have a certain cost to travel a path in one direction, but a different cost to travel the same path but in the opposite direction. The application of this feature can translate to real world problems involving hills, traffic patterns, etc. Second, cases were presented where the nodes near an obstacle were given higher costs to visit those nodes. This is in

an effort to keep the autonomous agent at a safe distance from obstacles. This grid weighting can also be used to differentiate among terrains such as sand, ice, gravel, or smooth pavement. Experimental results were presented for both cases.

Chapter 5 presented results of a mobile manipulator robot tasked to play the classic Towers of Hanoi game. First, the control algorithms necessary to enable necessary game-playing behavior were discussed, and results were provided of implementing the methodology in a high fidelity 3D environment. After attaining success in the simulation environment, results were shown on implementation of the same control software using physical robot hardware. Additionally, analysis for implementing analog Proportional-Integral-Derivative (PID) control on this platform using a floating-gate based reconfigurable analog IC was explored. Using this concept of floating-gate analog arrays for control enables offloading of the processing, which could be helpful for real-time implementation of robot behavior.

Chapter 6 described a mixed signal image processing algorithm designed to filter and subsample an image. Two systems being developed for a hybrid analog-digital approach to image processing were also described. One of these systems is based on the RASP 2.9v IC and the second is based on the RASP 3.0 IC. Each has its own PCB embedded system which was also described.

7.2 Extending Analog Reconfigurable Circuits to Additional Autonomous System Problems

One extension of the planning research is to apply this work to 3-Dimensional (3D) grids. In theory, the bipartite grid algorithm is amenable to this added dimension. Figure 98 shows how a simple 3D grid is mapped onto the RASP 2.9V IC. The 3D planner may also be useful for cases such as a non-holonomic robot [129].

Autonomous Underwater Vehicles (AUVs) [189] are a power constrained robot platform, navigate with a 3D environment map (two space dimensions, one time dimension),

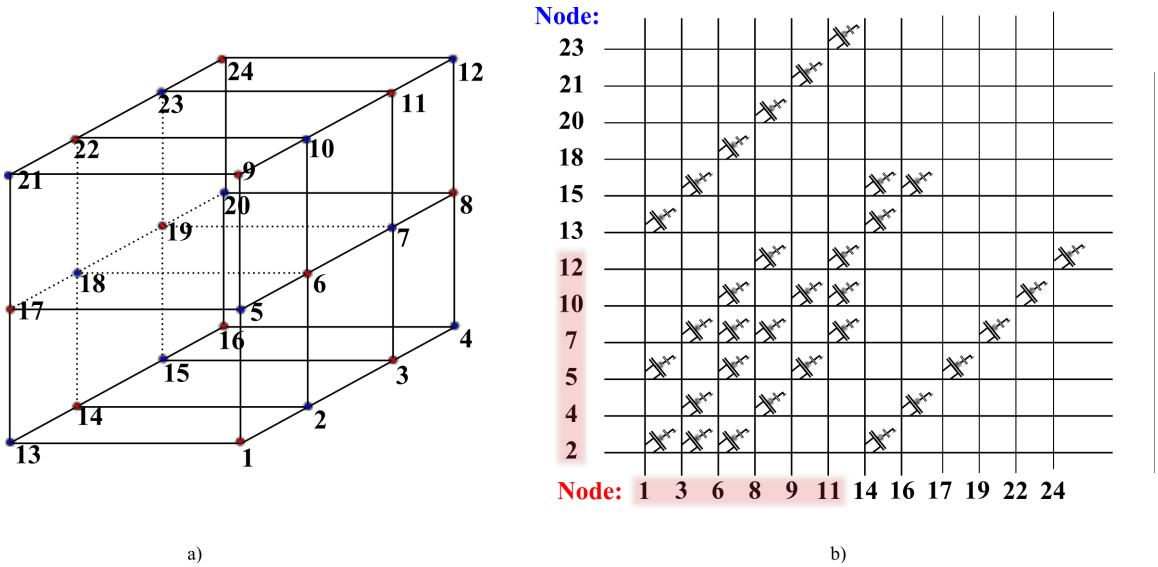


Figure 98. a) Three dimensional grid space. b) Mapping of Nodes 1-12 of the 3D grid in (a) onto a RASP 2.9V FPAA using a bipartite grid.

and have map updates on the order of minutes. These three qualities make AUV an excellent platform on which to apply the FPAA planner. One AUV application could be to use realistic 3D ocean data with the existing FPAA planner to solve a path planning task. Another 3D planning problem of interest is planning a path for an Unmanned Aerial Vehicle (UAV) in a three dimensional spacial environment. Regarding this 3D task, we may be able to draw upon work by Grupen, et al, in which 3D nonholonomic grids are explored [56], or work based on Fast Marching (FM) wavefronts [190, 191]. FM methods are especially applicable to aerial or under water environments because they allow one to take wind or current influences into account [190].

Another extension of the planning work is to choose a different environment representation. For instance, the neuron path planners might be used in a search space that has been decomposed into a road map method called a *visibility graph*, Figure 99. Visibility graphs are based on obstacle geometry [21]. If the resolution is high enough, these graphs will allow one to find a complete solution that will move the robot in the configuration space. A drawback to this representation is that the path will take the robot as close as possible to the objects.

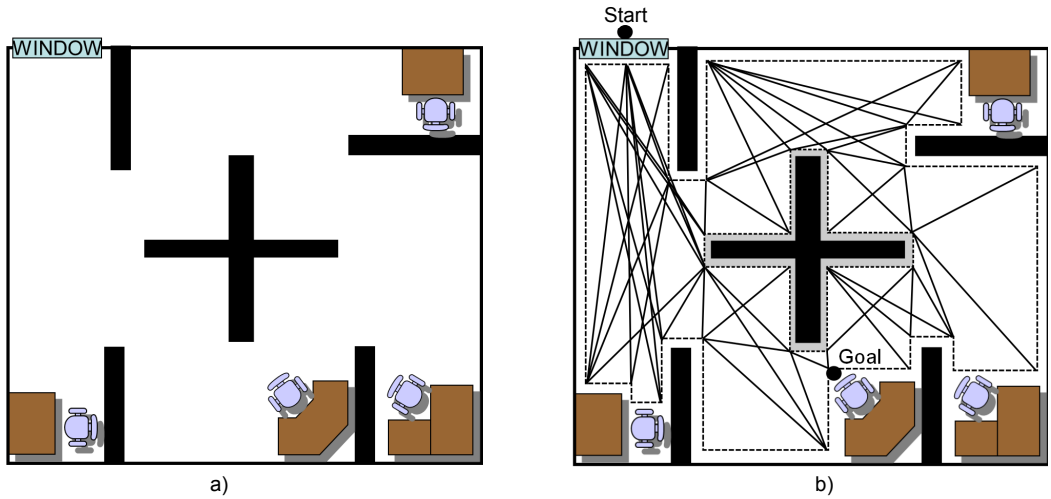


Figure 99. Representing the office environment as a Visibility graph. a) Office environment. b) Nodes of visibility graph are the start and end nodes and the corners of the obstacles, walls, and along boundaries [21].

7.3 HMMs, Dendrites, Diffusors, Analog ICs and Robotics

Finally, this dissertation concludes with some of my preliminary work which links multiple areas covered in my research: Hidden Markov Models (HMMs), Dendrites, Analog diffusors, and Robotics [192, 193]. An initial robotics application is to use HMMs to allow robots to recognize doors. Another application is to apply HMMs to robot state estimation. The link to my research is that I and present and past members of the Georgia Tech ICE Lab have been forging a link between the neural components called *dendrites* and HMMs and also working to implement these dendrite-diffusor-HMMs on FPAA platforms, [194, 195, 196, 197, 198, 192, 199, 193, 200]. Others have made the connection between analog integrated circuits and HMMs [201, 202]. Ideally, we will one day be able to use FPAA based dendrite-diffusor-HMMs to perform robotic object classification such as door finding or robot state estimation. In some preliminary work, a laser ranging sensor was mounted on a robot and used to detect the presence or absence of doorways. In this system, the sensor makes a single 180 degree row scan in front of the robot. This row scan data is the input to the HMM classifier. The HMM classifier under development is currently a digital Matlab solution [203], but the desire is to use this as a basis for implementing it on

the FPAA as an analog HMM-dendrite classifier.

The Hidden Markov Model (HMM) door finder is inspired by previous work using HMMs to characterize numbers [128]. Another paper using HMMs for door finding is [204]. In Bishop's example, numbers are characterized on-line as a pencil traces out a number though time. The number 2, for example, can be made in many ways. An individual may start at the top left and finish at the bottom right, or they may start at the bottom right and finish at the top left. In this type of situation the starting point and direction of pencil travel matters. This work uses HMMs to "trace along" the laser data and identify sections of the curve that are consistent with a trained "door model." Since doors are typically symmetrical one may assume that the starting point and direction of trace along the data is not as important as it is in number or character recognition problems.

7.4 Converting Discrete HMMs to Continuous HMMs

This section describes a method of converting an HMM classifier that has discrete states and discrete times into one that has a continuum of states in continuous time. This process is important because analog circuit systems are inherently continuous time systems. This discrete to continuous conversion is accomplished by using variable substitutions and Taylor series approximations. First, discrete HMMs are reviewed, and this is followed by a discussion on the continuous time version.

7.4.1 Discrete Hidden Markov Models

HMMs are used as pattern classifiers, i.e. discriminators, and have been applied to a wide range of problems such as speech recognition, hand and face recognition, gesture recognition, and robot guidance [205, 206, 207]. An HMM is a state machine. A unique HMM state machine is created (or better "learned") for each item to be discriminated. The input to an HMM system is a time series of observations. These observations can take many forms. For isolated word recognition or sentence recognition these may be phonemes. For gesture recognition, the observations may be the temporal component of the hand as it moves

[206]. The output of an HMM is a number: the likelihood that a sequence of observations came from a certain HMM model. The output of an HMM classifier is to discriminate the observation among various models.

Each HMM model is composed of three parameters: the probability of each state being the starting state, the probability of each state producing certain observations, and the probability of transitions among states. Nominally, for a discrete time HMM Model, each state can produce an observed output with some probability at each time step. The sequence of transitions between states is not known a priori, but is characterized by a set of probabilities. The reason this is called a "Hidden" Markov model is because an observer to the system only sees outputs but does not know a priori which states produced the observations and when each state was visited.

Rabiner does a great job of describing three basic problems that must be solved in an HMM classifier [25, 208]. The wave propagating PDE (described later and in [198]) was posed as a solution to Rabiner's "Problem 1" statement:

Given an observation sequence $\mathbf{O} = [\mathbf{o_1}, \mathbf{o_2}, \cdots, \mathbf{o_T}]$ *and an HMM model* $\lambda_Y = (\mathbf{A}_Y, \mathbf{b}_Y, \pi_Y)$ *how does one compute the probability of the observation sequence given the model:* $P[\mathbf{O}|\lambda]$*?*

Where $o(n)$ are continuous observations (inputs to the HMM classifier), $\mathbf{A}_Y$ is the state transition matrix, $\mathbf{b}_Y$ are continuous symbol observation densities, and π_Y is the initial state probability vector. The subscripts Y indicate that these matrices are for a specific HMM model.

The italicized "Problem 1" statement above represents the *a posteriori* probability that an observed sequence came from a certain model Y. This is very computationally expensive to calculate directly as it requires $2TN^T - 1$ operations where N is the number of states and T is the length of the observations [208]. It can, however, be calculated more efficiently using (86) [25]. The discrete observations (such as phonemes) from time step n=1 to time step n=T are $[o_1, o_2, \ldots, o_T]$ [25]:

$$P\left[\left[o_1, o_2, \ldots, o_T\right] | \lambda_Y\right] = \sum_{i=1}^{\mathrm{N}} \varphi_i\left(T\right) \tag{86}$$

Where variable φ is the recursion variable, (87). (φ is called α in the *forward-backward* procedure in previous publications [25, 209, 205]) Variable i is the current state and variable j is the next state.

$$\varphi_j(n+1) = \left[\sum_{i=1}^{\mathrm{N}} \varphi_i(n) a_{ij}\right] b_j(o_{n+1}) \tag{87}$$

Where the range of the time step is:

$$1 < n \leq T - 1 \tag{88}$$

And the index over states is:

$$1 \leq j \leq N \tag{89}$$

Assuming a left-right state model, it is possible to estimate (86) by (90) where N is the number of states:

$$P\left[\left(o_1, o_2, \ldots, o_T\right) | \lambda_Y\right] \approx \varphi_N\left(T\right) \tag{90}$$

For $n > 1$, and assuming a left-right model where a state can only transition either to the next state or back to itself (Figure 104c), the recursion variable, φ, in (87) simplifies to (93).

$$\varphi_j\left(n\right) = b_j\left(o_n\right)\left[\varphi_j\left(n-1\right) a_{jj} + \varphi_{j-1}\left(n-1\right) a_{j-1,j}\right] \tag{91}$$

Redefine the state transition variable:

$$a_{j-1} \equiv a_{j-1,j} \tag{92}$$

and substituting (92) into (91) gives the equation that is stated in previous papers [201, 202, 198]:

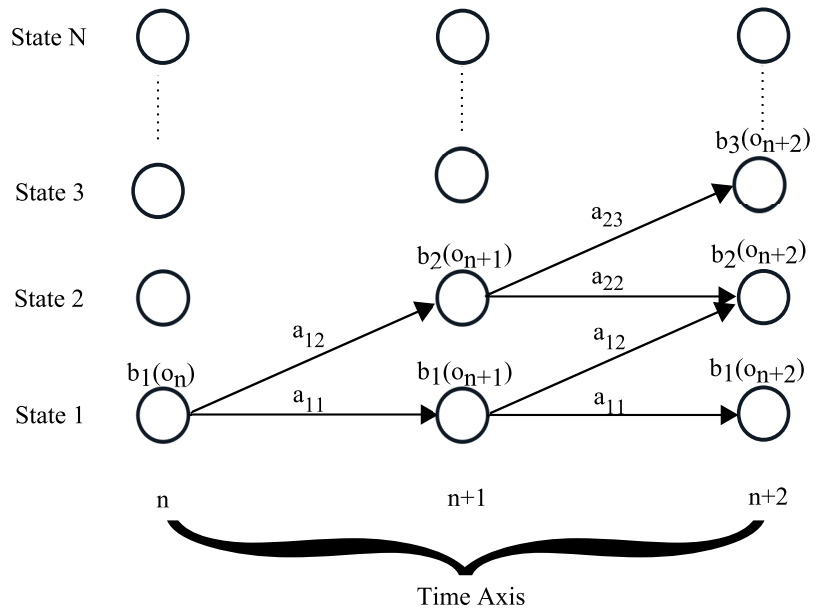


Figure 100. Visualizing the recursion for the discrete HMM using a trellis diagram. The observation sequence in this example is $[o_1, o_2, o_3, o_4]$, where a_{12} represents state transition probability from state 1 to state 2, and $b_1(o_3)$ represents the probability of getting observation o_3 at n = 3 while in state 1.

$$\varphi_j(n) = b_j(o_n)\left[\varphi_j(n-1)\left(1-a_j\right) + \varphi_{j-1}(n-1)\,a_{j-1}\right] \tag{93}$$

where a_j and b_j are elements of the state transition matrix $\mathbf{A}_Y$ and the symbol probability matrix $\mathbf{B}_Y$ respectively. Equation 94 initializes the recursion:

$$\varphi_j(1) = \pi_j b_j(o_1) \tag{94}$$

And the index over states is:

$$1 \le j \le N \tag{95}$$

The recursive process of calculating the *a posteriori* probability (93) can be visualized as a trellis, Figure 100.

7.4.2 Continuous Hidden Markov Models

The HMM is typically thought of as a discrete time model, but it can be expressed as a function of continuous time and states [210]. One may restate the recursion variable in

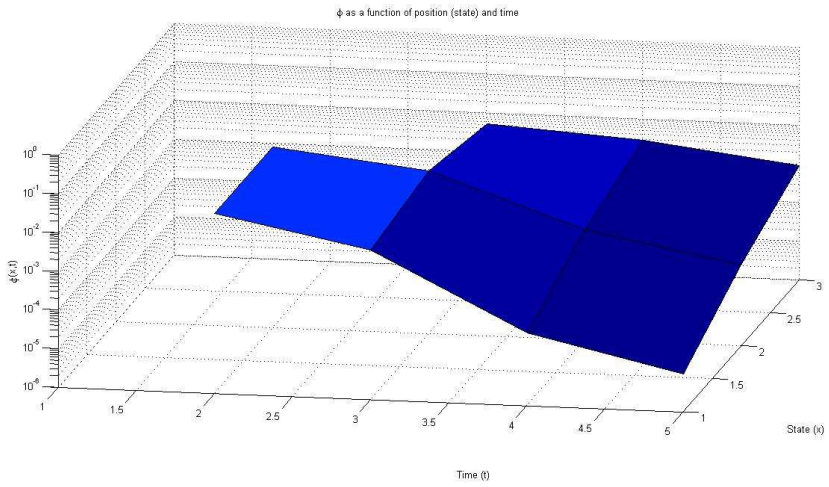


Figure 101. A three dimensional surface representing the recursion variable φ in the continuous HMM as a function of continuous time and state.

(93) as a function of time by replacing the discrete variable n with a continuous variable t and discrete variable $n-1$ with a continuous variable $t-\tau$:

$$\varphi_j(t) = b_j(o_t)\left[\left(1-a_j\right)\varphi_j(t-\tau) + a_{j-1}\varphi_{j-1}(t-\tau)\right] \tag{96}$$

The time for a specific state is now continuous in (96), but the states and probabilities are still discrete. One may restate the discrete states j in (96) as a continuous function of position x as in (97) and (98).

$$\varphi_j(t) \equiv \varphi(x,t) \tag{97}$$

$$\varphi_{j-1}(t) \equiv \varphi(x-\Delta,t) \tag{98}$$

The value of φ in (97) can be seen as a point on a three dimensional surface (Figure 101). The symbol and state transition probabilities, b and a respectively in (96), are currently expressed as a function of a discrete state but can be expressed in terms of the continuous state. The continuous representation of the state transition probabilities, a, is shown in (99).

$$\begin{aligned} a_j &\equiv a(x) \\ a_{j-1} &\equiv a(x-\Delta) \end{aligned} \tag{99}$$

The continuous representation of the symbol observation probability, b, is shown in (100).

$$b_i(o_t) \equiv b(x, o_t) \tag{100}$$

The *a posteriori* probability in (86) may now be expressed as a continuous function of time and state, (101).

$$P\left[\underset{0<t\leq T}{o_t} |\lambda_Y\right] = \int_{x=0}^{N*\Delta} \varphi(x,T)dx \tag{101}$$

The approximation of (90) can also be restated as a continuous function of time and state:

$$P\left[\underset{0<t\leq T}{o_t} |\lambda_Y\right] \approx \varphi(N*\Delta, T) \tag{102}$$

The recursive expression for calculating φ in the discrete HMM can now be expressed as a function of continuous time and state by substituting (99) and (100) into (96). This is shown in (103):

$$\varphi(x,t) = b(x,o_t)\begin{bmatrix} (1-a(x))\varphi(x,t-\tau) \\ +a(x-\Delta)\varphi(x-\Delta,t-\tau) \end{bmatrix} \tag{103}$$

A state's φ value at a previous time can be related to the state's current time using the Taylor series approximation [198]:

$$\varphi(x,t-\tau) \approx \varphi(x,t) - \tau\frac{\partial\varphi(x,t)}{\partial t} \tag{104}$$

One state's φ value at time t may be related to another state's value at time t using the Taylor series approximation [198]:

$$\varphi(x-\Delta,t) \approx \varphi(x,t) - \Delta\frac{\partial\varphi(x,t)}{\partial x} \tag{105}$$

Since the shape of φ is a surface as in Figure 101, one can combine the ideas of (104) and (105) and relate a state at one time to a previous state at a different time by:

$$\varphi(x-\Delta, t-\tau) \approx \varphi(x,t) - \Delta\frac{\partial\varphi(x,t)}{\partial x} - \tau\frac{\partial\varphi(x,t)}{\partial t} \tag{106}$$

Substituting (104) and (106) into (103) and simplifying yields the general differential equation for a left to right model continuous time and state HMM, (107).

$$\begin{bmatrix} \left[\frac{1}{b(x,t)} - 1 - a(x-\Delta) + a(x)\right]\varphi(x,t) \\ +\left[1 + a(x-\Delta) - a(x)\right]\tau\frac{\partial\varphi(x,t)}{\partial t} \\ +a(x-\Delta)\,\Delta\frac{\partial\varphi(x,t)}{\partial x} \end{bmatrix} = 0 \tag{107}$$

Re-arranging terms:

$$\begin{bmatrix} \tau\frac{\partial\varphi(x,t)}{\partial t} \\ +\frac{\frac{1}{b(x,t)}-1-a(x-\Delta)+a(x)}{1+a(x-\Delta)-a(x)}\varphi(x,t) \\ +\frac{a(x-\Delta)}{1+a(x-\Delta)-a(x)}\Delta\frac{\partial\varphi(x,t)}{\partial x} \end{bmatrix} = 0 \tag{108}$$

Further re-arranging terms:

$$\begin{bmatrix} \tau\frac{\partial\varphi(x,t)}{\partial t} \\ +\left(\frac{\frac{1}{b(x,t)}}{1+a(x-\Delta)-a(x)} - 1\right)\varphi(x,t) \\ +\frac{a(x-\Delta)}{1+a(x-\Delta)-a(x)}\Delta\frac{\partial\varphi(x,t)}{\partial x} \end{bmatrix} = 0 \tag{109}$$

If one assumes that the state transition probabilities are equal, (110), then the differential equation in (108) simplifies to (111) [198].

$$a(x-\Delta) \equiv a(x) \tag{110}$$

The final expression for the continuous HMM can be seen as a wave propagating PDE [198].

$$\underbrace{\tau\frac{\partial\varphi(x,t)}{\partial t}}_{\substack{\text{state}\\\text{element}}} + \underbrace{\left(\frac{1}{b(x,t)} - 1\right)\varphi(x,t)}_{\substack{\text{decay}\\\text{term}}} + \underbrace{a(x)\,\Delta\frac{\partial\varphi(x,t)}{\partial x}}_{\substack{\text{wave}\\\text{propagation}}} = 0 \tag{111}$$

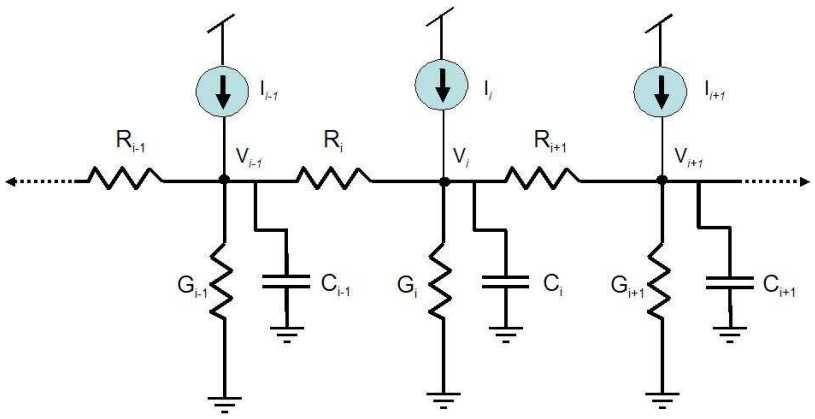


Figure 102. RC Delay line model.

7.5 Diffusors Implementing HMM Computation

It can be shown that the wave propagating PDE from Section 7.4.2 is similar to the differential equation used to describe diffusor circuits. Two types of diffusors include an RC delay line and also a delay line with transistors in place of resistors [194].

7.5.1 RC Delay Line Diffusor

The classical RC delay line is reviewed in Mead's text [186]. Figure 102 shows the topology. Kirchhoff's Current Law (KCL) can be used to derive a differential equation for this circuit, (112) where G is conductance.

$$I_i(t) = C_i \frac{dV_i(t)}{dt} + V_i(t) G_i + \frac{[V_i(t) - V_{i+1}(t)]}{R_{i-1}} + \frac{[V_i(t) - V_{i-1}(t)]}{R_i} \tag{112}$$

Assuming the horizontal resistances are equal as in (113) allows one to simplify (112) to (114):

$$R_i = R_{i-1} = R_x \tag{113}$$

$$I_i(t) = C_i \frac{dV_i(t)}{dt} + V_i(t) G_i + \frac{1}{R_x} [2V_i(t) - V_{i+1}(t) - V_{i-1}(t)] \tag{114}$$

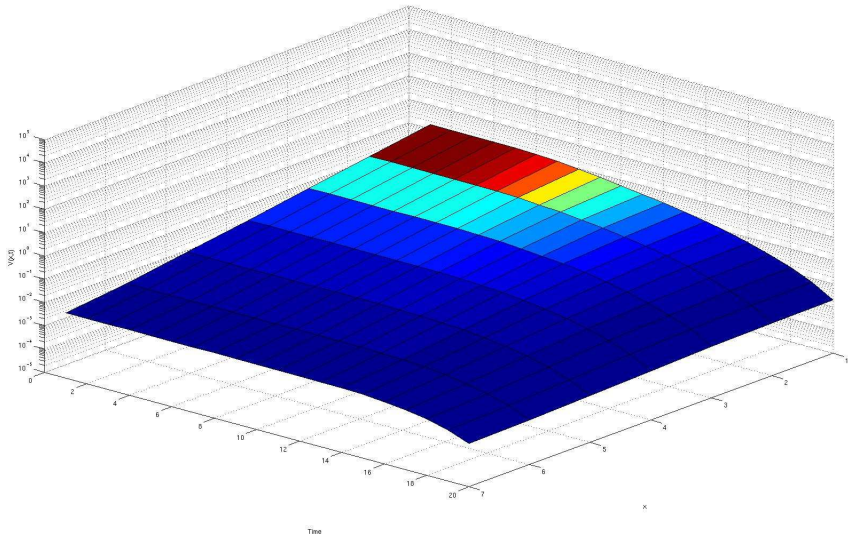


Figure 103. Voltages for a 7 tap RC Delay Line (R = .9 , G = .9, C= .01).

Assuming there are many nodes allows one to perform the following change of notation from discrete nodes to continuous nodes:

$$V_i(t) = V(x,t) \tag{115}$$

$$V_{i+1}(t) = V(x+\Delta_x, t) \tag{116}$$

$$V_{i-1}(t) = V(x-\Delta_x, t) \tag{117}$$

The value of V in (115) can be seen as a point on a three dimensional surface, Figure 103. Assuming that Δ_x represents something like a “position delta” one may use the Taylor series to describe the continuous nodes in terms of Δ_x, (118), (119).

$$V(x+\Delta_x, t) = V(x,t) + \Delta_x \frac{dV(x,t)}{dx} + \frac{1}{2}(\Delta_x)^2 \frac{d^2V(x,t)}{dx^2} + \cdots \tag{118}$$

$$V(x-\Delta_x, t) = V(x,t) - \Delta_x \frac{dV(x,t)}{dx} + \frac{1}{2}(\Delta_x)^2 \frac{d^2V(x,t)}{dx^2} + \cdots \tag{119}$$

The bracketed term in (114) can be re-written in continuous node terms:

$$\begin{aligned} &2V_i(t) - V_{i+1}(t) - V_{i-1}(t) = \\ &2V(x,t) - [V(x+\Delta_x,t) + V(x-\Delta_x,t)] \end{aligned} \tag{120}$$

Substitute (118) and (119) into (120) and simplify:

$$2V_i(t) - V_{i+1}(t) - V_{i-1}(t) \approx -(\Delta_x)^2 \frac{d^2V(x,t)}{dx^2} \tag{121}$$

Substituting (121) into (114) and simplifying yields (122), the generalized PDE describing the RC delay line diffusor.

$$I_i(t) R_x = R_x C_i \frac{dV_i(t)}{dt} + R_x G_i V_i(t) - (\Delta_x)^2 \frac{d^2V(x,t)}{dx^2} \tag{122}$$

If one assumes no input current at the top of each node $I_i = 0$, then one can put the diffusor circuit into a form similar to the continuous time HMM equation, (123).

$$\underbrace{R_x C_i \frac{dV(x,t)}{dt}}_{\substack{state \\ element}} + \underbrace{R_x G_i V(x,t)}_{\substack{decay \\ term}} - \underbrace{(\Delta_x)^2 \frac{d^2V(x,t)}{dx^2}}_{\substack{diffusion \\ term}} = 0 \tag{123}$$

7.5.1.1 Comparing the HMM PDE to the RC Delay Line PDE

Table 10 compares terms from the HMM PDE and the RC delay line differential equation.

Table 10. Comparing HMM PDE and RC Delay Line Terms

Element Description	HMM PDE	RC Delay Line
Recursion Variable	$\varphi(x,t)$	$V(x,t)$
State Element Coefficient	τ	$R_x C_i$
Decay Term Coefficient	$\frac{1}{b(x,t)} - 1$	$R_x G_i$
Wave Propagation/Diffusion Term	$a(x)\Delta\frac{\partial\varphi(x,t)}{\partial x}$	$-(\Delta_x)^2 \frac{d^2V(x,t)}{dx^2}$

If one assumes that the resistances are NOT equal and that the conductance of the line increases towards the right:

$$R_{i-1} \gg R_i \tag{124}$$

One can re-write (112) as:

$$I_i(t) = C_i \frac{dV_i(t)}{dt} + V_i(t)\left[G_i + \frac{1}{R_{i-1}} + \frac{1}{R_i}\right] - \left[\frac{V_{i+1}(t)}{R_{i-1}} + \frac{V_{i-1}(t)}{R_i}\right] \tag{125}$$

Which, using the resistance assumption, simplifies to:

$$I_i(t) = C_i \frac{dV_i(t)}{dt} + V_i(t)\left[G_i + \frac{1}{R_i}\right] - \frac{V_{i-1}(t)}{R_i} \tag{126}$$

Substituting the Taylor series expansions of (115) and (117) into the above:

$$I_i(t) = C_i \frac{dV(x,t)}{dt} + V(x,t)\left[G_i + \frac{1}{R_i}\right] - \frac{1}{R_i} V(x - \Delta_x, t) \tag{127}$$

$$I_i(t) = C_i \frac{dV(x,t)}{dt} + V(x,t)\left[G_i + \frac{1}{R_i}\right] - \frac{1}{R_i}\begin{bmatrix} V(x,t) \\ -\Delta_x \frac{dV(x,t)}{dx} \\ +\frac{1}{2}(\Delta_x)^2 \frac{d^2V(x,t)}{dx^2} \end{bmatrix} \tag{128}$$

Assuming that:

$$\Delta x \ll 1 \tag{129}$$

Then one can neglect higher order terms of the Taylor series:

$$(\Delta_x)^2 \approx 0 \tag{130}$$

$$I_i(t) = C_i \frac{dV(x,t)}{dt} + V(x,t)\left[G_i + \frac{1}{R_i}\right] - \frac{1}{R_i}\left[V(x,t) - \Delta_x \frac{dV(x,t)}{dx}\right] \tag{131}$$

Re-arranging terms:

$$\begin{aligned} I_i(t) = C_i \tfrac{dV(x,t)}{dt} + V(x,t)\left[G_i + \tfrac{1}{R_i} - \tfrac{1}{R_i}\right] \\ + \tfrac{1}{R_i}\Delta_x \tfrac{dV(x,t)}{dx} \end{aligned} \tag{132}$$

$$I_i(t) R_i = R_i C_i \frac{dV(x,t)}{dt} + V(x,t) G_i R_i + \Delta_x \frac{dV(x,t)}{dx} \tag{133}$$

$$0 = \underbrace{R_i C_i \frac{dV(x,t)}{dt}}_{\substack{state \\ element}} + \underbrace{V(x,t) G_i R_i - I_i(t) R_i}_{\substack{decay \\ term}} + \underbrace{\Delta_x \frac{dV(x,t)}{dx}}_{\substack{wave \\ propagation}} \tag{134}$$

$$0 = \underbrace{R_i C_i \frac{dV(x,t)}{dt}}_{\substack{state \\ element}} + \underbrace{V(x,t)[G_i R_i - 1]}_{\substack{decay \\ term}} + \underbrace{\Delta_x \frac{dV(x,t)}{dx}}_{\substack{wave \\ propagation}} \tag{135}$$

Assuming that HMM will always propagate to the next state and there is no probability that it will remain in its current state leads to the assumption in (136):

$$a(x) = 1 \tag{136}$$

Further assuming that the delta's are the same in both the HMM and RC-delay line, $\Delta = \Delta_x = K$, and updating Table 10 yields Table 11.

Table 11. Comparing HMM PDE and RC Delay Line Terms w/Assumptions

Element Description	HMM PDE	RC Delay Line
Recursion Variable	$\varphi(x,t)$	$V(x,t)$
State Element Coefficient	τ	$R_i C_i$
Decay Term Coefficient	$\frac{1}{b(x,t)} - 1$	$G_i R_i - 1$
Wave Propagation/Diffusion Term	$K \frac{\partial \varphi(x,t)}{\partial x}$	$K \frac{dV(x,t)}{dx}$

7.6 HMMs and Analog Systems for Robot Navigation

Previous works have made connections among HMMs and dendrites, and transistor models of dendrites [192, 198, 197, 196, 195, 194, 193]. This section first describes navigation systems and an important estimation tool used for navigation called the Kalman filter. A close relationship is also described between the equations for HMMs and Kalman filters. It is natural then to consider that there may also be a connection between dendrite computation and state estimation computation in Kalman filters, Figure 104a. It has also been shown that Kalman filters can be represented using electrical networks, [211], and electrical networks in Factor Graph format, [212, 213].

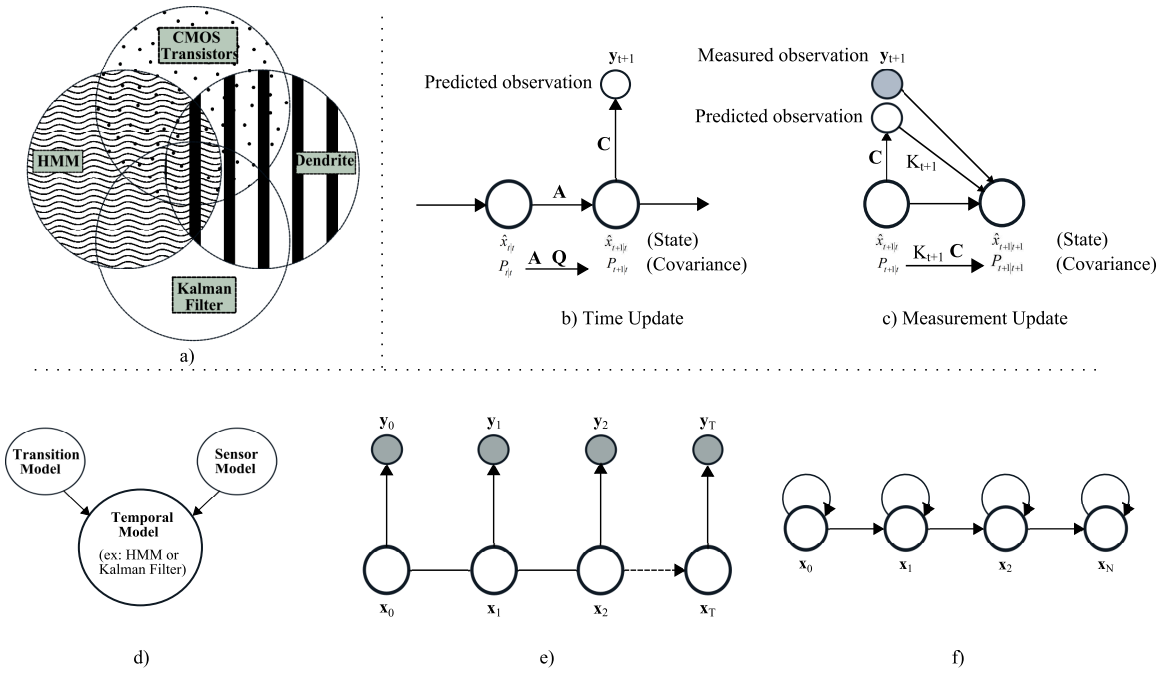
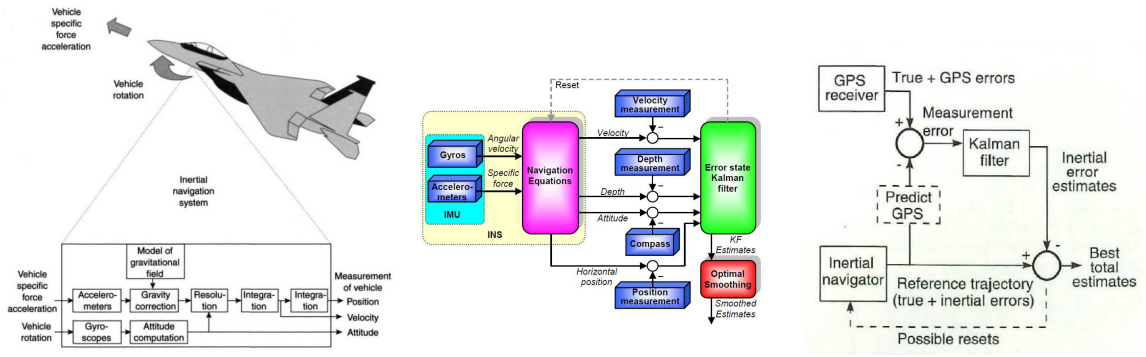


Figure 104. a) Future research seeks to make a connection among HMMs, Kalman filters, Dendrites and CMOS transistors. There are two main recursive steps in the Kalman filter [22] b) Time update step which estimates first the covariance and expected state. After calculating the expected state, one may predict the expected measurement. c) Measurement update step first calculates the Kalman gain matrix using, among other things, the expected covariance from step (b). This gain matrix is then used to compute an update to the Covariance matrix and also and update to the state. d) A Temporal Model is useful for modeling Navigation. The HMM and Kalman Filter are two such types of Temporal Models. The inputs to these temporal models are transition models and sensor models [1]. e) Probabilistic Independence Network for HMM (grey circles are observed) [23, 22, 24]. f) Left-Right HMM model [25].

The goal of a navigation system, is to give the robot or autonomous agent an estimate of its position in the world and often its orientation, velocity, and acceleration. This is often called estimating the *state* of the agent. Navigation can be accomplished in many ways such as using celestial bodies, radio waves, radar, satellites, or dead reckoning (DR) [214]. Inertial navigation is a type of DR and is the focus of this work. An *inertial* reference frame has its origin at the center of the Earth and axes which are non-rotating with respect to the fixed stars [26].

One of the problems in navigation systems is that the robot can not measure all of the elements of its state. Instead, in inertial navigation the robot's position and velocity are assumed to be a linear function of quantities that the robot *can* measure, like acceleration and angular velocity. Inertial Measurement Units (IMUs) are devices that typically contain three orthogonal rate-gyroscopes and three orthogonal accelerometers to measure angular velocities and acceleration [215]. This sensor data is then integrated to yield velocity and position information. Integrating the measured acceleration gives velocity, and a second integration yields position, Figure 105a. "*To integrate in the correct direction, attitude is needed. This is obtained by integrating the sensed angular velocity... Equations integrating the gyro and accelerometer measurements into velocity, position and orientation are called navigation equations [216].*" [26] An Inertial Navigation System (INS) combines an IMU with a computer, Figure 105b. The computer performs the navigation equation calculations and produces the state information [216, 27].

To further complicate the navigation problem, the measurements are assumed to be noisy. The estimated state will therefore be a random variable with some degree of confidence that is often expressed in a covariance matrix. Kalman filtering is a common digital based method for dealing with the state estimation task of mobile robots in the presence of Gaussian noise. This filtering algorithm is also commonly used for guided missiles [217], radar tracking systems [218], etc. The Kalman filter is optimal in the sense that the "expected value of the square of the error magnitude is minimized [219]." In our robot context,



a) b) c)

Figure 105. INS figures a) from [26]. b) Aided Inertial Navigation System (AIS) from [27]. c) Using a Kalman filter to combine the sensor data from an INS and GPS [28].

the Kalman filter is a method of combining noisy sensor measurements from sensors such as gyroscopes, accelerometers, magnetometers, GPS (or a surrogate system), airspeed sensors, etc. to determine the state of the robot [220]. For instance, "Integrating GPS with an inertial navigation system (INS) and a Kalman filter provides improved overall navigation performance. Essentially, the INS supplies virtually noiseless outputs that slowly drift off with time. GPS has minimal drift but much more noise. The Kalman filter, using statistical models of both systems, can take advantage of their different error characteristics to optimally minimize their deleterious traits [28]." [221] In another, more self contained non-GSP dependent solution, it is possible to use a Kalman filter to combine gyroscope data with accelerometer and magnetometer data for an improved navigation solution [222, 223]. In this system, the accelerometer and magnetometer data are used to compensate for the drift of Micro Electro-Mechanical (MEMs) based rate gyroscopes [222, 224, 225]. In these types of systems the Kalman filter often operates on signals which represent the attitude errors, [226].

This process occurs over time and seeks to reduce the estimation error by combining the measurements. "In Kalman filtering we wish to estimate samples of an unobserved process x, given samples of some observed process y and a (statespace) dynamic stochastic model for processes x and y [227]." In our case, the unobserved process x could be the robot

position and velocity, and the observed process *y* is the sensor data. "The Kalman filter not only works well in practice, but it is theoretically attractive because it can be shown that of all possible filters, it is the one that minimizes the variance of the estimation error [228]."

The Kalman filter is a statistical estimation tool. Previous research has established that there is a connection between Kalman filters and HMMs [229]. "Now, it is well known (see e.g. [230, 231]) that the statespace model which underlies the Kalman filter is indeed an HMM (with continuous state process) [227]," or, put another way, "The Kalman-filter model is an HMM with linear Gaussian model densities [232]." In [232], the authors provide "a comprehensive framework in which linear Kalman-filter models are subsumed by HMMs."

A temporal probability model, Figure 104d can be used to describe the navigation problem. These types of models have two sub-models: A *transition model* and a *sensor model* [1]. Hidden Markov Models (HMM) and Kalman Filters are special cases of a temporal model called a *Dynamic Bayesian Network* (DBN) [1]. Figure 104e shows a Probabilistic Independence Network (PIN) representation of an HMM, and Figure 104e shows a left-right representation of an HMM.

Future work is to continue to make the link between dendritic processing and autonomous vehicle navigation with the goal of implementing the algorithm on a reconfigurable platform such as an FPAA.

REFERENCES

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[2] S. Koziol, P. Hasler, and M. Stilman, "Robot Path Planning Using Field Programmable Analog Arrays," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 1747–1752.

[3] B. Marr, B. Degnan, P. Hasler, and D. Anderson, "Scaling Energy Per Operation via an Asynchronous Pipeline," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 1, pp. 147–151, 2013.

[4] J. P. Uyemura, *Introduction to VLSI Circuits and Systems*. J. Wiley, 2002.

[5] P. Hasler, "NSF Proposal," *Georgia Institute of Technology-NSF*, 2011.

[6] C. Petre, C. Schlottmann, and P. Hasler, "Automated Conversion of Simulink Designs to Analog Hardware on an FPAA," in *Circuits and Systems. IEEE International Symposium on*, May 2008, pp. 500–503.

[7] S. Koziol and P. Hasler, "Reconfigurable Analog VLSI Circuits for Robot Path Planning," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, June 2011, pp. 36 –43.

[8] S. Koziol, D. Lenz, S. Hilsenbeck, S. Chopra, P. Hasler, and A. Howard, "Using Floating-Gate Based Programmable Analog Arrays for Real-Time Control of a Game-Playing Robot," in *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, 2011, pp. 3566–3571.

[9] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue," *Robotics Automation Magazine, IEEE*, vol. 19, no. 3, pp. 46 –56, Sept. 2012.

[10] S. Brink, S. Nease, P. Hasler, S. Ramakrishnan, R. Wunderlich, A. Basu, and B. Degnan, "A Learning-Enabled Neuron Array IC Based Upon Transistor Channel Models of Biological Phenomena," *Biomedical Circuits and Systems, IEEE Transactions on*, vol. 7, no. 1, pp. 71–81, 2013.

[11] E. Farquhar and P. Hasler, "A Bio-Physically Inspired Silicon Neuron," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, no. 3, pp. 477 – 488, March 2005.

[12] M. Schmidt and D. Fey, "Akers's Wavefront Planner: One of the Fastest Stencil-Based Path Planners on FPGAs," in *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, 2012, pp. 1–6.

[13] J. Barraquand and J.-C. Latombe, *Robot Motion Planning: A Distributed Representation Approach*. Stanford Univ CA Dept of Computer Science, 1989.

[14] C. Erdal, A. Toker, and C. Acar, "Ota-C Based Proportional-Integral-Derivative (PID) Controller And Calculating Optimum Parameter Tolerances," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 9, no. 2, pp. 189–198, 2001.

[15] C. Schlottmann and P. Hasler, "A Highly Dense, Low Power, Programmable Analog Vector-Matrix Multiplier: The FPAA Implementation," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 1, no. 3, pp. 403 –411, Sept. 2011.

[16] T. H. Lee, *The Design of CMOS Radio-Frequency Integrated Circuits*. Cambridge University Press, 2004.

[17] A. S. Sedra and K. C. Smith, *Microelectronic Circuits Fifth Edition*. Oxford University Press, Inc., 2004.

[18] A. Mehta and K. Pister, "WARPWING: A Complete Open Source Control Platform for Miniature Robots," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 5169–5174.

[19] R. Groups.com. (2013, August) RC Controlled AR.Drone. . [Online]. Available: http://www.rcgroups.com/forums/showthread.php?t=1335257

[20] open source project. (2013, August) Ardudrone. . [Online]. Available: http://code.google.com/p/ardudrone/

[21] R. Siegwart and I. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. The MIT Press, 2004.

[22] N. Funk, "A Study of the Kalman Filter Applied to Visual Tracking," *Rap. tech, University of Alberta, Project for CMPUT 652*, 2003.

[23] P. Smyth, D. Heckerman, and M. Jordan, "Probabilistic Independence Networks for Hidden Markov Probability Models," *Neural computation*, vol. 9, no. 2, pp. 227–269, 1997.

[24] K. Murphy, "An Introduction to Graphical Models," *Rap. tech*, 2001.

[25] L. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257 –286, Feb. 1989.

[26] D. Titterton and J. Weston, *Strapdown Inertial Navigation Technology*. Peter Peregrinus Ltd, 2004, vol. 17.

[27] K. Gade, “Introduction to Inertial Navigation and Kalman Filtering.”

[28] L. Levy, “The Kalman filter: Navigation’s Integration Workhorse,” *GPS World*, vol. 8, no. 9, pp. 65–71, 1997.

[29] H. Woithe, I. Chigirev, D. Aragon, M. Iqbal, Y. Shames, S. Glenn, O. Schofield, I. Seskar, and U. Kremer, “Slocum Glider Energy Measurement and Simulation Infrastructure,” in *OCEANS 2010 IEEE - Sydney*, May 2010, pp. 1 –8.

[30] M. Eichhorn, “Solutions for Practice-Oriented Requirements for Optimal Path Planning for the AUV SLOCUM Glider,” in *OCEANS 2010*, Sept. 2010, pp. 1 –10.

[31] A. Basu, S. Brink, C. Schlottmann, S. Ramakrishnan, C. Petre, S. Koziol, F. Baskaya, C. Twigg, and P. Hasler, “A Floating-Gate-Based Field-Programmable Analog Array,” *Solid-State Circuits, IEEE Journal of*, vol. 45, no. 9, pp. 1781 –1794, Sept. 2010.

[32] R. Chawla, A. Bandyopadhyay, V. Srinivasan, and P. Hasler, “A 531 nW/MHz, 128 Times;32 Current-Mode Programmable Analog Vector-Matrix Multiplier with Over Two Decades of Linearity,” in *Custom Integrated Circuits Conference, 2004. Proceedings of the IEEE 2004*, Oct. 2004, pp. 651 – 654.

[33] B. Morgan and S. Bedair, “Power for Microsystems Strategic Technology Initiative Report on MAST Mission Power Requirements,” DTIC Document, Tech. Rep., 2009.

[34] S. Bouabdallah, M. Becker, and R. Siegwart, “Autonomous Miniature Flying Robots: Coming Soon! - Research, Development, and Results,” *Robotics Automation Magazine, IEEE*, vol. 14, no. 3, pp. 88 –98, Sept. 2007.

[35] N. Tsourveloudis, D. Gracanin, and K. Valavanis, “Design and Testing of Navigation Algorithm for Shallow Water Autonomous Underwater Vehicle,” in *OCEANS ’98 Conference Proceedings*, vol. 1, Sep-1 Oct 1998, pp. 342 –346 vol.1.

[36] C. Petres, Y. Pailhas, P. Patron, Y. Petillot, J. Evans, and D. Lane, “Path Planning for Autonomous Underwater Vehicles,” *Robotics, IEEE Transactions on*, vol. 23, no. 2, pp. 331 –341, April 2007.

[37] J. Ratti, J.-H. Moon, and G. Vachtsevanos, “Towards Low-Power, Low-Profile Avionics Architecture and Control for Micro Aerial Vehicles,” in *Aerospace Conference, 2011 IEEE*, March 2011, pp. 1 –8.

[38] J. Ratti and G. Vachtsevanos, “High Endurance, Micro Aerial Surveillance and Reconnaisance Robot,” in *Technologies for Practical Robot Applications (TePRA), 2011 IEEE Conference on*, April 2011, pp. 1 –6.

[39] L. Wowwee Group. (2013, August) Robosapien Website. [Online]. Available: www.wowwee.com/robosapien-x/

[40] G. I. of Technology. (2013, August) Georgia DARPA Urban Grand Challenge. [Online]. Available: www.gtresearchnews.gatech.edu/images/sting1_md.jpg

[41] G. Hu, W.-P. Tay, and Y. Wen, "Cloud Robotics: Architecture, Challenges and Applications," *Network, IEEE*, vol. 26, no. 3, pp. 21–28, 2012.

[42] E. Guizzo, "Robots with their Heads in the Clouds," *Spectrum, IEEE*, vol. 48, no. 3, pp. 16–18, 2011.

[43] J. Hutchinson, C. Koch, J.Luo, and C. Mead, "Computing Motion using Analog and Binary Resistive Networks," *Computer*, vol. 21, no. 3, pp. 52–63, 1988.

[44] M. Stan, W. Burleson, C. Connolly, and R. Grupen, "Analog VLSI for Robot Path Planning," *The Journal of VLSI Signal Processing*, vol. 8, no. 1, pp. 61–73, 1994.

[45] U. Roth, M. Walker, A. Hilmann, and H. Klar, "Dynamic Path Planning with Spiking Neural Networks," *Biological and Artificial Computation: From Neuroscience to Technology*, pp. 1355–1363, 1997.

[46] R. Glasius, A. Komoda, and S. C. Gielen, "Neural Network Dynamics for Path Planning and Obstacle Avoidance," *Neural Networks*, vol. 8, no. 1, pp. 125–133, 1995.

[47] C. Mead, "Neuromorphic Electronic Systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629 –1636, Oct. 1990.

[48] L. Tarassenko and A. Blake, "Analogue Computation of Collision-free Paths," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, Apr. 1991, pp. 540–545 vol.1.

[49] L. Tarassenko, G. Marshall, F. Gomez-Castaneda, and A. Murray, "Parallel Analogue Computation for Real-Time Path Planning," in *Proceedings of 2nd Workshop on VLSI for Artificial Intelligence and Neural Networks*, vol. 3, 1990.

[50] M. Stan and W. Burleson, "Analog VLSI for Robot Path Planning," in *Signals, Systems and Computers, 1992. 1992 Conference Record of The Twenty-Sixth Asilomar Conference on*, Oct. 1992, pp. 915 –919 vol.2.

[51] G. Marshall and L. Tarassenko, "Robot Path Planning Using VLSI Resistive Grids," in *Artificial Neural Networks, 1993., Third International Conference on*, May 1993, pp. 163 –167.

[52] ——, "Robot Path Planning Using Resistive Grids," in *Artificial Neural Networks, 1991., Second International Conference on*, Nov. 1991, pp. 149 –152.

[53] ——, "Robot Path Planning Using VLSI Resistive Grids," *Vision, Image and Signal Processing, IEE Proceedings* -, vol. 141, no. 4, pp. 267 –272, Aug. 1994.

[54] K. Althofer, D. Fraser, and G. Bugmann, "Rapid Path Planning for Robotic Manipulators Using an Emulated Resistive Grid," *Electronics Letters*, vol. 31, no. 22, pp. 1960 –1961, Oct. 1995.

[55] M. Kanaya, G.-X. Cheng, K. Watanabe, and M. Tanaka, “Shortest Path Searching for Robot Walking Using an Analog Resistive Network,” in *Circuits and Systems, 1994. ISCAS ’94., 1994 IEEE International Symposium on*, vol. 6, May-2 Jun 1994, pp. 311 –314 vol.6.

[56] R. Grupen, C. Connolly, K. Souccar, and W. Burleson, “Toward a Path Co-Processor for Automated Vehicle Control,” in *Intelligent Vehicles ’95 Symposium., Proceedings of the*, Sept. 1995, pp. 164 –169.

[57] L.-J. Yun, Z.-J. Liu, H.-X. Sun, and J. Yuan, “A Path Planner of Mobile Robot Based on Multi-Grid Circuit Map,” in *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, vol. 2, Aug. 2005, pp. 1279 – 1284 Vol. 2.

[58] M. Stan, W. Burleson, C. Connolly, and R. Grupen. (1994, May) Analog VLSI for Robot Path Planning. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.6921&rep=rep1&type=pdf

[59] K. B. Ariyur, E. Lautenschlager, and M. R. Elgersma, “Method and Device for Three-Dimensional Path Planning to Avoid Obstacles Using Multiple Planes,” U.S. Patent Application 0 090 228 205, Mar. 10, 2008.

[60] M. R. Elgersma, S. Dajani-Brown, S. G. Pratt, K. Fregene, and K. Ariyur, “Method and System for Automatic Path Planning and Obstacle/Collision Avoidance of Autonomous Vehicles,” U.S. Patent Application 0 088 916A1, June 4, 2008.

[61] S. Ravela, R. Weiss, B. Draper, B. Pinette, A. Hanson, and E. Riseman, “Stealth Navigation: Planning and Behaviors,” in *Proceedings of ARPA Image Understanding Workshop*, 1994, pp. 1093–1100.

[62] T. Zourntos and N. Mathai, “A BEAM-Inspired Lyapunov-Based Strategy for Obstacle Avoidance and Target-Seeking,” in *American Control Conference, 2007. ACC ’07*, July 2007, pp. 5302 –5309.

[63] M. Minsky and P. Seymour, *Perceptrons*. MIT Press, 1969.

[64] B. Widrow and M. Lehr, “30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation,” *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.

[65] T. Poggio and C. Koch, “Ill-Posed Problems in Early Vision: From Computational Theory to Analogue Networks,” *Proceedings of the Royal society of London. Series B. Biological sciences*, vol. 226, no. 1244, pp. 303–323, 1985.

[66] J. G. Harris, C. Koch, and J. Luo, “A Two-Dimensional Analog VLSI Circuit for Detecting Discontinuities in Early Vision,” *Science*, vol. 248, no. 4960, pp. 1209–1211, 1990.

[67] C. Koch, J. Marroquin, and A. Yuille, "Analog "neuronal" Networks in Early Vision," *Proceedings of the National Academy of Sciences*, vol. 83, no. 12, pp. 4263–4267, 1986.

[68] S. Koziol, C. Schlottmann, A. Basu, S. Brink, C. Petre, B. Degnan, S. Ramakrishnan, P. Hasler, and A. Balavoine, "Hardware and Software Infrastructure for a family of Floating-Gate Based FPAAs," IEEE International Symposium on Circuits and Systems (ISCAS) 2010, May 2010.

[69] S. Ganesan and R. Vemuri, "Analog-Digital Partitioning for Field-Programmable Mixed Signal Systems," in *Advanced Research in VLSI, 2001. ARVLSI 2001. Proceedings. 2001 Conference on*, 2001, pp. 172 –185.

[70] M. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks, "The Accelerator Store Framework for High-Performance, Low-Power Accelerator-Based Systems," *Computer Architecture Letters*, vol. 9, no. 2, pp. 53 –56, Feb. 2010.

[71] C. Winstead, J. Dai, W. J. Kim, and S. Little, "Analog MAP Decoder for (8, 4) Hamming Code in Subthreshold CMOS," in *Advanced Research in VLSI, 2001. ARVLSI 2001. Proceedings. 2001 Conference on*, 2001, pp. 132 –147.

[72] S. Koziol, R. Wunderlich, P. Hasler, and M. Stilman, "Path Planning Using Reconfigurable Analog VLSI," *to be submitted:*, 2013.

[73] C. Schlottmann, S. Shapero, S. Nease, and P. Hasler, "A Digitally Enhanced Dynamically Reconfigurable Analog Platform for Low-Power Signal Processing," *Solid-State Circuits, IEEE Journal of*, vol. 47, no. 9, pp. 2174–2184, 2012.

[74] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, Mar. 1985, pp. 500 – 505.

[75] C. Twigg and P. Hasler, "Incorporating Large-Scale FPAAs Into Analog Design and Test Courses," *Education, IEEE Transactions on*, vol. 51, no. 3, pp. 319–324, Aug. 2008.

[76] C. Twigg, P. Hasler, and F. Baskaya, "A Self-Contained Large-Scale FPAA Development Platform," in *Circuits and Systems, 2007. IEEE International Symposium on*, May 2007, pp. 1187–1191.

[77] T. Hall, C. Twigg, P. Hasler, and D. Anderson, "Application Performance of Elements in a Floating-Gate FPAA," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 2, May 2004, pp. II – 589–92 Vol.2.

[78] C. Twigg and P. Hasler, "A Large-Scale Reconfigurable Analog Signal Processor (RASP) IC," in *Custom Integrated Circuits Conference, 2006. CICC '06. IEEE*, Sept. 2006, pp. 5 –8.

[79] A. Basu, C. Twigg, S. Brink, P. Hasler, C. Petre, S. Ramakrishnan, S. Koziol, and C. Schlottmann, "RASP 2.8: A New Generation of Floating-Gate Based Field Programmable Analog Array," in *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, 2008, pp. 213–216.

[80] A. Basu, S. Ramakrishnan, and P. Hasler, "Neural Dynamics in Reconfigurable Silicon," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 30 2010-June 2 2010, pp. 1943 –1946.

[81] R. Wunderlich, F. Adil, and P. Hasler, "Floating Gate-Based Field Programmable Mixed-Signal Array," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 8, pp. 1496–1505, 2013.

[82] B. Pankiewicz, M. Wojcikowski, S. Szczepanski, and Y. Sun, "A Field Programmable Analog Array for CMOS Continuous-Time OTA-C Filter Applications," *Solid-State Circuits, IEEE Journal of*, vol. 37, no. 2, pp. 125 –136, Feb. 2002.

[83] E. Lee and P. Gulak, "Field Programmable Analogue Array Based on MOSFET Transconductors," *Electronics Letters*, vol. 28, no. 1, pp. 28 –29, Jan. 1992.

[84] C. Looby and C. Lyden, "A CMOS Continuous-Time Field Programmable Analog Array," in *Proceedings of the 1997 ACM fifth international symposium on Field-programmable gate arrays*. ACM, 1997, pp. 137–141.

[85] V. Gaudet and G. Gulak, "10 MHz Field Programmable Analog Array Prototype Based on CMOS Current Conveyors," in *Micronet Annual Workshop, Ottawa, Ontario*, 1999.

[86] J. Becker and Y. Manoli, "A Continuous-Time Field Programmable Analog Array (FPAA) Consisting of Digitally Reconfigurable GM-Eells," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 1, May 2004, pp. I – 1092–5 Vol.1.

[87] G. Cowan, R. Melville, and Y. Tsividis, "A VLSI Analog Computer/Math Co-Processor for a Digital Computer," in *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, Feb. 2005, pp. 82 –586 Vol. 1.

[88] J. Becker, F. Henrici, S. Trendelenburg, M. Ortmanns, and Y. Manoli, "A Field-Programmable Analog Array of 55 Digitally Tunable OTAs in a Hexagonal Lattice," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 12, pp. 2759 –2768, Dec. 2008.

[89] C. S. Corporation. (2013, August) Psoc5 webpage. [Online]. Available: www.cypress.com/psoc/

[90] T. Hall, C. Twigg, J. Gray, P. Hasler, and D. Anderson, "Large-Scale Field-Programmable Analog Arrays for Analog Signal Processing," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, no. 11, pp. 2298–2307, Nov. 2005.

[91] C. Twigg, J. Gray, and P. Hasler, "Programmable Floating Gate FPAA Switches Are Not Dead Weight," in *Circuits and Systems, 2007. IEEE International Symposium on*, May 2007, pp. 169–172.

[92] A. Basu and P. Hasler, "A Fully Integrated Architecture for Fast Programming of Floating Gates," in *Circuits and Systems, 2007. IEEE International Symposium on*, May 2007, pp. 957–960.

[93] C. Schlottmann, C. Petre, and P. Hasler, "A High-Level Simulink-Based Tool for FPAA Configuration," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 1, pp. 10 –18, Jan. 2012.

[94] I. Baskaya, "Physical Design Automation for Large Scale Field Programmable Analog Arrays," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, Aug. 2009. [Online]. Available: http://hdl.handle.net/1853/31810

[95] F. Baskaya, D. Anderson, P. Hasler, and S. K. Lim, "A Generic Reconfigurable Array Specification and Programming Environment (GRASPER)," in *Circuit Theory and Design, 2009. European Conference on*, Aug. 2009, pp. 619–622.

[96] F. Baskaya, B. Gestner, C. Twigg, S. K. Lim, D. Anderson, and P. Hasler, "Rapid Prototyping of Large-scale Analog Circuits With Field Programmable Analog Array," in *Field-Programmable Custom Computing Machines. 15th Annual IEEE Symposium on*, April 2007, pp. 319–320.

[97] F. Baskaya, D. Anderson, and S. K. Lim, "Net-Sensitivity-Based Optimization of Large-Scale Field-Programmable Analog Array (FPAA) Placement and Routing," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 56, no. 7, pp. 565–569, July 2009.

[98] L. Geppert, "The new Indelible Memories," *Spectrum, IEEE*, vol. 40, no. 3, pp. 48–54, 2003.

[99] P. E. Hasler, "Foundations of Learning in Analog VLSI," Ph.D. dissertation, California Institute of Technology, 1997.

[100] R. Robucci, "Development of a Computational Image Sensor With Applications in Integrated Sensing and Processing," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, May 2009. [Online]. Available: http://hdl.handle.net/1853/33943

[101] A. Basu, "Neural Dynamics in Reconfigurable Silicon," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, march 2010. [Online]. Available: http://hdl.handle.net/1853/39542

[102] R. Wood, "The First Takeoff of a Biologically Inspired At-Scale Robotic Insect," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 341–347, 2008.

[103] R. Wood, S. Avadhanula, E. Steltz, M. Seeman, J. Entwistle, A. Bachrach, G. Barrows, and S. Sanders, "An Autonomous Palm-Sized Gliding Micro Air Vehicle," *IEEE Robotics & Automation Magazine*, vol. 14, no. 2, pp. 82–91, 2007.

[104] R. Beard, D. Kingston, M. Quigley, D. Snyder, R. Christiansen, W. Johnson, T. McLain, and M. Goodrich, "Autonomous Vehicle Technologies for Small Fixed Wing UAVs," *AIAA Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 1, pp. 92–108, 2005.

[105] G. Torres and T. Mueller, "Micro Aerial Vehicle Development: Design, Components, Fabrication, and Flight-Testing," in *Association for Unmanned Vehicle Systems International (AUVSI) Unmanned Systems 2000 Symposium and Exhibition*, Orlando, FL, July 2000, pp. 123–456.

[106] S. Mittal and K. Deb, "Three-Dimensional Offline Path Planning for UAVs Using Multiobjective Evolutionary Algorithms," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, Sept. 2007, pp. 3195 –3202.

[107] Z. Qi, Z. Shao, Y. S. Ping, L. M. Hiot, and Y. K. Leong, "An Improved Heuristic Algorithm for UAV Path Planning in 3D Environment," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2010 2nd International Conference on*, vol. 2, Aug. 2010, pp. 258 –261.

[108] M. Hwangbo, J. Kuffner, and T. Kanade, "Efficient Two-phase 3D Motion Planning for Small Fixed-Wing UAVs," in *Robotics and Automation, 2007 IEEE International Conference on*, April 2007, pp. 1035 –1041.

[109] J. Yang, Z. Qu, J. Wang, and R. Hull, "A Real-Time Optimized Path Planning for a Fixed Wing Vehicle Flying in a Dynamic and Uncertain Environment," in *Advanced Robotics, 2005. ICAR '05. Proceedings., 12th International Conference on*, July 2005, pp. 96 –102.

[110] B. Weiss, M. Naderhirn, and L. del Re, "Global Real-Time Path Planning for UAVs in Uncertain Environment," in *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, Oct. 2006, pp. 2725 –2730.

[111] S. Bortoff, "Path Planning for UAVs," in *American Control Conference, 2000. Proceedings of the 2000*, vol. 1, no. 6, 2000, pp. 364–368 vol.1.

[112] E. Frazzoli, M. Dahleh, and E. Feron, "Real-Time Motion Planning for Agile Autonomous Vehicles," *Journal of Guidance Control and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[113] D. Pines and F. Bohorquez, "Challenges Facing Future Micro-Air-Vehicle Development," *Journal of aircraft*, vol. 43, no. 2, pp. 290–305, 2006.

[114] Y. Mei, Y.-H. Lu, Y. Hu, and C. S. G. Lee, "A Case Study of Mobile Robot's Energy Consumption and Conservation Techniques," in *Advanced Robotics, 2005. ICAR '05. Proceedings., 12th International Conference on*, 2005, pp. 492–497.

[115] J. Kok, L. Gonzalez, and N. Kelson, "FPGA Implementation of an Evolutionary Algorithm for Autonomous Unmanned Aerial Vehicle On-Board Path Planning," *Evolutionary Computation, IEEE Transactions on*, vol. 17, no. 2, pp. 272–281, 2013.

[116] M. Schmidt, M. Reichenbach, and D. Fey, "A Generic VHDL Template for 2D Stencil Code Applications on FPGAs," in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2012 15th IEEE International Symposium on*, 2012, pp. 180–187.

[117] D. Powers, *Boundary Value Problems: and Partial Differential Equations*. Academic Press, 2010.

[118] C. Connolly, J. Burns, and R. Weiss, "Path Planning Using Laplace's Equation," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, May 1990, pp. 2102 –2106 vol.3.

[119] C. Connolly and R. Grupen, "The Applications of Harmonic Functions to Robotics," *Journal of Robotic Systems*, vol. 10, no. 7, pp. 931 –946, June 1993.

[120] A. K. Mitra. (2013, August) Finite Difference Method for the Solution of Laplace Equation. Laplace.pdf. [Online]. Available: www.public.iastate.edu/~akmitra/aero361/design(underscore)web/Laplace.pdf

[121] C. Mead, *Analog VLSI and Neural Systems*. Addison Wesley Publishing Company. USA, 1989.

[122] S. Liu, *Analog VLSI: Circuits and Principles*. The MIT press, 2002.

[123] P. Smith and P. Hasler, "A Kappa Projection Algorithm (KPA) for Programming to Femtoampere Currents in Standard CMOS Floating-Gate Elements," in *Circuits and Systems, 2005. 48th Midwest Symposium on*, August 2005, pp. 75 –78 Vol. 1.

[124] R. Olfati-Saber, "Near-Identity Diffeomorphisms and Exponential ε-Tracking and ε-Stabilization of First-Order Nonholonomic SE (2) Vehicles," in *Proceedings of the American Control Conference*. Citeseer, 2002.

[125] M. Robots. (2013, August) Mobile Robots Website. [Online]. Available: www.mobilerobots.com

[126] R. Borras. (2010, April) Blob Detection. . [Online]. Available: www.ros.org/doc/api/cvblobslib/html/classCBlob.html

[127] B. Gerkey, R. Vaughan, K. Stoy, A. Howard, G. Sukhatme, and M. Mataric, "Most Valuable Player: A Robot Device Server for Distributed Control," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 3, 2001, pp. 1226–1231 vol.3.

[128] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[129] C. Connolly and R. Grupen, *Nonholonomic Path Planning Using Harmonic Functions*. Citeseer, 1994.

[130] M. Sipser, *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[131] J.-O. Kim and P. Khosla, "Real-Time Obstacle Avoidance Using Harmonic Potential Functions," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, Apr. 1991, pp. 790 –796 vol.1.

[132] ——, "Real-Time Obstacle Avoidance Using Harmonic Potential Functions," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 3, pp. 338 –349, Jun. 1992.

[133] L. Vacariu, F. Roman, M. Timar, T. Stanciu, R. Banabic, and O. Cret, "Mobile Robot Path-Planning Implementation in Software and Hardware," in *Proceedings of the 6th WSEAS International Conference on Signal Processing, Robotics and Automation*. World Scientific and Engineering Academy and Society (WSEAS), 2007, pp. 140–145.

[134] K. Sridharan, S. Lam, and T. Srikanthan, "VLSI Architectures for Autonomous Robots-A Review," *Autonomous Robots Research Advances*, p. 105, 2008.

[135] G. Indiveri, E. Chicca, and R. Douglas, "Artificial Cognitive Systems: From VLSI Networks of Spiking Neurons to Neuromorphic Cognition," *Cognitive Computation*, vol. 1, no. 2, pp. 119–127, 2009.

[136] R. Douglas, M. Mahowald, and C. Mead, "Neuromorphic Analogue VLSI," *Annual review of neuroscience*, vol. 18, pp. 255–281, 1995.

[137] T. Lande, *Neuromorphic Systems Engineering: Neural Networks in Silicon*. Springer, 1998, vol. 447.

[138] L. Smith and A. Hamilton, *Neuromorphic Systems: Engineering Silicon from Neurobiology*. World Scientific Publishing Company Incorporated, 1998, vol. 10.

[139] F. Kendoul, "Survey of Advances in Guidance, Navigation, and Control of Unmanned Rotorcraft Systems," *Journal of Field Robotics*, 2012.

[140] Y. K. Hwang and N. Ahuja, "Gross Motion PlanningA Survey," *ACM Computing Surveys (CSUR)*, vol. 24, no. 3, pp. 219–291, 1992.

[141] J. Barraquand and J. Latombe, "Robot Motion Planning: A Distributed Representation Approach," *The International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, 1991.

[142] S. M. LaValle, *Planning Algorithms*. Cambridge university press, 2006.

[143] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma, "Flying Fast and Low Among Obstacles: Methodology and Experiments," *The International Journal of Robotics Research*, vol. 27, no. 5, p. 549, 2008.

[144] S. Scherer, S. Singh, L. Chamberlain, and S. Saripalli, "Flying Fast and Low Among Obstacles," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 2023–2029.

[145] R. Mahony, V. Kumar, and P. Corke, "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor," *Robotics Automation Magazine, IEEE*, vol. 19, no. 3, pp. 20 –32, Sept. 2012.

[146] H. Lim, J. Park, D. Lee, and H. Kim, "Build Your Own Quadrotor: Open-Source Projects on Unmanned Aerial Vehicles," *Robotics Automation Magazine, IEEE*, vol. 19, no. 3, pp. 33 –45, Sept. 2012.

[147] J.-C. Latombe, *Robot Motion Planning*. Springer, 1991.

[148] J. Lengyel, M. Reichert, B. Donald, and D. Greenberg, "Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware," in *ACM SIGGRAPH Computer Graphics*, vol. 24, no. 4. ACM, 1990, pp. 327–335.

[149] L. Dorst and K. Trovato, "Optimal Path Planning by Cost Wave Propagation in Metric Configuration Space," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, vol. 1007, 1988, p. 186.

[150] J. Mitchell and C. Papadimitriou, "The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision," *Journal of the ACM (JACM)*, vol. 38, no. 1, pp. 18–73, 1991.

[151] P. Batavia and I. Nourbakhsh, "Path Planning for the Cye Personal Robot," in *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 1, 2000, pp. 15 –20 vol.1.

[152] J. Lin, P. Merolla, J. Arthur, and K. Boahen, "Programmable Connections in Neuromorphic Grids," in *Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest Symposium on*, vol. 1, Aug. 2006, pp. 80 –84.

[153] A. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "Pynn: a common interface for neuronal network simulators," *Frontiers in neuroinformatics*, vol. 2, 2008.

[154] N. C. Rowe and R. S. Alexander, "Finding Optimal-Path Maps for Path Planning Across Weighted Regions," *The International Journal of Robotics Research*, vol. 19, no. 2, pp. 83–95, 2000.

[155] C. Goerzen, Z. Kong, and B. Mettler, "A Survey of Motion Planning Algorithms From the Perspective of Autonomous UAV Guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1-4, pp. 65–100, 2010.

[156] S. Koziol, S. Brink, and P. Hasler, "A neuromorphic approach to path planning using a reconfigurable neuron array ic," in *Under review:*, 2013.

[157] K. Stoy. (2011, March) The Player Project. . [Online]. Available: http://playerstage.sourceforge.net

[158] S. Zhao, C. Chen, C. Liu, and M. Liu, "Algorithm of Location of Chess-Robot System Based on Computer Vision," in *Control and Decision Conference, 2008. CCDC 2008. Chinese*, July 2008, pp. 5215 –5218.

[159] A. Curtis, J. Shim, E. Gargas, A. Srinivasan, and A. Howard, "Dance Dance Pleo: Developing a Low-Cost Learning Robotic Dance Therapy Aid," in *Proceedings of the 10th International Conference on Interaction Design and Children*. ACM, 2011, pp. 149–152.

[160] A. Trevor, H. W. Park, A. Howard, and C. Kemp, "Playing With Toys: Towards Autonomous Robot Manipulation for Therapeutic Play," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, May 2009, pp. 2139 –2145.

[161] M. Jones, T. Trapp, N. Jones, D. Brooks, and A. Howard. (2010) Engaging Childern with Severe Physical Disabilities via Teleoperated Control of a Robot Piano Player. [Online]. Available: http://vip.gatech.edu/wiki/images/6/65/Robotpianoplayer.pdf

[162] H. W. Park and A. Howard, "Case-Based Reasoning for Planning Turn-Taking Strategy With a Therapeutic Robot Playmate," in *Biomedical Robotics and Biomechatronics (BioRob), 2010 3rd IEEE RAS and EMBS International Conference on*, Sept. 2010, pp. 40 –45.

[163] E. Krotkov, "Robotics Laboratory Exercises," *IEEE Transactions on Education*, vol. 39, no. 1, pp. 94–97, 1996.

[164] I. J. Goodfellow, N. Koenig, M. Muja, C. Pantofaru, A. Sorokin, and L. Takayama, "Help Me Help You: Interfaces for Personal Robots," in *Human-Robot Interaction (HRI), 2010 5th ACM/IEEE International Conference on*, 2-5 2010, pp. 187 –188.

[165] W.-H. Fu, J. Jiang, X. Qin, T. Yi, and Z.-L. Hong, "A Reconfigurable Analog Processor Based on FPAA with Coarse-Grained, Heterogeneous Configurable Analog Blocks," in *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, 31 2010-Sept. 2 2010, pp. 211 –216.

[166] P. Dong, G. Bilbro, and M.-Y. Chow, "Controlling A Path-Tracking Unmanned Ground Vehicle With a Field-Programmable Analog Array," in *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on*, 2005, pp. 1263–1268.

[167] V. Aggarwal, M. Mao, and U.-M. O'Reilly, "A Self-Tuning Analog Proportional-Integral-Derivative (PID) Controller," in *Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on*, June 2006, pp. 12 –19.

[168] M. A. Terry, "Evolving Circuits on a Field Programmable Analog Array Using Genetic Programming," Boston, May 2005.

[169] J. Dorsey, *Continuous and Discrete Control Systems*. McGraw Hill, 2002.

[170] K. H. Ang, G. Chong, and Y. Li, "PID Control System Analysis, Design, and Technology," *Control Systems Technology, IEEE Transactions on*, vol. 13, no. 4, pp. 559 – 576, July 2005.

[171] Y. Li, K. H. Ang, and G. Chong, "PID Control System Analysis and Design," *Control Systems Magazine, IEEE*, vol. 26, no. 1, pp. 32 – 41, Feb. 2006.

[172] ——, "Patents, Software, and Hardware for PID Control: An Overview and Analysis of the Current Art," *Control Systems, IEEE*, vol. 26, no. 1, pp. 42 – 54, feb. 2006.

[173] V. Michal, C. Premont, G. Pillonet, and N. Abouchi, "Single active element PID controllers," in *Radioelektronika, 2010 20th International Conference*, April 2010, pp. 1 –4.

[174] C. Erdal, "A New Current-Feedback Amplifiers (CFA) Based Proportional-Integral-Derivative (PID) Controller Realization And Calculating Optimum Parameter Tolerances," *Pakistan Journal Of Applied Sciences*, vol. 2, no. 1, pp. 56–59, 2002.

[175] A. Stoica, D. Keymeulen, M. Mojarradi, R. Zebulum, and T. Daud, "Progress in the Development of Field Programmable Analog Arrays for Space Applications," in *Aerospace Conference, 2008 IEEE*, March 2008, pp. 1 –9.

[176] D. Keymeulen, A. Stoica, R. Zebulum, S. Katkoori, P. Fernando, H. Sankaran, M. Mojarradi, and T. Daud, "Self-Reconfigurable Analog Array Integrated Circuit Architecture for Space Applications," in *Adaptive Hardware and Systems, 2008. AHS '08. NASA/ESA Conference on*, June 2008, pp. 83 –90.

[177] S. by Willow Garage. (2010, May) OpenCV. Webpage. [Online]. Available: http://opencv.willowgarage.com/wiki/

[178] A. Singh. (2013, August) Hanoimania! Webpage.

[179] L. Itti and C. Koch, "Computational Modelling of Visual Attention," *Nature reviews neuroscience*, vol. 2, no. 3, pp. 194–203, 2001.

[180] J. Hasler, "Ultra Low-power Sensor Hardware for Autonomous Copter Tracking," *Work in progress*, July 2012.

[181] L. Itti and C. Koch, "Feature Combination Strategies for Saliency-Based Visual Attention Systems," *Journal of Electronic Imaging*, vol. 10, no. 1, pp. 161–169, 2001.

[182] D. Matthys. (2001, march) Log filter. [Online]. Available: http://academic.mu.edu/phys/matthysd/web226/Lab02.htm

[183] K. Fisher, S. Perkins, A. Walker, and E. Wolfart. (2003) Hypermedia image processing refernece. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

[184] Atmel. (2013, August) Arm926 processor performance specifications. [Online]. Available: http://www.arm.com/products/processors/classic/arm9/arm926.php

[185] C. Schlottmann, C. Petre, and P. Hasler, "Vector Matrix Multiplier on Field Programmable Analog Array," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, 2010, pp. 1522–1525.

[186] C. Mead and M. Ismail, *Analog VLSI Implementation of Neural Systems*. Springer, 1989.

[187] J. Nilsson and S. Riedel, *Electric Circuits*. Addison-Wesley Pub. Co., 1996.

[188] P. Hasler and J. Dugger, "An Analog Floating-Gate Node for Supervised Learning," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, no. 5, pp. 834–845, 2005.

[189] N. Tsourveloudis, D. Gracanin, and K. Valavanis, "Design and Testing of Navigation Algorithm for Shallow Water Autonomous Underwater Vehicle," in *OCEANS '98 Conference Proceedings*, vol. 1, 1998, pp. 342–346 vol.1.

[190] C. Petres, Y. Pailhas, Y. Petillot, and D. Lane, "Underwater Path Planing Using Fast Marching Algorithms," in *Oceans 2005 - Europe*, vol. 2, 2005, pp. 814–819 Vol. 2.

[191] C. Petres, Y. Pailhas, P. Patron, Y. Petillot, J. Evans, and D. Lane, "Path Planning for Autonomous Underwater Vehicles," *Robotics, IEEE Transactions on*, vol. 23, no. 2, pp. 331–341, 2007.

[192] P. Hasler, S. Kozoil(sic), E. Farquhar, and A. Basu, "Transistor Channel Dendrites Implementing HMM Classifiers," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, May 2007, pp. 3359 –3362.

[193] S. George, J. Hasler, S. Koziol, S. Nease, and S. Ramakrishnan, "Low Power Dendritic Computation for Wordspotting," *Journal of Low Power Electronics and Applications*, vol. 3, no. 2, pp. 73–98, 2013.

[194] S. Nease, S. George, P. Hasler, S. Koziol, and S. Brink, "Modeling and Implementation of Voltage-Mode CMOS Dendrites on a Reconfigurable Analog Platform," *Biomedical Circuits and Systems, IEEE Transactions on*, vol. 6, no. 1, pp. 76–84, 2012.

[195] S. George and P. Hasler, "HMM Classifier Using Biophysically Based CMOS Dendrites for Wordspotting," in *Biomedical Circuits and Systems Conference (BioCAS), 2011 IEEE*, Nov. 2011, pp. 281 –284.

[196] P. Smith and P. Hasler, "A Programmable Diffuser Circuit Based on Floating-Gate Devices," in *Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium on*, vol. 1, Aug. 2002, pp. I – 291–4 vol.1.

[197] P. D. Smith and P. Hasler, "Analog Speech Recognition Project," in *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, vol. 4, May 2002, pp. IV–3988 –IV–3991.

[198] P. Hasler, P. Smith, E. Farquhar, and D. Anderson, "A Neuromorphic IC Connection Between Cortical Dendritic Processing and HMM Classification," in *Digital Signal Processing Workshop, 2004 and the 3rd IEEE Signal Processing Education Workshop. 2004 IEEE 11th*, Aug. 2004, pp. 334 – 337.

[199] P. Smith, "An Analog Architecture for Auditory Feature Extraction and Recognition," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, Nov. 2004. [Online]. Available: http://hdl.handle.net/1853/4839

[200] S. George and P. Hasler, "HMM Classifier Using Biophysically Based CMOS Dendrites for Wordspotting," in *Biomedical Circuits and Systems Conference (BioCAS), 2011 IEEE*, 2011, pp. 281–284.

[201] J. Lazzaro, J. Wawrzynek, and R. Lippmann, "A Micropower Analog VLSI HMM State Decoder for Wordspotting," in *NIPS*. Citeseer, 1996, pp. 727–733.

[202] ——, "A Micropower Analog Circuit Implementation of Hidden Markov Model State Decoding," *Solid-State Circuits, IEEE Journal of*, vol. 32, no. 8, pp. 1200–1209, 1997.

[203] S. Koziol, J. Wang, and H. Wu, "Feature Integration for Robust Detection: Fusing Laser and Vision to Detect Doorways," *CS8803, Georgia Institute of Technology*, Dec. 2009.

[204] D. Schröter, T. Weber, M. Beetz, and B. Radig, "Detection and Classification of Gateways for the Acquisition of Structured Robot Maps," in *Proc. of 26th Pattern Recognition Symposium (DAGM), Tübingen/Germany*, 2004.

[205] B.-H. Juang and S. Furui, "Automatic Recognition and Understanding of Spoken Language - A First Step Toward Natural Human-Machine Communication," *Proceedings of the IEEE*, vol. 88, no. 8, pp. 1142 –1165, Aug. 2000.

[206] S. Mitra and T. Acharya, "Gesture Recognition: A Survey," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 3, pp. 311 –324, May 2007.

[207] Q. Zhu, "Hidden Markov Model for Dynamic Obstacle Avoidance of Mobile Robot Navigation," *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 3, pp. 390 –397, Jun. 1991.

[208] L. Rabiner and B. Juang, "An Introduction to Hidden Markov Models," *ASSP Magazine, IEEE*, vol. 3, no. 1, pp. 4 – 16, Jan. 1986.

[209] ——, *Fundamentals of speech recognition*. Prentice hall Englewood Cliffs, New Jersey, 1993, vol. 103.

[210] W. Wei, B. Wang, and D. Towsley, "Continuous-Time hidden Markov models for network performance evaluation," *Performance Evaluation*, vol. 49, no. 1-4, pp. 129–146, 2002.

[211] D. Carter, "A Circuit Theory of the Kalman Filter," in *Decision and Control, 1993., Proceedings of the 32nd IEEE Conference on*, Dec. 1993, pp. 1224 –1226 vol.2.

[212] P. Vontobel and H.-A. Loeliger, "Factor Graphs and Dynamical Electrical Networks," in *Information Theory Workshop, 2003. Proceedings. 2003 IEEE*, March-4 April 2003, pp. 218 – 221.

[213] P. Vontobel, D. Lippuner, and H. Loeliger, "Kalman Filters, Factor Graphs, and Electrical Networks," in *Proc. Fifteenth Intern. Symp. on Math. Theory of Networks and Systems*. Citeseer, 2002, pp. 12–16.

[214] Wikipedia. Navigation. . [Online]. Available: http://en.wikipedia.org/wiki/Navigation

[215] O. Woodman, "An Introduction to Inertial Navigation," *University of Cambridge, Computer Laboratory, Tech. Rep. UCAMCL-TR-696*, 2007.

[216] K. Gade, "Introduction to Inertial Navigation."

[217] P. Zarchan, "Tactical and Strategic Missile Guidance, ser," *Progress in Astronautics and Aeronautics. Reston, VA: American Institute of Aeronautics and Astronautics, Inc*, vol. 176, 1997.

[218] Y. Bar-Shalom and X. Li, "Estimation and Tracking- Principles, Techniques, and Software," *Norwood, MA: Artech House, Inc, 1993.*, 1993.

[219] H. Sorenson and A. Stubberud, "Linear Estimation Theory (Mathematical Models for Unbiased Minimum Variance Linear Estimation Problem)," *Theory and applications of Kalman filtering, editor Leondes, C.T.*, pp. 1–42, 1970.

[220] A. Kelly, "A 3D State Space Formulation of a Navigation Kalman Filter for Autonomous Vehicles," DTIC Document, Tech. Rep., 1994.

[221] A. Brown, "GPS/INS Uses Low-Cost MEMS IMU," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 20, no. 9, pp. 3–10, 2005.

[222] L. Xue, W. Yuan, H. Chang, and C. Jiang, "MEMS-based Multi-Sensor Integrated Attitude Estimation Technology for MAV Applications," in *Nano/Micro Engineered and Molecular Systems, 2009. NEMS 2009. 4th IEEE International Conference on*, Jan. 2009, pp. 1031 –1035.

[223] C. Brigante, N. Abbate, A. Basile, A. Faulisi, and S. Sessa, "Towards Miniaturization of a MEMS-Based Wearable Motion Capture System," *Industrial Electronics, IEEE Transactions on*, vol. 58, no. 8, pp. 3234 –3241, Aug. 2011.

[224] H. B. Christophersen, R. W. Pickell, J. C. Neidhoefer, A. A. Koller, S. K. Kannan, and E. N. Johnson, "A Compact Guidance, Navigation, and Control System for Unmanned Aerial Vehicles," *Journal of aerospace computing, information, and communication*, vol. 3, no. 5, pp. 187–213, 2006.

[225] F. Kendoul, Y. Zhenyu, and K. Nonami, "Embedded Autopilot for Accurate Waypoint Navigation and Trajectory Tracking: Application to Miniature Rotorcraft UAVs," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, May 2009, pp. 2884 –2890.

[226] Y. Bar-Shalom, X. Li, T. Kirubarajan, and J. Wiley, *Estimation with Applications to Tracking and Navigation*. Wiley Online Library, 2001.

[227] W. Pieczynski and F. Desbouvries, "Kalman Filtering Using Pairwise Gaussian Models," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, vol. 6, April 2003, pp. VI – 57–60 vol.6.

[228] D. Simon, "Kalman Filtering," *Embedded Systems Programming*, vol. 14, no. 6, pp. 72–79, 2001.

[229] S. Roweis and Z. Ghahramani, "A Unifying Review of Linear Gaussian Models," *Neural computation*, vol. 11, no. 2, pp. 305–345, 1999.

[230] R. Elliott, L. Aggoun, and J. Moore, *Hidden Markov Models: Estimation and Control*. Springer, 1995, vol. 29.

[231] P. Caines, *Linear Stochastic Systems*. John Wiley & Sons, Inc., 1987.

[232] P. Ainsleigh, N. Kehtarnavaz, and R. Streit, "Hidden Gauss-Markov Models for Signal Classification," *Signal Processing, IEEE Transactions on*, vol. 50, no. 6, pp. 1355 –1367, Jun. 2002.