SEMANTIC MAPPING FOR SERVICE ROBOTS: BUILDING AND USING MAPS FOR MOBILE MANIPULATORS IN SEMI-STRUCTURED ENVIRONMENTS

A Thesis
Presented to
The Academic Faculty

by

Alexander J. B. Trevor

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology
May 2015

SEMANTIC MAPPING FOR SERVICE ROBOTS: BUILDING AND USING MAPS FOR MOBILE MANIPULATORS IN SEMI-STRUCTURED ENVIRONMENTS

ACKNOWLEDGEMENTS

Much of the work presented in this thesis was performed in collaboration with other students at Georgia Tech. I want to thank my colleagues at RIM, especially John G. Rogers III, Changhyun Choi, Akansel Cosgun, Carlos Nieto, and Siddharth Choudhary, who all collaborated on this work.

Thanks also to the Boeing Corporation for financially supporting this work, and providing the opportunity to test at their facilities. I had the opportunity to intern at Willow Garage, where I learned a great deal from my mentors Michael Dixon, Suat Gedikli, and Radu Rusu. Thanks also to my colleagues at Fyusion, particularly Radu Rusu, Stefan Holzer, and Stephen Miller, for their support during the final part of my graduate studies.

I'd also like to thank Mike Stilman for his friendship, and for being an inspiration to all of us at RIM. Mike expanded my thinking about what robots can be, especially by pointing out that for sufficiently large and powerful robots, no part of a map needs to be considered static. Mike is missed dearly.

Thanks to my committee for their helpful comments and discussions: Frank Dellaert, Dieter Fox, Ayanna Howard, and Professor James Rehg. Of course, a special thanks to Henrik Christensen for his inspiration, support, and patience for so very many years. Thanks also to my parents, who have done so much for me. Finally, thanks to my wife Crystal Chao, whose love, support, and comments on this research were invaluable.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . **iii**

LIST OF TABLES . . . . **ix**

LIST OF FIGURES . . . . **x**

SUMMARY . . . . **xx**

I INTRODUCTION **1**

1.1 Service Robots 1

1.2 Localization and Mapping 1

1.3 Semantic Mapping 2

1.3.1 What are maps for? 5

1.4 Challenges 6

1.4.1 Efficient Semantic Segmentation 6

1.4.2 Complex Features for SLAM 7

1.4.3 Multi-modal Mapping 7

1.4.4 Semantic Labeling 8

1.5 Contributions 8

1.6 Thesis Statement 9

1.7 Outline 9

II RELATED WORK **11**

2.1 Simultaneous Localization and Mapping (SLAM) 11

2.2 Semantic Mapping 13

2.3 Human Augmented Mapping 15

2.4 Object Detection & Object Recognition 15

III 3D PERCEPTION: SEMANTIC SEGMENTATION & FEATURE DETECTION **17**

3.1 3D Edge Detection 18

3.1.1 Occluding, Occluded, & Boundary Edges 18

3.1.2 High Curvature Edges 20
3.1.3 RGB Edges 21
3.1.4 Edge Detection Runtime 22
3.2 Line Segmentation of Laser Scans 23
3.3 Unorganized Point Cloud Segmentation 25
3.3.1 Discussion 26
3.4 Organized Connected Component Segmentation 27
3.4.1 Related Work 27
3.4.2 Approach 29
3.4.3 Planar Segmentation 31
3.4.4 Planar Refinement Algorithm 33
3.4.5 Planar Segmentation with Color 37
3.4.6 Euclidean Clustering 37
3.4.7 Road Segmentation 38
3.4.8 Implementation 41
3.4.9 Discussion & Evaluation 42
3.5 Object Discovery & Recognition 55
3.5.1 Object Detection 55
3.5.2 3D Cluster Recognition 57
3.5.3 Door Sign Recognition 58
IV SLAM & SEMANTIC MAPPING 63
4.1 3D Edge-based SLAM 63
4.1.1 Edge-based Pair-wise Registration 64
4.1.2 Edge-based SLAM 69
4.2 Plane Landmarks 70
4.2.1 Related Work 71
4.2.2 Planar Mapping Approach 74
4.2.3 Laser Plane Measurements 77

4.2.4 Discussion 78

4.3 Object Landmarks: Door Signs 79

4.3.1 Related Work 81

4.3.2 Door sign landmarks 81

4.4 Object Landmarks: SLAM with Object Discovery, Modeling, and Mapping 85

4.4.1 Object Mapping Approach 88

4.4.2 Object Discovery, Recognition & Modeling 92

4.5 Modeling Relationships between landmarks with Virtual Measurements 94

4.5.1 Virtual Measurements 95

4.5.2 Virtual Measurements Results 97

4.6 Moveable Objects 102

4.6.1 Related Work 103

4.6.2 Moveable Object Approach 104

4.6.3 Moveable Object Results 109

4.6.4 Discussion 114

V OMNIMAPPER: A MODULAR MULTI-MODAL MAPPING SYSTEM 116

5.1 Related Work 120

5.2 Mapping Approach 121

5.2.1 Factor Graphs & Optimization 122

5.2.2 Common Measurement Types 122

5.2.3 Trajectory Management 124

5.3 Software Architecture 130

5.3.1 Optimization & SLAM Backend 130

5.3.2 OmniMapperBase 131

5.3.3 Measurement Plugins 133

5.3.4 Pose Plugins 135

5.3.5 Output Plugins 136

5.3.6 Adding User Defined Plugins 137
5.4 Example Mapping Applications 137
5.4.1 3D Semantic Mapping for Service Robots 137
5.4.2 Large Scale 3D Mapping 138
5.4.3 Mapping with a handheld RGB-D Camera 138
5.4.4 Scan Matching for Mobile Robots in Industrial Environments 138
5.5 Mapping Results 142
5.5.1 3D Edge-based SLAM Results 142
5.5.2 3D and 2D Measurements of Planar Landmarks Results . . . 146
5.5.3 Object Landmarks: Door Signs Results 149
5.5.4 Object Landmarks: Online Object Discovery 158
5.6 Discussion 169
VI HUMAN-ROBOT INTERACTION FOR MAP ANNOTATION 170
6.1 Pointing Gestures for Human-Robot Interaction 170
6.1.1 Related Works 172
6.1.2 Pointing Gesture Recognition 174
6.1.3 Representing Pointing Directions 175
6.1.4 Determining Intended Target 176
6.1.5 Data Collection 178
6.1.6 Error Analysis 180
6.1.7 Pointing Evaluation 184
6.1.8 Conclusions 188
6.2 Interactive Map Annotation & Object Modeling 188
6.2.1 Related Work 191
6.2.2 Approach 192
6.2.3 Gesture Recognition 194
6.2.4 Person Following 195
6.2.5 Tablet UI 196

6.2.6 Navigating to labeled structures 196
6.2.7 Annotation Results 198
6.2.8 Labeling Object Models 201
6.2.9 Discussion and Conclusion 202
6.3 Mapping Human Usage of Space 205
6.3.1 Mapping Approach 206
6.3.2 Standby Location Selection 207
6.3.3 Experiment & User Study 208
6.3.4 Results 213
6.3.5 Discussion 215

VII CONCLUSIONS & FUTURE WORK 218

7.1 Efficient Semantic Segmentation 219
7.2 Semantic Features for SLAM 220
7.3 Multi-modal Mapping 221
7.4 Semantic Labeling 222

REFERENCES 223

LIST OF TABLES

1 Average computation time of edges in Freiburg 1 sequences 23

2 Average running times for normal estimation and plane segmentation without planar refinement. 43

3 Average running times for normal estimation and plane segmentation with planar refinement. 44

4 Comparison of running times to our previous approach. Note that RANSAC based approaches were only run on down-sampled data. . . 45

5 Ground Segmentation Results 54

6 Door Sign classifier SVM cross validation results. 61

7 SVM confusion matrix results on trained buildings. True positive rate is listed on the left, true negative rate is listed on the right. 61

8 RMSE of translation, rotation, and average time in Freiburg 1 sequences 66

9 Simulated results both with and without virtual measurements, with respect to the ground truth from the simulator. 100

10 Performance of RGB-D SLAM [43] and our edge-based SLAM over some of Freiburg 1 sequences 145

11 Number of true positive objects discovered and the effect of object refinement for merging under-segmented objects. 163

12 μ and σ angular errors in degrees for each of Targets 1-4 (Figure 90), for each pointing method. 182

13 μ and σ of angular error in degrees for Targets 5-7 (Figure 90), for each pointing method. The aggregate μ and σ is also shown. 182

LIST OF FIGURES

1 An example of a map without semantic information. 3

2 An example of a map with semantic labels. 4

3 An example of a map with semantic labels, and additional semantic information. 4

4 Example edges in several scenes. Top Row: *occluding edges* are shown in green, *occluded edges* are shown in red, and *boundary edges* are shown in blue. Center Row: *RGB* edges are shown in cyan. Bottom Row: *high curvature* edges are shown in yellow. Figure best viewed in color. 19

5 Surface normal image response to Sobel operator in x (left) and y (right). 21

6 Surface normal gradients $\mathbf{G}_x$ (left) and $\mathbf{G}_y$ (right). 21

7 Detected high curvature edges, shown in yellow. 22

8 An example of 2d line segments extracted from a laser scan. Top: raw laser scan points. Bottom: detected line segments. Note that many walls are not perfectly flat, especially at edges, so some portions of walls may be excluded. 24

9 An example scene segmented using this RANSAC-based approach. Segmented planes are outlined in green. 26

10 An example of a segmented tabletop scene. Planar surfaces are outlined in color, with large red arrows indicating the segmented plane's normal. Euclidean clusters are indicated by colored points. 28

11 An example where two labels must be merged, if the pixel highlighted in blue matches both the neighbor above and to the left, shown in green. Merging is performed in a second pass using the union-find algorithm. 31

12 An example scene with noisy surface normals near object boundaries. 34

13 Top: a scene segmented without using the planar refinement approach described in Section 3.4.4. Bottom: a similar scene segmented with the planar refinement approach. 35

14 Planar refinement requires two additional passes over the image, one that extends regions "down and right", and one that extends regions "up and left". 36

15 An example scene segmented for planes of similar color. The sheets of paper on the desk are segmented as separate planar regions, as their color does not match the desk. 37

16 A diagram demonstrating the "expected ground plane" – the plane relative to the sensor pose that the vehicle's wheels should rest on. . . 39

17 Given the expected ground plane normal in the center, the normals at the sides indicate the acceptable range of normals given an angular threshold of 15 degrees. 39

18 At left, adjacent normals with acceptably smooth variation given an angular threshold of 2 degrees. At right, normals without smooth variation. Note that all normals shown are within the acceptable range from the example above. 40

19 An example of the obstacle detection. Any connected component not part of the road is considered as an obstacle. 41

20 A system diagram demonstrating our planar segmentation processing pipeline. The normal estimation and plane segmentation operations occur in parallel, enabling real-time performance. 42

21 An example image from the desk1 dataset, with a colorized representation of the label image superimposed on top. The displayed label image was generated using the planar segmentation approach with refinement. 44

22 Comparison of running times to our previous approach. Note that RANSAC based approaches were only run on 2.5cm voxel grid downsampled data both for the dominant plane only (center) and for all planes (right), while full resolution data was used for our approach (left). 46

23 An example of an indoor mapping tasks using planar landmarks. The colored points represent boundaries of planar surfaces segmented from RGB-D data, and merged from multiple observations. 48

24 Example scenes of Kinect data segmented using our approach. This was performed to validate the segmentation approach during the development of the stereo tools. 50

25 An example of our segmentation results on the Castro dataset. Segments detected using the road surface comparator are shown in green, and pixels added by the road region growing are shown in yellow. Pixels without valid depth data are shown in blue. 51

26 Top: an example of the ground truth segmentation for frame 2500 of the Castro dataset. Roadway is labeled in green, while sidewalks, curbs, and other low-lying areas are labeled in yellow. All other pixels are labeled with white. Bottom: our segmentation result for this image. 52

27 An image of a door sign seen by the robot is shown in figure 27(a) and the resulting saliency mask is shown in figure 27(b) 56

28 Examples images of signs from our classifier's training set. Signs on the top are from the College of Computing dataset, and signs on the bottom are from the Klaus data set. 61

29 Plots of trajectories from pair-wise registrations. 67

30 An example of the type of map produced by our system. Planar features are visible by the red convex hulls, and red normal vectors. The small red arrows on the ground plane show the robot's trajectory. The point clouds used to extract these measurements are shown in white, and have been rendered in the map coordinate frame by making use of the optimized poses from which they were taken. 72

31 A visualization of the relative fields of view of our Hokuyo UTM-30LX and Asus Xtion range camera. A top-down view is shown on top, with the laser's FOV shown in blue, and the range camera's FOV shown in green. Below this, is a side view of the relative FOVs. The 2D line is the FOV of the laser scanner, while the green triangle is the FOV of the range camera. 72

32 A system diagram, showing an overview of our approach. 74

33 A drawing demonstrating a 2D measurement of a 3D plane. Such measurements are under-constrained, and have a rotational degree of freedom about the measured line. 77

34 Laser scanner data from the robot. The structure of the hallway is recognized by the laser line extractor as two walls on either side, shown with green lines. The wall at the end of the hall is too small in this view to be recognized as a line. Raw laser points are shown in white. 82

35 Snapshot of the process at one instant. The robot trajectory is shown in red. Green lines add constraints between the object landmarks and the robot poses they are seen from. Light background shows the aggregated map cloud generated using the current SLAM solution. 86

36 Flowchart of the data processing pipeline. 88

37 The robot platform used for these experiments. The camera attached to the laptop is the one that is used to detect the AR ToolKit Plus markers. 97

38 The experimental setup. The black and white squares on the cabinets are the AR ToolKit markers used as landmarks. 98

39 The initial state of the map, before optimization. Poses are shown in red (dark), static landmarks are shown in green (light), and walls are shown in blue. 99

40 The map after optimization, but without using constraints between features. Poses are shown in red, static landmarks are shown in green, and walls are shown in blue. 99

41 The map after optimization, including constraints between features. Poses are shown in red, static landmarks are shown in green, and walls are shown in blue. 99

42 The initial state of our simulated experiment. Note that the robot's initial belief (shown) is that it moved in a perfect circle. The "actual" poses in the simulator were corrupted by Gaussian noise. Poses are shown in red, static landmarks are shown in green. 101

43 The simulated experiment after optimization, without inclusion of virtual measurements. Poses are shown in red, static landmarks are shown in green. 101

44 The simulated experiment after optimization, including constraints between features. Poses are shown in red, static landmarks are shown in green, and walls are shown in blue. 102

45 One of the images used as part of our experiments, as taken from the robot. 111

46 The initial state of the map, before optimization. Poses are shown in red, static landmarks are shown in green. The dark green points are laser scans, and are for visualization purposes only. 112

47 The resulting map with all measurements (including moveable objects) prior to the first weight assignment. 112

48 The resulting map after two iterations of the EM algorithm. 113

49 The resulting map after all iterations and thresholding to exclude moveable objects. Poses are shown in red, static landmarks are shown in green, and moveable landmarks are shown in blue, connected by a blue dotted line showing their movement. The dark green points are laser scans, and are for visualization purposes only. 113

50 Examples of environments with different properties. Left: an environment with a lot of visual texture. Right: an environment with little visual texture. Images are from the TUM RGB-D Dataset. 117

51 Examples of robots with different sensors, configurations, and computational abilities. Left: The Jeeves service robot platform, equipped with a laser scanner and an RGB-D sensor, and a laptop with a multi-core CPU. Right: a Turtlebot robot with an RGB-D sensor and a low-end netbook. 118

52 Examples of different map representations taken in the same environment. Left: an occupancy grid map. Right: a map that includes planar surfaces such as walls and tables. 118

53 An example of a relative pose measurements between two time stamps. Top: a representation of the graph. Bottom: a timeline representing the time intervals. 126

54 An example of a relative pose measurements between two time stamps after adding odometric information. Top: a representation of the graph. Bottom: a timeline representing the time intervals. 127

55 An example of a relative pose measurement, a landmark measurement, and an incorrectly handled relative pose measurements from odometry, because the odometric information has been double counted. Top: a representation of the graph. Bottom: a timeline representing the time intervals. 128

56 An example of a relative pose measurement, a landmark measurement, and correctly handled relative pose measurements from odometry, with no double counting. Top: a representation of the graph. Bottom: a timeline representing the time intervals. 129

57 An overview of the plugin-based architecture. 132

58 Top Left: The Jeeves service robot platform. Top Right: An example map produced by our system with this platform and configuration. Bottom: A system diagram representing the configuration and plugins used. 139

59 Top Left: An iRobot PackBot equipped with a Velodyne 32E 3D laser scanner, a MicroStrain GX2 IMU, and an onboard computer. Top Right: An example map produced by our system with this platform and configuration. Bottom: A system diagram representing the configuration and plugins used. 140

60 Top Left: A handheld RGB-D camera. The pictured IMU is not used in this work. Top Right: An example map produced by our system with this platform and configuration. Bottom: A system diagram representing the configuration and plugins used. 141

61 Left: A holonomic mobile robot equipped with laser scanners and wheel odometry. Center: A system diagram representing the configuration and plugins used. Right: An example map produced by our system with this platform and configuration. 143

62 Plots of trajectories from two SLAM approaches. 147

63 The robot platform used in this work. Point cloud data is collected with the Asus Xtion Pro camera, and laser scan data is collected with the Hokuyo UTM-30. Both of these sensors are placed on the gripper near the middle of the robot. The Asus camera is mounted upside-down on top of the gripper for more rigid and repeatable attachment. 148

64 Shown are the number of features observed per pose from each of our three data sets. From left to right, shown are results for our low-clutter office dataset, high clutter dataset, and home environment dataset. . 150

65 Shown are the number of landmarks observed by line measurements only, plane measurements only, and measured by both. From left to right, shown are results for our low-clutter office dataset, high clutter dataset, and home environment dataset. 151

66 Another view of the Robotics & Intelligent Machines lab at the Georgia Institute of Technology, where mapping was performed. 152

67 A photo of a relatively high-clutter lab in the Robotics & Intelligent Machines lab at the Georgia Institute of Technology. 152

68 A visualization of a map produced from our low-clutter office environment. Many planar features are visible, including some that have been measured only by the 2D laser scanner. Planar features are visible by the red convex hulls, and red normal vectors. The small red arrows on the ground plane show the robot's trajectory. The point clouds used to extract these measurements are shown in white, and have been rendered in the map coordinate frame by making use of the optimized poses from which they were taken. 153

69 A top-down visualization of a map produced from our low-clutter office environment. Many planar features are visible, including some that have been measured only by the 2D laser scanner. Planar features are visible by the red convex hulls, and red normal vectors. The small red arrows on the ground plane show the robot's trajectory. The point clouds used to extract these measurements are shown in white, and have been rendered in the map coordinate frame by making use of the optimized poses from which they were taken. A 1 meter square grid is displayed for scale. 154

70 A floorplan of the low-clutter area that was mapped. 155

71 A visualization of a map that includes a low coffee table, a desk, and several walls. 155

72 The Segway RMP 200 with LMS 291 laser scanner for wall measurements and a Prosilica 650c camera with a Hokuyo UTM30 laser scanner on a PTU-46-70 pan-tilt unit. Four caster wheels were added for stability. The robot is shown in a position typical of reading a door sign. 156

73 This sign is recognized and a measurement is made in the mapper. GoogleGoggles has read both the room number and the text, so this sign can be used for data association. 157

74 A close-up of the west hallway. Robot poses are shown as red arrows. Wall features are shown as red lines. Door signs are shown as pink spheres. The occupancy grid is displayed only for clarity. This figure is best viewed in color. 158

75 A map through a cluttered lab space which has few line features for measurement, resulting in significant pose error and a failure to close the loop before re-measuring a door sign. 159

76 A door sign is re-observed and the text is matched, resulting in a loop closure. 159

77 A longer mapping run through the hallways in our building. The mapping run goes through a cluttered area under construction and gets lost, resulting in a several meter trajectory error, before a door sign is re-observed. 160

78 Once a door sign is re-observed, the loop is closed and the map is corrected. 160

79 The robot has completed the long loop around the building, but has not yet found a sign match to perform a loop closure. Note that there is significant error in this map when the door signs have not been re-observed. 161

80 The robot has proceeded further along and it has just re-observed a door sign from the first time around this loop. The map is corrected and the robot now knows where it is. 161

81 The robot and a snapshot of the scene where the experiment was conducted. 162

82 Various stages of the object discovery pipeline. (a) The map of the scene and the trajectory along which the robot is tele-operated. (b) One particular frame from the trajectory showing the objects kept on the table. (c) The corresponding object segments estimated by segmentation propagation (Section 4.4.1.2). (d) The same frame with labels given to discovered objects after object refinement. Each color represents a different object. 163

83 The top four objects discovered sorted in an descending order of the number of frames it is seen in. (a) Snapshot of the discovered objects. (b) The corresponding reconstructed object models. 164

84 Precision and Recall curves of Object Discovery 164

85 Trajectories estimated with and without using objects as landmarks during loop closure. (a) Robot trajectory estimated when using no object landmarks. There is an error in the bottom right section when the robot returns to the same location. (b) Robot trajectory estimated when using object landmarks for loop closure. The object-object constraints joins the bottom right location when loop closure is detected. (c) Shows the approximate ground truth trajectory w.r.t the floor plan. 167

86 Covariance determinant of the latest pose w.r.t time when not using objects (left, corresponds to Figure 85(a)) as compared to when using objects for loop closure (right, corresponds to Figure 85(b)). There is a sudden fall in the value of covariance determinant due to a loop closure event. 167

87 Trajectories estimated with and without using objects as landmarks during loop closure. (a) Robot trajectory estimated when using no object landmarks. (b) Robot trajectory estimated when using object landmarks for loop closure. (c) Shows the approximate ground truth trajectory w.r.t the floor plan. 168

88 (Left) Our approach allows a robot to detect when there is ambiguity on the pointing gesture targets. (Right) The point cloud view from robot's perspective is shown. Both objects are identified as potential intended targets, therefore the robot decides that there is ambiguity. . 172

89 Vertical (ψ) and horizontal (θ) angles in spherical coordinates are illustrated. A potential intended target is shown as a star. The z-axis of the hand coordinate frame is defined by either the Elbow-Hand (this example) or Head-Hand ray. 176

90 Our study involved 6 users that pointed to 7 targets while being recorded using 30 frames per target. 179

91 Data capturing pipeline for error analysis. 180

92 Euclidean distance error in cartesian coordinates for each method and target. The gesture ray intersection points in centimeters with the target plane, are shown here for each target (T1-T7) as the columns. Each subject's points are shown in separate colors. There are 30 points from each subject, corresponding to the 30 frames recorded for the pointing gesture at each target. Axes are shown in centimeters. The circle drawn in the center of each plot has the same diameter (13 cm) as the physical target objects used. 181

93 Box plots of the errors in spherical coordinates θ and ψ for each pointing method. 183

94 Resulting Mahalanabis distances of pointing targets from the Object Separation Test is shown for a) Elbow-Hand and b) Head-Hand pointing methods. Intended object are shown in green and other object is in red. Solid lines show distances after correction is applied. Less Mahalanobis distance for intended object is better for reducing ambiguity. 185

95 Example scenarios from the object separation test is shown. Our experiments covered separations between 2cm (left images) and 30cm (right images). The object is comfortably distinguished for the 30cm case, whereas the intended target is ambiguous when the targets are 2cm apart. Second row shows the point cloud from Kinect's view. Green lines show the Elbow-Hand and Head-Hand directions whereas green circles show the objects that are within the threshold $D_{mah} < 2$. . . . 187

96 Top: A photo of a user pointing at a nearby table, after entering the desired label on a handheld tablet UI. Bottom: A visualization of the robot's map and sensor data. Red outlines represent the convex hulls of mapped planar features. The detected gesture can be seen via the green sphere at the user's hand, the blue sphere at the user's elbow, and the red sphere (near the "a" in Table) indicating where the pointing gesture intersects a mapped plane. 190

97 A system diagram of our approach. 193

98 Screenshots of our tablet user interface. Left: A screenshot of the interface that allows the user to command the robot to follow, or stop following them, or enter a label. Right: A screenshot of the interface for labeling landmarks. Previous labels are available for the user's convenience when labeling multiple surfaces with the same label (e.g. the four walls of a room). New labels can be entered via the onscreen keyboard. Upon pressing the "Okay" button, the user points at the landmark that should be annotated with the desired label. 197

99 A goal point corresponding to a labeled table is shown in yellow. . . . 200

100 A photo of the robot after navigating to the goal point corresponding to the table. 200

101 A goal point corresponding to a labeled hallway is shown in yellow. . 201

102 A photo of the robot after navigating to the goal point corresponding to the hallway. 202

103 Our smartphone UI for labeling objects and landmarks. 203

104 Top: A user pointing at an object. Center: A detected pointing gesture (blue and green spheres), and referenced object (red sphere). Bottom: Features are extracted from an image patch corresponding to the cluster, and annotated with the provided label. 204

105 Objects that have been previously annotated with labels can later be recognized, and referenced by the appropriate label. 205

106 The area mapped by our system with relevant locations labeled, and person measurements shown in red. 207

107 Low traffic locations selected by our system. 209

108 High traffic locations selected by our system. 210

109 A screenshot of the tablet UI. 216

110 An additional screenshot of the tablet UI. 217

SUMMARY

Although much progress has been made in the field of robotic mapping, many challenges remain including: efficient semantic segmentation using RGB-D sensors, map representations that include complex features (structures and objects), and interfaces for interactive annotation of maps.

This thesis addresses how prior knowledge of semi-structured human environments can be leveraged to improve segmentation, mapping, and semantic annotation of maps. We present an organized connected component approach for segmenting RGB-D data into planes and clusters. These segments serve as input to our mapping approach that utilizes them as planar landmarks and object landmarks for Simultaneous Localization and Mapping (SLAM), providing necessary information for service robot tasks and improving data association and loop closure. These features are meaningful to humans, enabling annotation of mapped features to establish common ground and simplifying tasking. A modular, open-source software framework, the OmniMapper, is also presented that allows a number of different sensors and features to be combined to generate a combined map representation, and enabling easy addition of new feature types.

CHAPTER I

INTRODUCTION

1.1 Service Robots

One of the foci of recent robotics research has been to create a home service robot capable of assisting with everyday tasks. There have been many examples of this dream in science fiction, such as “Rosie the Robot” from “The Jetsons”, or “C-3PO” from “Star Wars”. For example, Rosie was able to do household chores such as dishes, laundry, and cleaning tasks. Modern robots cannot perform most of these tasks robustly and consistently, but there has been much progress in developing some of the required competencies. Robots will need robust and general skills in perception, manipulation, planning, communication, and mapping. This thesis will be focused on developing a mapping subsystem that will get us closer to this type of application.

Service robots are of particular interest to several populations. Service robots with even basic object manipulation and navigation capabilities can be of assistance to persons with motor disabilities to perform object manipulation tasks. Additionally, service robots may also enable elderly people to live independently and stay in their homes longer by assisting with household tasks. In addition to these populations, service robots can help us all by performing household tasks and giving us additional time for work or leisure.

1.2 Localization and Mapping

Some of the key competencies that such a robot will need are the ability to understand the environment it is in, keep track of its position in the environment, and avoid obstacles. Doing this requires some kind of map, but providing a prior maps is unrealistic for many environments, so an online mapping system is also required. This means that we will need to perform simultaneous localization and mapping (SLAM), which is the focus of this thesis. Robot mapping has been studied for many years [41, 8], and much progress has been made. Maps enable robots to localize themselves, plan paths and navigate, and avoid obstacles. Much of the mapping work has focused on designing maps that will allow robots to navigate robustly, but as service robots become capable of more complex tasks, and as robots need to interact with humans more, robotic mapping systems will need to become more capable in order to support these tasks. For example, service robots intended to guide a user to a requested destination require labeled places and regions, as in [147]. A mobile manipulator capable of performing fetch & carry tasks may require information about detect objects to be stored in the map. Object search tasks may benefit from maps that contain locations of tables, shelves, and counters, where objects might be likely to be found.

1.3 Semantic Mapping

Adding semantic information to mapping systems is useful for several reasons. Service robots will operate in human environments, and will need to be able to accept commands from human users. Requiring users to understand the robot's map coordinate frame to enable it to accept metric commands may not be the most intuitive user interface. A map for service robots should enable communication between the human and the robot, enabling dialogs to reference spatial elements including specific landmarks, e.g. "the kitchen table", or regions, e.g. "the kitchen".

Map representations should also include spatial information to make it easy for a robot to plan and execute its tasks. If the task is more complex than obstacle avoidance, the map should include more than just occupancy information. For example, if we will need to interact with landmarks, objects, or people, a map should include information about these.

For these reasons, adding semantic information to maps will be an important step forward. To further motivate this, let us consider how semantically annotated maps can help us as humans to go about our tasks. Consider a person who has traveled to a conference, and is given a map of the surrounding area. Figure 1 shows a map that does not include semantic information. We can make out which areas are traversable (streets, sidewalks), and which are occupied by buildings, etc, so this is similar to an occupancy based map. If we're given two metric coordinates on this map, we could plan a path and navigate between these points. However, if a colleague asks us to meet at a nearby restaurant or landmark, we're out of luck. To handle such tasks, we will need a richer map representation.

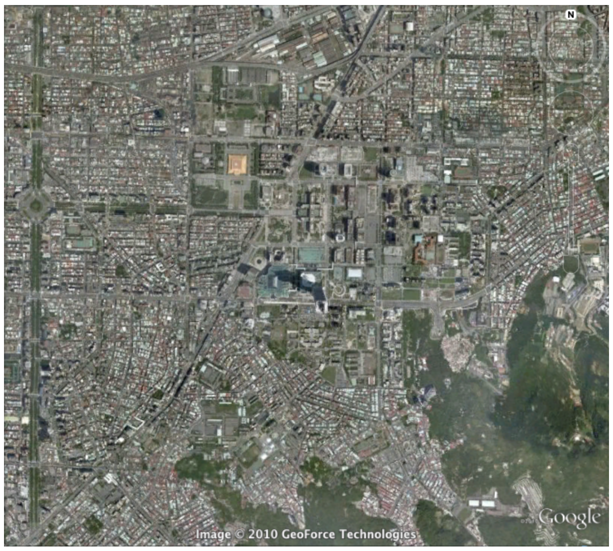


Figure 1: An example of a map without semantic information.

Now consider a map such as the one shown in Figure 2. We now have a map with some labeled landmarks (roads, allow us to find points of interest, and plan paths more easily. This type of semantic information can be helpful, but we can do better still.

Finally, consider a map such as the one shown in Figure 3. In addition to having semantic labels provided for various landmarks and areas, we also have points of

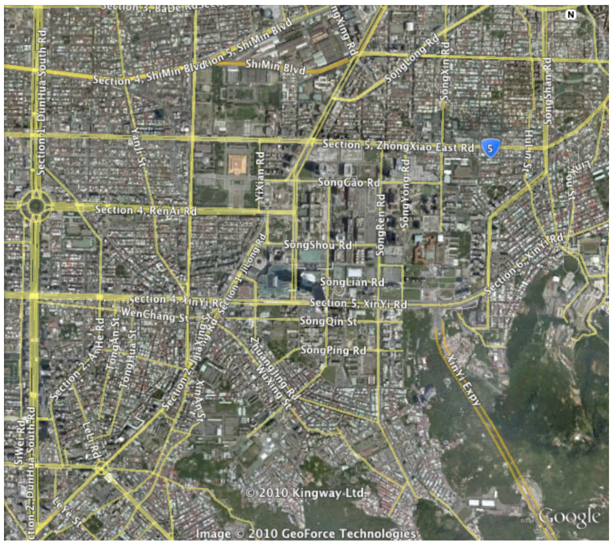


Figure 2: An example of a map with semantic labels.

interest in several categories. We know that if we are hungry, we can obtain food from a restaurant, if we need laundry done, we can go to a laundromat, and if we need to get groceries, we can go to a grocery store.

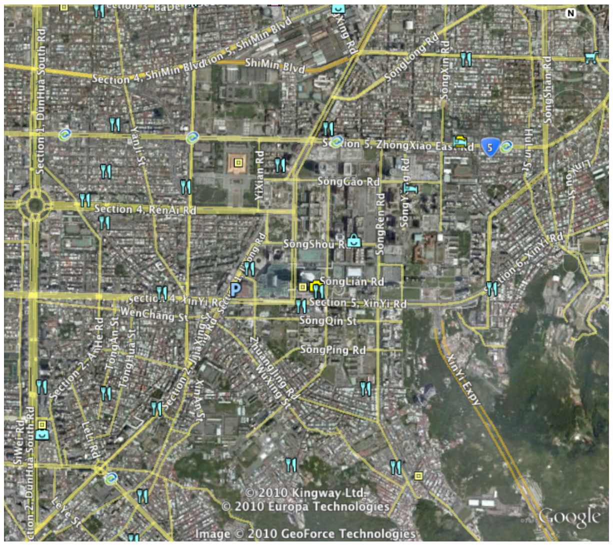


Figure 3: An example of a map with semantic labels, and additional semantic information.

In this work, we will take steps towards making maps that contain information for service robots to support higher level tasks than just navigation. We will create a mapping system that supports labeled structures, objects, and regions, so that a service robot can use it plan and perform tasks such as fetch & carry and object search, in addition to navigation and obstacle avoidance.

Much as humans make use of maps to understand, communicate about, and reason about spatial information, robot mapping systems serve a similar purpose. Humans use a variety of types of maps for different applications. Road maps are used to compute routes between points, suitable for driving. Topographic maps include height information via contours, for when 3D information is required. Mapping tools such as Google Earth allow maps to be annotated with many different types of information, depending on the needs of the user. The space of maps for robots is currently much less varied, but we'll need to expand our concept of "robot maps" as service robots expand their capabilities.

1.3.1 What are maps for?

One of the key questions to consider in the design of a mapping system is the type of map representation to be used. This is necessarily tied to what the map will be used for.

Traditional robot maps have focused on recording occupancy information, for the purposes of localizing a mobile robot, as well as for motion planning and obstacle avoidance. However, as robots become more capable, the map representation may need to include additional types of information. In this chapter, we will discuss several types of information that can be useful for service robot mapping.

What types of questions might we want to ask our map? Some of these include:

- At what coordinate am I currently located?
- How can I get from point A to point B?

These queries enable basic localization and navigation tasks. However, as robots start to perform more complex tasks, they may require some additional spatial information.

- What is the name of the place I'm in?
- What objects are around me?
- Where can I find person X?

In this work, we aim to build maps that are suitable for both localization and navigation tasks, as well as higher level service robotic tasks that require various types of semantic information.

Information is stored in robot maps for two main reasons: for the robot to use to localize and map, and additional spatial information required for tasks. Note that these are not necessarily mutually exclusive – in Chapter 4 and Chapter 5 we will demonstrate maps made of structures including walls and tables, and objects such as door signs or tabletop objects, and show that these are useful both for SLAM and other service robotic tasks.

1.4 Challenges

In recent years, much progress has been made in mapping, 3D perception, object recognition, and human-robot interaction, which all contribute towards making better service robots. However, many challenging areas still remain. We highlight four challenges that we address in this thesis: efficient semantic segmentation using RGB-D sensors, map representations that include complex features (structures and objects), flexible multi-modal mapping approaches, and interfaces for interactive annotation of maps to establish common ground.

1.4.1 Efficient Semantic Segmentation

3D perception techniques based on point clouds have greatly improve service robots' ability to segment scenes and detect objects. However, techniques designed for unorganized 3D point clouds as produced by laser scanners can be computationally expensive, leading to long execution times or requiring input data to be down-sampled.

RGB-D sensors such as the Microsoft Kinect provide 3D data at VGA (640x480) resolution at 30Hz, which has been a challenge for existing approaches. We address this in Chapter 3, introducing a technique capable of planar segmentation and object detection in real-time for data produced from these sensors.

1.4.2 Complex Features for SLAM

Most traditional mapping techniques have focused on either raw sensor data, such as laser scan matching, or on low level features, such as corner features commonly used in monocular SLAM. While most human environments such as homes and offices are not as structured as factories and manufacturing settings, they still include a significant amount of high level structure that we can utilize as features in SLAM. Such environments tend to have large flat floors, with spaces partitioned into rooms and corridors by walls. Rooms tend to include counters, shelves, tables, while objects tend to rest on these horizontal planar surfaces. We consider these to be "semi-structured" environments. Leveraging our prior knowledge of this structure has been a challenge for existing SLAM techniques. In addition to being useful for localization, such features are semantically meaningful to humans, as they correspond to discrete structures and objects recognizable by humans, in contrast to low-level features like corners or points. This can facilitate communication with humans, as will be shown. Chapter 4 describes contributions in this area, including approaches for using planar surfaces and objects as landmarks.

1.4.3 Multi-modal Mapping

Service robots require a sufficiently descriptive map representation that includes the information required to complete tasks, while still being suitable for localization, and being computationally and memory efficient. Most current mapping systems are designed for a specific application with a fixed sensor suite, and modifying them for new applications has been a challenge. Adding new measurement types and combining them with existing approaches has also been a challenge. We address this in Chapter 5 by introducing a software framework for multi-modal mapping, aiming to reduce the development effort both for configuring mappers for new situations, and also for extending and combining existing approaches.

1.4.4 Semantic Labeling

Another important challenge in robotic mapping is how to annotate maps with semantic information and labels. To enable communication between humans and robots, it is important to have a map representation that allows labels and terms to be grounded appropriately.

Many tasks may require semantic information. For example, let us consider the task "fetch the mug from the kitchen table". From a mapping perspective, our map representation must tell us which portion of the map is considered "kitchen table". In order to support this, we will require our mapping system to include information about both the location and extent of such landmarks. In addition, we will need an intuitive user interface to allow such labels to be provided by a user. Annotation of regions that correspond to rooms and corridors, and also locations within these have been addressed previously in [147], we extend this to include structures and objects. Chapter 6 describes our approach to addressing this.

1.5 Contributions

In addressing the above challenges, the primary contributions of this thesis are as follows:

- An approach to planar estimation and clustering for RGB-D point clouds that leverages the organized structure, eliminating the need to subsample VGA data to achieve real-time operation.

- Complex feature types for SLAM including planes, lines, objects, and additional constraints between these features that leverage our prior knowledge of semi-structured human environments to improve data association, loop closure, and provide necessary information for service robotic tasks.

- A modular software front-end that allows utilization of the above mentioned features and/or traditional 2D and 3D scan matching approaches in any combination, and enables plug-and-play integration of new sensors and features.

- An approach to interactively annotating maps and object models by combining deictic gestures and a tablet UI, to provide common ground for tasking of service robots in missions such as fetch & carry.

1.6 Thesis Statement

The key contribution is the following thesis statement:

Utilizing prior knowledge of semi-structured environments enables semantic feature representations for SLAM and improves data association and loop closure. Such features are meaningful to humans to establish common ground and simplify tasking.

1.7 Outline

Chapter 2 gives a high level overview of related work, while more specifically related work is highlighted throughout. Chapter 3 discusses perception techniques for segmentation, feature extraction, and object recognition as applicable to mapping. Chapter 4 describes our approaches to using complex features such as planes and objects in SLAM, while Chapter 5 describes OmniMapper, our multi-modal mapping framework and the results generated with it. Human-Robot Interaction for map annotation is described in Chapter 6, followed by a conclusion and future work in Chapter 7.

A note on the ordering of things: much of the work in this document is not presented in chronological order, but instead is presented in a bottom-up fashion.

Many of the lowest level contributions, such as the segmentation approach in Section 3.4 and gesture recognition in Section 6.1 were developed towards the end of the PhD, due to shortcomings I saw in existing approaches while performing the earlier (but higher level) work. Please keep this in mind while reading. The bulk of the work has been published in conference or workshop proceedings which are cited at the start of each section, from which the chronological development order is apparent.

CHAPTER II

RELATED WORK

Research pertaining to service robots has been ongoing for several years, and a large body of work exists that is related to this thesis. In this chapter, we will provide a brief survey of some related work in the areas of simultaneous localization and mapping (SLAM), semantic mapping, Human Augmented Mapping (HAM), and Object Recognition and Segmentation for Mapping.

2.1 Simultaneous Localization and Mapping (SLAM)

The Simultaneous Localization and Mapping (SLAM) problem was first proposed by Smith and Cheeseman, who used an Extended Kalman Filter (EKF) on landmark positions and the robot position, in [132]. SLAM has since become an important area of research for mobile robotics. A detailed overview of SLAM's development is given by Durrant-Whyte and Bailey in [41] and [8]. A treatment of SLAM is also given in a textbook on Probabilistic Robotics by Thrun, Burgard, and Fox [140].

Many modern SLAM techniques eschew the EKF formulation in favor of graph based representations. Instead of filtering and solving for only the current robot pose, these techniques typically solve the *full SLAM* problem and maintain a graph of the entire robot trajectory in addition to the landmark positions. This has the advantage of resulting in a more sparse representation which can be solved efficiently. Some examples of this type of approach involve Folkesson and Christensen's GraphSLAM [47], and Dellaert's Square Root Smoothing and Mapping (SAM) [33]. SAM has been extended to allow incremental updates for improved online operation in [70, 71]. We make use of the GTSAM library [34] based on these techniques as our optimization engine. Other modern graph optimization techniques of note include Toro [51] and

g^2o [88].

Another related area of research is *feature-based* SLAM techniques, which use landmarks to solve the SLAM problem, in contrast to techniques which create pose-graphs. Some examples of frameworks for handling this type of technique include: the M-Space model [48] and the SP-Model [18].

There has also been some previous work on SLAM using planar features for SLAM. In [124], Rusu *et.al.* found candidate objects from 3D point clouds by first segmenting horizontal surfaces and then finding clusters which are supported by the horizontal surfaces. This technique can be used to find multiple planes, and objects can be extracted which are supported by each of these planar surfaces. Semantic labeling of planar surfaces was also presented in [125]. These papers showed that planar surfaces can be extracted from point clouds very effectively, and objects can be found which are supported by these planar surfaces.

Another approach to finding horizontal surfaces such as tables was investigated previously by Donsung and Nevatia [79]. The method presented in this work measured the relative pose of tables or desks with an edge-based computer vision approach. This technique was used for finding the relative pose of these surfaces from the robot's current pose, but it was not used for building large-scale maps of surfaces.

A related planar SLAM approach is by Weingarten [160]. This work involves using planar features extracted from rotating laser-range finder data as features in an EKF-based SLAM framework, making use of the SPmodel. The features are represented by their normals, and bounded by an alpha shape to represent the extent of the feature, which can be extended incrementally as new portions are observed. We will use a similar feature type in our work, but we employ a graph-based approach to SLAM instead, which allows us to solve for the full robot trajectory rather than only the most recent pose. We have found this to be important for accurately mapping the extent of planes, as errors in past poses need to be accounted for and corrected in order to accurately map planar extents.

Other previous approaches to plane mapping include Pathak *et.al.* [108][107] have removed the dependency on ICP and focus on extracted planar features. ICP based techniques are susceptible to local minima particularly when there is insufficient overlap between point clouds. Extracted planar features also exhibit significant compression and computational advantages when compared to full point clouds. In [108], planar features are extracted at each robot pose. These planar features are matched against the features which were seen in prior poses to find correspondences. This technique does not make use of any odometry; therefore, the authors have developed a technique which enables them to sequentially associate planes which is more efficient than typical RANSAC techniques. The authors then compute the least squares rotation and translation which brings the associated planes into alignment. The rotation and translation are used to build an *pose graph* which is optimized. As opposed to building a pose graph, we will take the alternative approach of maintaining the planar features as landmarks in the optimization problem, along with odometry.

2.2 Semantic Mapping

Semantic mapping aims to build richer, more useful maps that include semantic information. Kuipers [87] proposed the *Spatial Semantic Hierarchy* (SSH), which is a qualitative and quantitative model of knowledge of large-scale space consisting of multiple interacting representations. This map also informs the robot of the control strategy that should be used to traverse between locations in the map. This representation is based on the relationship of objects, actions and the dependencies from the environment. More recently, Beeson *et al.* [9] provided a more specific framework representation of metric spatial knowledge in local small scale space. This framework is focused on the robot's sensory horizon e.g.(global and local symbolic, and metrical reasoning of the space), but also human interaction.

Martínez-Mozos and Rottmann [98] [121] introduce a semantic understanding of the environment creating a conceptual representation referring to functional properties of typical indoor environments. A method for representing room shapes based on laser scan data was presented in [85]. Pronobis has also extensively studied place categorization in the context of semantic mapping [113]. Both visual and laser based features are considered, and used to classify spaces. Detailed experiments evaluating the performance of the semantic mapping and place categorization system are presented in [112].

Ekvall *et. al* [42] integrated an augmented SLAM map with information based on object recognition, providing a richer representation of the environment in a service robot scenario.

Nüchter *et.al* investigated semantic labeling of points in 3D point cloud based maps in [105] and [104]. These papers demonstrated a SLAM system for building point cloud based maps based upon Iterative Closest Point (ICP)[11]. Semantic interpretation was given to the resulting maps by labeling points or extracted planes with labels such as floor, wall, ceiling, or door. This technique was applied in outdoor as well as indoor environments.

One of the key concepts in semantic mapping is that of "grounding", or establishing "common ground" [26]. Of particular interest for mapping is grounding references, in order to ensure that the human and robot have common ground when referring to regions of a map, structures, or objects. Many spatial tasks may require various terms to be grounded in the map.

One application of semantic maps is for directions following tasks, as have been studied by Kollar et. al [81]. This work describes a system capable of following directions given in natural language, i.e. "go past the computers". The system made use of appearance-based landmarks represented by points, and could employ spatial relations like "go past", "go to", and "go through", allowing directions to be followed relative to landmarks.

Semantic maps can also be used for other language based tasks, such as understanding commands like “get the mug on the table”. The robot can complete such tasks only if it knows the location and extent of the table, can recognize mugs, and understands the meaning of “on”. Concepts of “on” and “in” have been studied by Aydemir et. al in [6].

2.3 Human Augmented Mapping

Another related body of work is on Human Augmented Mapping, introduced by Topp and Christensen [146] and [145], where a human assists the robot in the map building process. This is motivated by the scenario of a human guiding a service robot on a tour of an indoor environment, and adding relevant semantic information to the map throughout the tour, for later reference. Users could ask the robot to follow them through out the environment and provide labels for locations, which could later be referenced in commands such as “go to label”. This means of providing labels seems quite intuitive, as users are co-located in the environment with the robot platform.

Dialog in human augmented mapping has also been investigated in [86]. Clarification dialogs were studied in order to resolve ambiguities in the mapping process, for example, resolving whether or not a door is present in a particular location.

This was applied to the Cosy Explorer system, described in [166], which includes a semantic mapping system that multi-layered maps, including a metric feature based map, a topological map, as well as detected objects.

2.4 Object Detection & Object Recognition

As we will be dealing with objects, the body of work on detecting and recognizing objects is related. Of particular interest is work on detecting objects in indoor environments, as well as online object modeling, and object recognition. There is a great deal of other related work in the fields of object recognition and environment segmentation, but a more detailed survey is beyond the scope of this document. Instead, several particularly relevant works will be highlighted.

In [58] Herbst *et. al* detect objects in the environment that have moved since a prior visit to a location. This is done by essentially differencing a current and prior map of a location, and detecting differences. These differences are segmented out and detected as objects.

Online modeling of objects has also been investigated in [84], in which objects can be grasped, manipulated in front of an RGB-D sensor to get multiple views, and modeled for later recognition. Objects can also be re-grasped in order to observe and model portions that were occluded by the robot's manipulator in previous views.

CHAPTER III

3D PERCEPTION: SEMANTIC SEGMENTATION & FEATURE DETECTION

Mapping systems require the ability to sense and measure the environment to create maps. Some approaches operate on raw sensor data, such as alignment of laser scans or point clouds. However, a key component of our approach to mapping is the use of higher level features than raw sensor data, by mapping at the level of structures and objects. This allows our approach to take advantage of some semantic knowledge about the environment, and build richer maps that enable a wider range of tasks. One approach to this is to process individual sensor frames, such as point clouds or laser scans, to try to detect structures and objects within them.

Segmentation is an important step in many perception tasks, such as many approaches to object detection and recognition. Segmentation of images has been widely studied in the computer vision community, and point cloud segmentation has also been of interest. Segmentation and feature detection in 2D laser scans has also been studied, for detection of features such as walls (as we will describe in Section 3.2) and doors. Planar segmentation of point cloud data has been of particular interest, as this can be helpful for a variety of tasks. We will describe a similar technique extended to segment all planes in the scene rather than a single dominant plane in Section 3.3. We then describe a much faster connected-component based approach to multi-plane segmentation and clustering in Section 3.4. Chapter 4 describes how these segmented planes and objects can be utilized for the SLAM problem, and Chapter 6 describes how to annotate such features with labels, and use them to achieve service robotic tasks such as object retrieval.

This chapter describes several techniques used by our mapping system, including 3D edge detection, semantic segmentation, and object discovery, modeling, and recognition. Note that many of these topics are large and active subfields of computational perception, so a full review is not provided, only the specific approaches used in this work are described.

3.1 3D Edge Detection

2D edges such as Canny edges have been popular in computer vision and robotic perception for object recognition, pose estimation, and tracking. A key advantage of edge features over corners or point features is that they are more robust for textureless scenes and objects. In this section, we will extend this idea to cover 3D edges, as detected in RGB-D point cloud data, as originally described in [23]. Chapter 4 provides details on how such features can be used in SLAM, and results on a benchmark dataset comparing to other techniques.

Our approach to 3D edge detection is similar in principle to the organized connected component segmentation algorithm described in Section 3.4.2, in that we exploit the organized structure of the point cloud / image for efficient processing. We further make a distinction as to the types of edges found in the scene, by classifying them into several categories based on how they were detected. Both geometric information and photometric texture information are used. Our implementation is freely available under the BSD license as part of the Point Cloud Library [122].

3.1.1 Occluding, Occluded, & Boundary Edges

Several types of edges can be detected in RGB-D data. Abrupt changes in depth between neighboring points (depth discontinuities) in the 3D data provide both *occluding edges* and *occluded edges.* Occluding and occluded edges occur in pairs, with occluding edges occuring on foreground objects, thus closer to the sensor. Occluded edges are the corresponding edge on the other side of the depth discontinuity, on

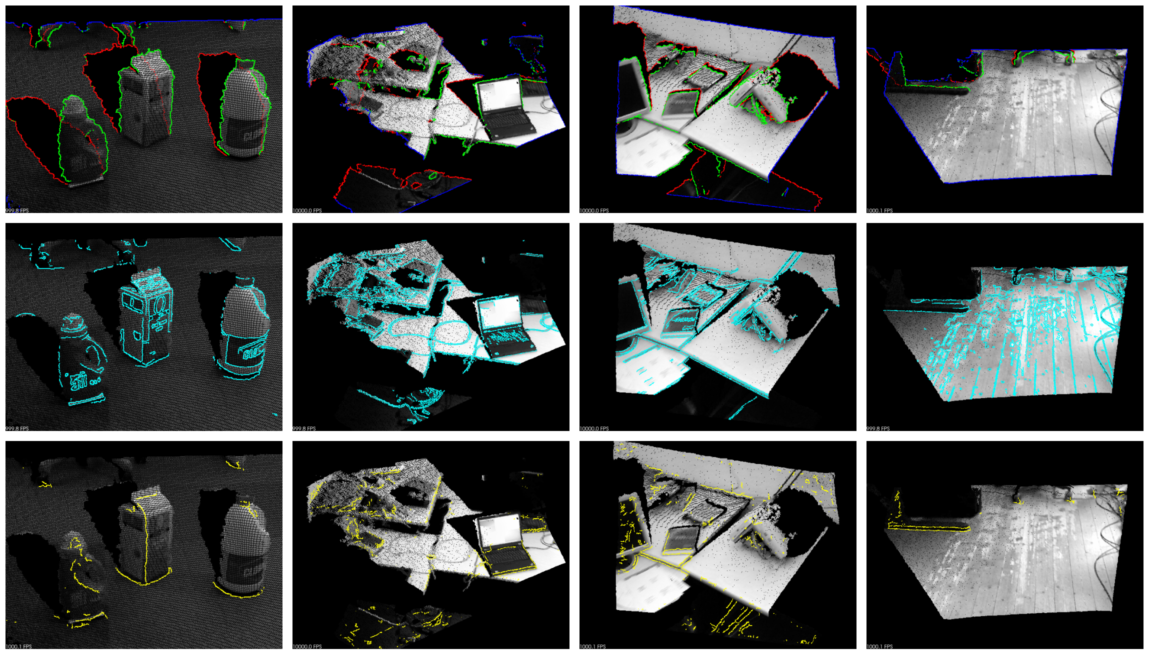


Figure 4: Example edges in several scenes. Top Row: *occluding edges* are shown in green, *occluded edges* are shown in red, and *boundary edges* are shown in blue. Center Row: *RGB* edges are shown in cyan. Bottom Row: *high curvature* edges are shown in yellow. Figure best viewed in color.

the surface being occluded by the foreground object. Examples of such edges can be found in the first row of Figure 4.

Occluding and occluded edges are detected based on euclidean distance to neighboring points in an 8-connected sense, which we denote as $\texttt{8} - \texttt{Neighbor}(\mathbf{D}, x, y)$. If the maximum distance $\widehat{\mathbf{d}}$ exceeds a threshold $\tau_{dd} \cdot \mathbf{D}(x, y)$, this point is considered as a candidate edge. Positive distances correspond to occluded edges, while negative distances correspond to occluding edges. Note that these edges are similar to the obstacle and shadow borders described in [134], but are computed in a way that leverages organization of RGB-D data. Note also that the "veil points" described in [134] are common in point clouds generated by laser scanners, but are not present in Kinect-like RGB-D sensor data.

Missing data due to invalid depth measurements is common in RGB-D data, and must be handled by the algorithm. These points are typically denoted as "NaN" (not a number) in the depth data. One common cause is when the normal of the surface is nearly orthogonal to the optical axis of the sensor. Such invalid measurements frequently appear on object boundaries, making this especially important to handle. The approach handles this by searching in image space across invalid points, to determine the nearest valid neighbor in a search direction $\texttt{SearchDirection}(\mathbf{N}, x, y)$. If a corresponding point can be found, each point is classified as occluding or occluded edge as described above. If no correspondence is found, the point is considered as a *boundary* edge, denoting a point that lies on the boundary of the RGB-D data (though not necessarily on an image boundary, as invalid depth measurements are often present for many points on the image boundaries).

3.1.2 High Curvature Edges

Densely computing surface normals also enables detection of edges based on the curvature of the sensed surface. If the surface normals between neighboring points are sufficiently different, we can consider this as a *high curvature* edge. Two types of such edges can occur: "convex" or "ridge" high curvature edges correspond to areas of high curvature where surface normals face away from each other, while "concave" or "valley" high curvature edges correspond to edges where neighboring normals face toward each other. Several approaches for detecting such edges have been proposed by the computer graphics community [106, 52, 109] in point clouds or meshes. However, these do not account for the noise or missing measurements present in RGB-D data, and can be computationally expensive, making them unsuitable for this application.

The Canny edge detector computes image gradients $\mathbf{G_x}$ and $\mathbf{G_y}$ using a Sobel operator. In computer vision applications, this is performed on a grayscale image. However, for detection of high curvature edges, we perform this on the surface normal image. Given densely computed normals $\mathbf{N}$ from depth image $\mathbf{D}$, the Sobel operator can be applied to detect responses to changes in surface normal for directions x and

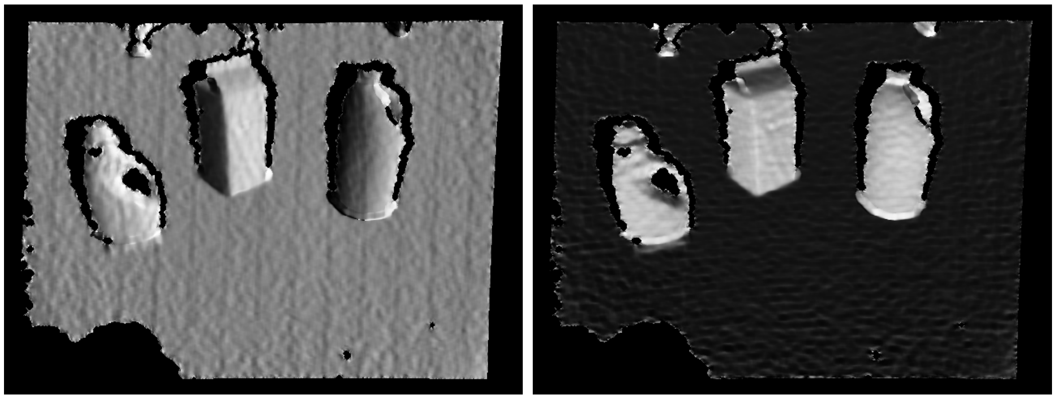


Figure 5: Surface normal image response to Sobel operator in x (left) and y (right).

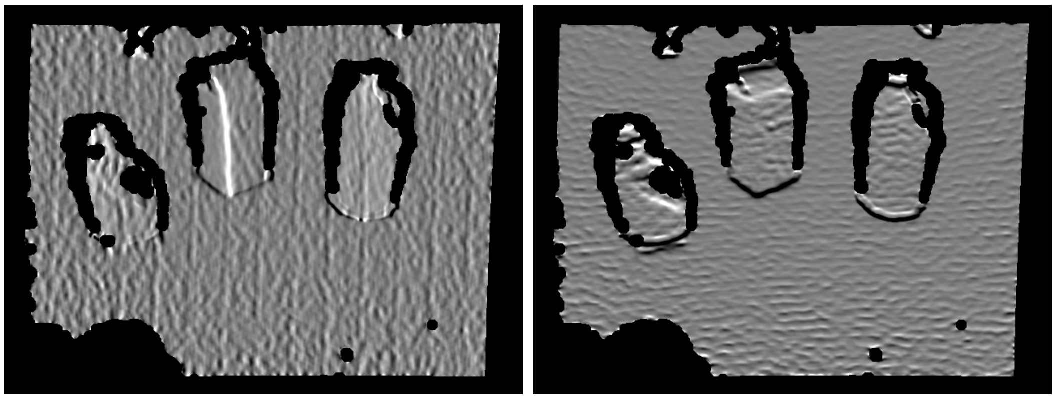


Figure 6: Surface normal gradients $\mathbf{G}_x$ (left) and $\mathbf{G}_y$ (right).

y: $\mathbf{N}_x$ and $\mathbf{N}_y$. The results of this step are shown in Figure 5. Normal gradients $\mathbf{G}_x$ and $\mathbf{G}_y$ are shown in Figure 6. As in traditional Canny edge detection, non-maximum suppression and thresholding are then applied to produce only strong and well-connected edges. The resulting high curvature edges are shown in Figure 7.

3.1.3 RGB Edges

The well known Canny edge detector [17] is applicable to images, such as the RGB channels of a sensor frame from an RGB-D sensor. As the RGB and depth data are aligned, it is straightforward to apply the Canny edge detector to the image, and back-project to obtain depth, resulting in 3D edges corresponding to photometric texture. Examples of such edges are given in the middle row of Figure 4.

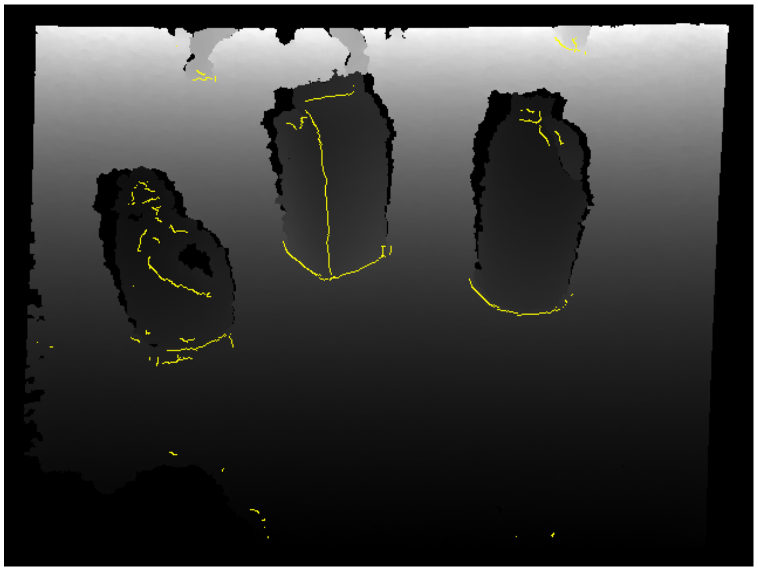


Figure 7: Detected high curvature edges, shown in yellow.

3.1.4 Edge Detection Runtime

Many robotics and mapping applications require near real-time performance, so we evaluated the running time of this approach. The Freiburg 1 RGB-D SLAM benchmark dataset [136] includes nine trajectories in indoor environments. The mean and standard deviation computation times for these trajectories are given in Table 1. This experiment used the OpenCV implementation [15] of Canny rather than the PCL implementation, as this was found to be more efficient. The experiments were performed on a laptop computer with an Intel Core i7 CPU and 8GB of memory.

As can be seen in the table, occluding edges and RGB edges are very efficient to compute, and are suitable for near real-time usage. High curvature edges require surface normal estimation, which increases their runtime. However, if latency is not important but throughput is, these steps could be implemented as a parallel pipeline, where surface normals are extracted for frame n in parallel with high curvature edge detection for frame $n-1$ (where normals would have been extracted in the previous step).

Table 1: Average computation time of edges in Freiburg 1 sequences

Sequence	Occluding edges[†]	RGB edges	HC edges
FR1 360	23.66 ± 1.03 ms	12.05 ± 0.99 ms	101.24 ± 5.28 ms
FR1 desk	24.06 ± 1.22 ms	13.04 ± 0.93 ms	96.24 ± 4.97 ms
FR1 desk2	24.71 ± 0.79 ms	12.76 ± 0.89 ms	99.55 ± 5.22 ms
FR1 floor	24.08 ± 1.87 ms	12.04 ± 0.94 ms	102.09 ± 5.03 ms
FR1 plant	24.61 ± 1.71 ms	13.04 ± 1.13 ms	99.25 ± 6.89 ms
FR1 room	23.86 ± 1.47 ms	13.07 ± 1.35 ms	100.91 ± 9.27 ms
FR1 rpy	23.89 ± 0.99 ms	12.87 ± 0.73 ms	98.33 ± 2.61 ms
FR1 teddy	25.20 ± 2.57 ms	13.14 ± 1.22 ms	103.70 ± 7.15 ms
FR1 xyz	24.45 ± 1.36 ms	13.27 ± 0.87 ms	98.85 ± 5.31 ms

[†] Please note that the computation time for occluding edges includes the time taken for both occluded and boundary edges as well, since our edge detection algorithm detects these three edges at the same time.

3.2 Line Segmentation of Laser Scans

While points from 2D laser scans are often used directly via the Iterative Closest Point (ICP) algorithm, such scans can also be processed for use in feature-based SLAM. A useful type of feature for this is 2D line segments. Many approaches exist to line segmentation in 2D laser scans; a survey of several methods is provided by Nguyen *et.al* [102]. Given the results of the survey and comparison in [102], we chose to segment lines from laser scans using an iterative RANdom SAmple Consensus (RANSAC) based approach [46], as it is a good balance between efficient runtime and accurate results. A brief description of this approach follows.

To fit one line to a laser scan, two points are uniformly selected from the laser scan. These two points define a line, and point-line-distance can be computed for each remaining point in the laser scan. Points that are within a certain distance threshold from this line are considered as inliers, and a least-squares line fit is performed, resulting in updated line parameters. The process is repeated for a set number of iterations, and the line with the largest count of inliers is considered to be the best fit line. The line coefficients are then re-estimated in a least-squares sense using all inliers.

To handle co-linear points that are spatially separated (such as wall segments on either side of a doorway), we additionally require that points in a line segment not be too far away from their neighbors. If a gap larger than a certain threshold is detected, the line segment is split into two segments.

Once a single line has been detected, its inliers are removed from the scan, and the algorithm repeats. The process repeats until the largest detected line segment has too few inliers to be considered.

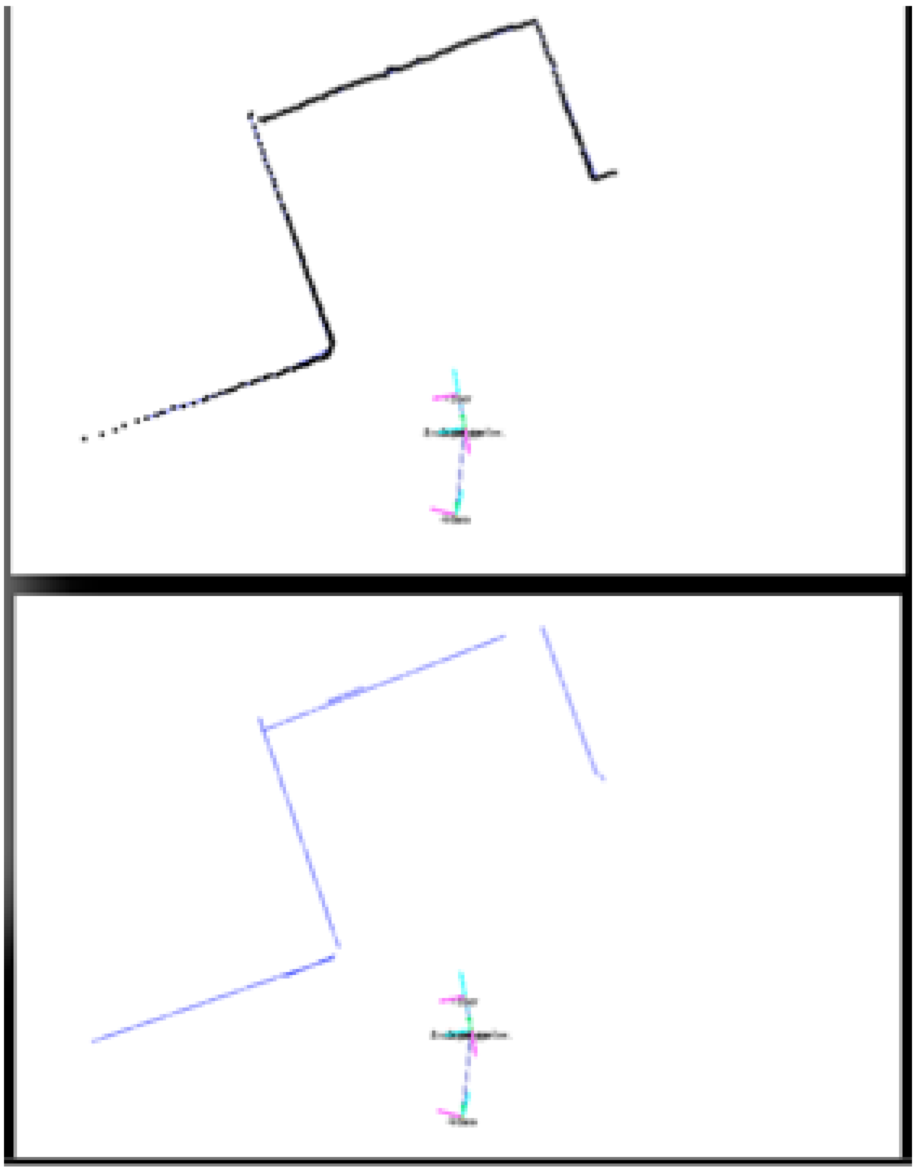


Figure 8: An example of 2d line segments extracted from a laser scan. Top: raw laser scan points. Bottom: detected line segments. Note that many walls are not perfectly flat, especially at edges, so some portions of walls may be excluded.

3.3 Unorganized Point Cloud Segmentation

Several approaches exist for planar segmentation of 3D point clouds. One of the simplest and most popular is based on the well known RANdom SAmple Consensus (RANSAC) algorithm for model fitting [46]. A RANSAC based approach to planar segmentation, and detection of tabletop objects by extracting contiguous clusters of points above planar surfaces was proposed by Rusu *et. al* [124]. We describe a similar method for segmenting multiple planes from the scene.

We use an iterative RANSAC to find planes in the scene, returning the plane with the most inliers from the point cloud. We remove all inliers for this plane from our point cloud, and then perform RANSAC again to find the next largest plane. The process terminates when no plane with a sufficient number of points can be found. For each detected plane, we perform clustering on the inliers to find contiguous regions of points within our plane, discarding clusters that are too small. This clustering step serves two purposes: to remove individual points or small clusters of points that fit to the plane but aren't part of a large contiguous surface, and to separate multiple surfaces that are coplanar but are in different locations, such as two tabletops at the same height. Each cluster with a sufficient number of points is saved and will be used for mapping purposes. Organized point clouds are not required for this algorithm, so point cloud data can be generated either by range camera sensors, as is used in most of this work, or by tilting laser scanners, as shown in Figure 9.

Our implementation of this apporach makes use of many of the tools available in the Point Cloud Library (PCL) developed by Rusu and others, which includes a variety of tools for working with 3D point cloud data including RANSAC plane fitting, outlier removal, and euclidean clustering methods. PCL is an open source library with ROS integration, and is freely available from the pointclouds.org website.

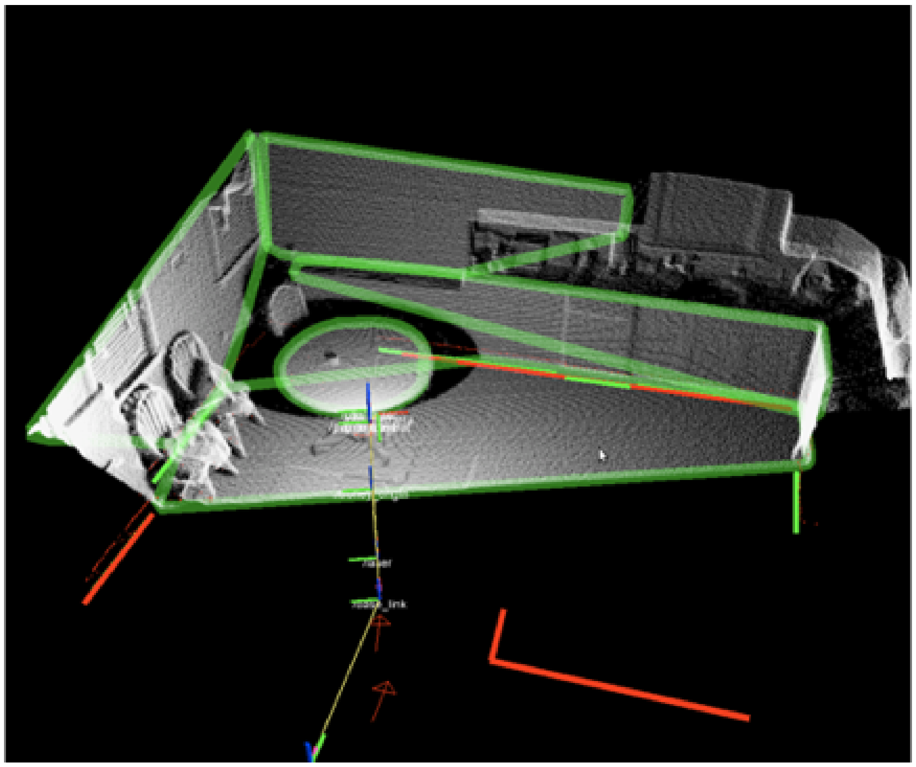


Figure 9: An example scene segmented using this RANSAC-based approach. Segmented planes are outlined in green.

3.3.1 Discussion

While this approach can typically detect most planes in the scene, it can be computationally expensive, as a large number of hypotheses must be iteratively evaluated. A comparison of the runtime of this approach to our RGB-D segmentation approach is provided in Section 3.4.9.2.

It can also be difficult to determine accurate boundaries for planar segments, which may be important to some mapping tasks. As can be seen in Figure 9, the boundaries shown in green do not match the segmented cabinet shapes, since a convex hull is used. While concave hull / alpha shape approaches do exist for finding boundaries in unorganized point clouds, these require setting parameters appropriately, and can be computationally expensive. Section 3.4 describes an algorithm for segmenting organized point clouds that addresses both of the above limitations.

3.4 Organized Connected Component Segmentation

Segmentation is an important step in many perception tasks, such as some approaches to object detection and recognition. Segmentation of images has been widely studied in the computer vision community, and point cloud segmentation has also been of interest. Planar segmentation of point cloud data has been of particular interest, as this can be helpful for a variety of tasks. Detection of tabletop objects by extracting contiguous clusters of points above planar surfaces such as tables was proposed by Rusu *et. al* [124]. Segmented planar surfaces have also been used as landmarks for feature-based SLAM [149] and semantic mapping applications [153].

In this Section, we describe an efficient method for segmenting organized point cloud data. This algorithm was originally published in [148]. The image-like structure of organized point clouds enables us to apply approaches from computer vision, including graph-based or connected-component segmentation approaches. However, in addition to RGB data as would be available in an image, we also make use of the 3D coordinates of each pixel / point, as well as surface normal information.

3.4.1 Related Work

Segmentation of images, range images, and point clouds have all been widely studied in the literature. Several of the most closely related works will be highlighted here.

Felzenszwalb and Huttenlocher [45] proposed a graph-based approach to image segmentation that shares some similarities with our approach. A graph structure is imposed on the image, such as a 4-connected grid, and various predicates are used to compute edge weights. Different graph structures and predicates can be used to segment in different ways. Our approach defines predicates in a similar way to operate on our image and point cloud data, but uses a fixed 4-connected graph structure.

Several approaches to planar segmentation of point cloud data have also been proposed. Many RANSAC-based approaches such as in [123] can be used to accurately

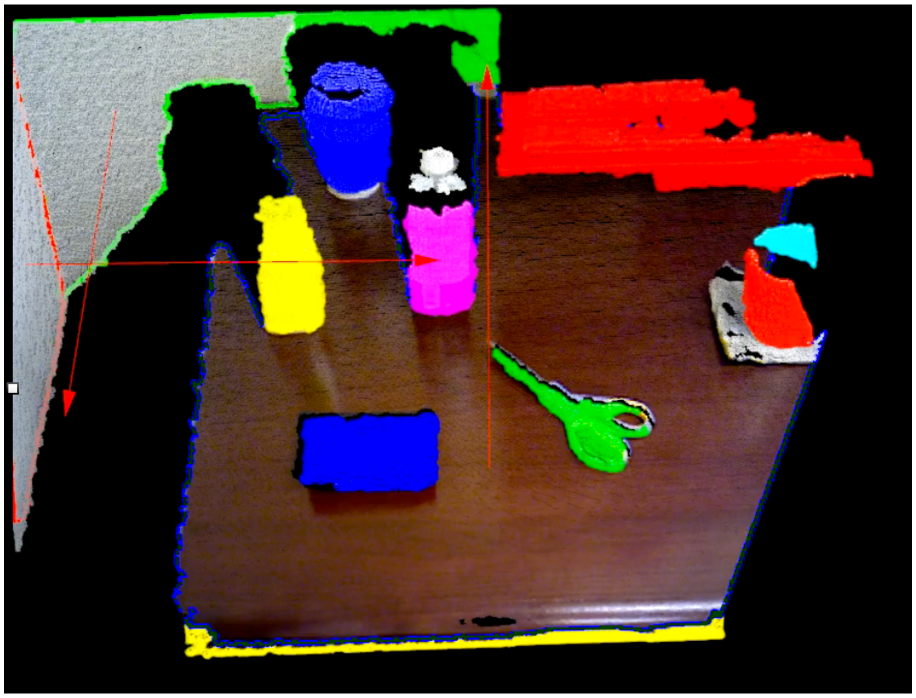


Figure 10: An example of a segmented tabletop scene. Planar surfaces are outlined in color, with large red arrows indicating the segmented plane's normal. Euclidean clusters are indicated by colored points.

segment planar surfaces, but these approaches were designed for unorganized point clouds as are produced by 3D laser scans, and so are much slower than approaches that can exploit organized data. Other approaches to unorganized point cloud segmentation have been surveyed in [156]. Some RANSAC-based approaches have been extended to exploit organized structure, such as the approach of Biswas and Veloso [13], which enables real-time performance.

Organized point cloud and range image segmentation has also been investigated. Poppinga *et. al* [110] proposed a region-growing based approach that selects a random point, and grows a region around it to the nearest neighbors that are nearby in plane space, incrementally updating the centroid and covariance matrix as points are added. Holz *et. al* extend this approach by pre-computing surface normals for the cloud, and incrementally update the plane's normal equation, which further reduces the computational cost [61] [60].

In contrast to these methods, our approach does not utilize seed points for region growing, but instead process each point sequentially. Additionally, we do not incrementally compute any statistics per region, instead delaying plane fitting until the image has been fully segmented, so that processing need only be performed for segments with a sufficient number of inlying points.

Hulik *et. al* evaluate several approaches to depth image segmentation, including a connected-component-like approach that operates on edge images [67]. This was reported to be quite efficient, but less accurate than some alternative methods such as RANSAC, for the datasets used in their testing.

3.4.2 Approach

3.4.2.1 Connected Component Algorithm

We propose a connected component based approach to segmentation which operates on organized point cloud data. An organized point cloud is a point cloud that has an image-like grid structure. We will denote an organized point cloud as $P(x, y)$, indicating the x and y coordinates of the point in the image-like structure. As in an image, given a point $P(x, y)$, neighboring points such as $P(x-1, y)$ and $P(x, y-1)$ are easily accessible in constant time thanks to the cloud's ordering. This organization is exploited by our segmentation approach, enabling us to skip the costly neighborhood step common to many segmentation approaches.

Different segmentation tasks may require different point representations, which may include RGB data as would be present in a color image, points in euclidean space as are commonly used in point clouds, as well as information like surface normals. Some types of information may not be available for each point in the organized structure. For example, RGB-D sensors often return invalid depth measurements, so no euclidean point is available. In these cases, a value such as NaN is used to signify that there is invalid data while maintaining the organized structure. For the purposes of segmentation, if required information such as a depth value or surface normal is not available for a point, the point will not be added to any segment.

The algorithm works by partitioning an organized point cloud P into a set of segments S. This is done by creating an integer label L for each point in the point cloud. This can be thought of as a label image. The label for a point at $P(x, y)$ will be denoted as $L(x, y)$. Points that have missing or invalid data can be excluded by using an invalid label such as NaN or maxint. Points that are members of the same segment will be assigned the same label. That is, if $P(x_1, y_1) \in S_i$ and $P(x_2, y_2) \in S_i$, then $L(x_1, y_1) = L(x_2, y_2)$. Points are compared using a comparison function or predicate. The exact comparison function will vary for different segmentation tasks, but it is of the form:

$$C(P(x_1, y_1), P(x_2, y_2)) = \begin{cases} true\ if\ similar \\ \\ false\ otherwise \end{cases}$$

If $C(P(x_1, y_1), P(x_2, y_2)) = true$, then $L(x_2, y_2) = L(x_1, y_1)$, else $L(x_2, y_2) \neq L(x_1, y_1)$. In the latter case, a new label is created by incrementing the largest assigned label by one. The exact comparison function C will vary based on the segmentation task, and several such functions will be described below.

The connected component labeling algorithm we use is similar to the two-pass binary image labeling algorithm described in [128], but has been modified to label point cloud data with continuous values based on some predicate, as in [45]. The algorithm begins by assigning the first point in the cloud with valid data with label 0. The first row and column of the cloud are then compared with the specified comparator C, to assign labels. The remainder of the points are treated by examining their neighboring points $P(x-1, y)$ and $P(x, y-1)$, as in a 4-connected grid. If both neighbors (above and to the left) have different labels, these labels must be merged with that of the current pixel, as these should be part of the same segment. An example of such a situation is shown in Figure 11, where a segment on the current row and previous row must be merged. We use the union-find algorithm to do this efficiently, as in [128]. Once the label image has been created, a second pass is performed to merge labels, assigning the lowest applicable label to the region, and producing the final connected-component label image $L(x, y)$.

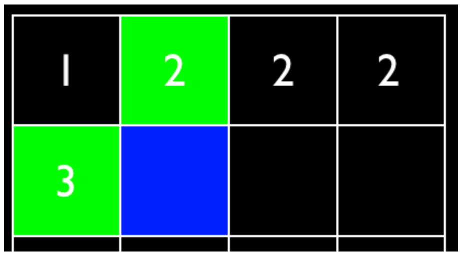


Figure 11: An example where two labels must be merged, if the pixel highlighted in blue matches both the neighbor above and to the left, shown in green. Merging is performed in a second pass using the union-find algorithm.

Many tasks can benefit from the boundary points of a segment. Given a segment, it is straightforward to compute the boundary by tracing the outer contour. Such an approach is also included in our implementation.

3.4.3 Planar Segmentation

Our planar segmentation approach segments the scene to detect large connected components corresponding to planar surfaces, including walls, the ground, tables, etc. We use the hessian-normal form to represent planes, which uses the well known equation $ax + by + cz + d = 0$. Our approach to planar segmentation begins by computing such a planar equation for each point in Euclidean space that has a valid surface normal.

Surface normals can be computed for the cloud using a variety of techniques. In this work, we use the integral image normal estimation method of Holzer *et. al* [62], which can compute normals in real-time for VGA RGB-D point clouds. After computing unit-length normals $\{n_x, n_y, n_z\}$ for each point, a point p can be represented as:

$$p = \{x, y, z, n_x, n_y, n_z\}$$

Additionally, given such a Euclidean point with its normal, we compute the perpendicular distance to this point with normal (the d variable of the plane equation), giving us a coordinate in plane space. This can be computed by the dot product:

$$n_d = \{x, y, z\} \cdot \{n_x, n_y, n_z\}$$

Augmenting our point representation with this information yields a point with a full plane equation:

$$p = \{x, y, z, n_x, n_y, n_z, n_d\}$$

Given this point representation, we then define distance metrics for both the normal direction and perpendicular distance components between two points. The distance in the range or d components of the plane equations is straightforward, as these are distances. The angular difference $dist_{normal}$ between the normal directions is given by the dot product.

$$dist_{normal}(p_1, p_2) = p_{1n} \cdot p_{2n}$$

$$dist_{range}(p_1, p_2) = |p_{1n_d} - p_{2n_d}|$$

We can then proceed with the connected component algorithm as described above, using a comparison function designed for planar segmentation. We first detect surfaces with smoothly changing surface normals, by requiring that surface normals of neighboring points are similar, within a threshold $thresh_{normal}$. Note that to avoid computing an inverse cosine for each point, we instead define the threshold $thresh_{normal}$ as the cosine of the desired angular threshold in radians.

$$C(p_1, p_2) = \begin{cases} true\ if((dist_{normal} < thresh_{normal}) \\ \quad \&\&\ (dist_{range} < thresh_{range})) \\ false\ otherwise \end{cases}$$

Using this comparison function with the above algorithm results in a set of labeled segments $L(x, y)$ corresponding to connected components in plane space. A visualization of such a label image is shown in Figure 21. Note that at this point, we have only examined local information, meaning that our segments may be only locally planar. The approach so far can be thought of as “smooth surface” segmentation rather than planar segmentation.

Next, we attempt a least squares plane fit for each segment with more than *min_inliers* points, resulting in a plane equation for each large segment. To ensure that the resulting segments are actually planar, the curvature is also computed, and a threshold *max_curvature* is used to filter out segments that are smooth but not planar.

3.4.4 Planar Refinement Algorithm

One shortcoming of the above approach is that it requires accurate surface normals, which are not always available. In particular, points / pixels near object boundaries and image boundaries tend to have noisy surface normals or no surface normals, which leads to segmented planar regions that end before the edge of the actual planar surface, as can be seen in Figure 12.

This can be addressed by performing additional passes of the connected component algorithm with a new comparator that extends existing planar segments to include adjacent points (in a 4-connected sense) that have a point-to-plane distance under a given threshold to the adjacent segment's planar equation. Given a point with normal $p = \{x, y, z\}$ and a plane equation $eqn = \{n_x, n_y, n_z, n_d\}$, the point-to-to

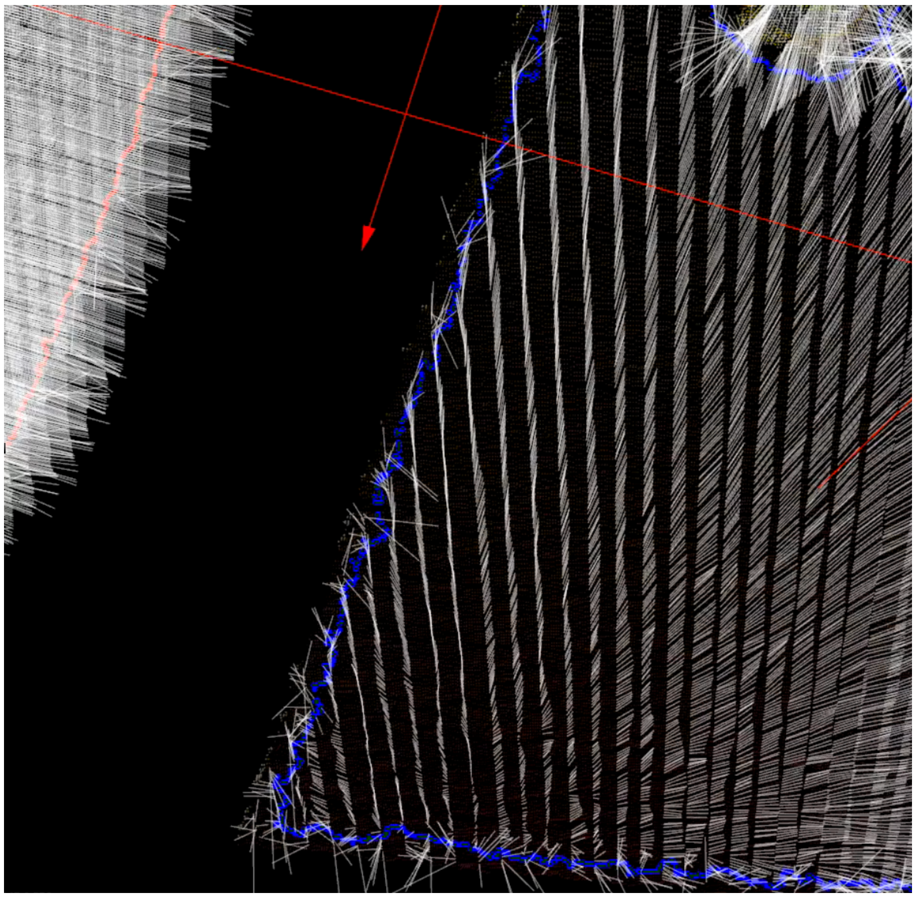


Figure 12: An example scene with noisy surface normals near object boundaries.

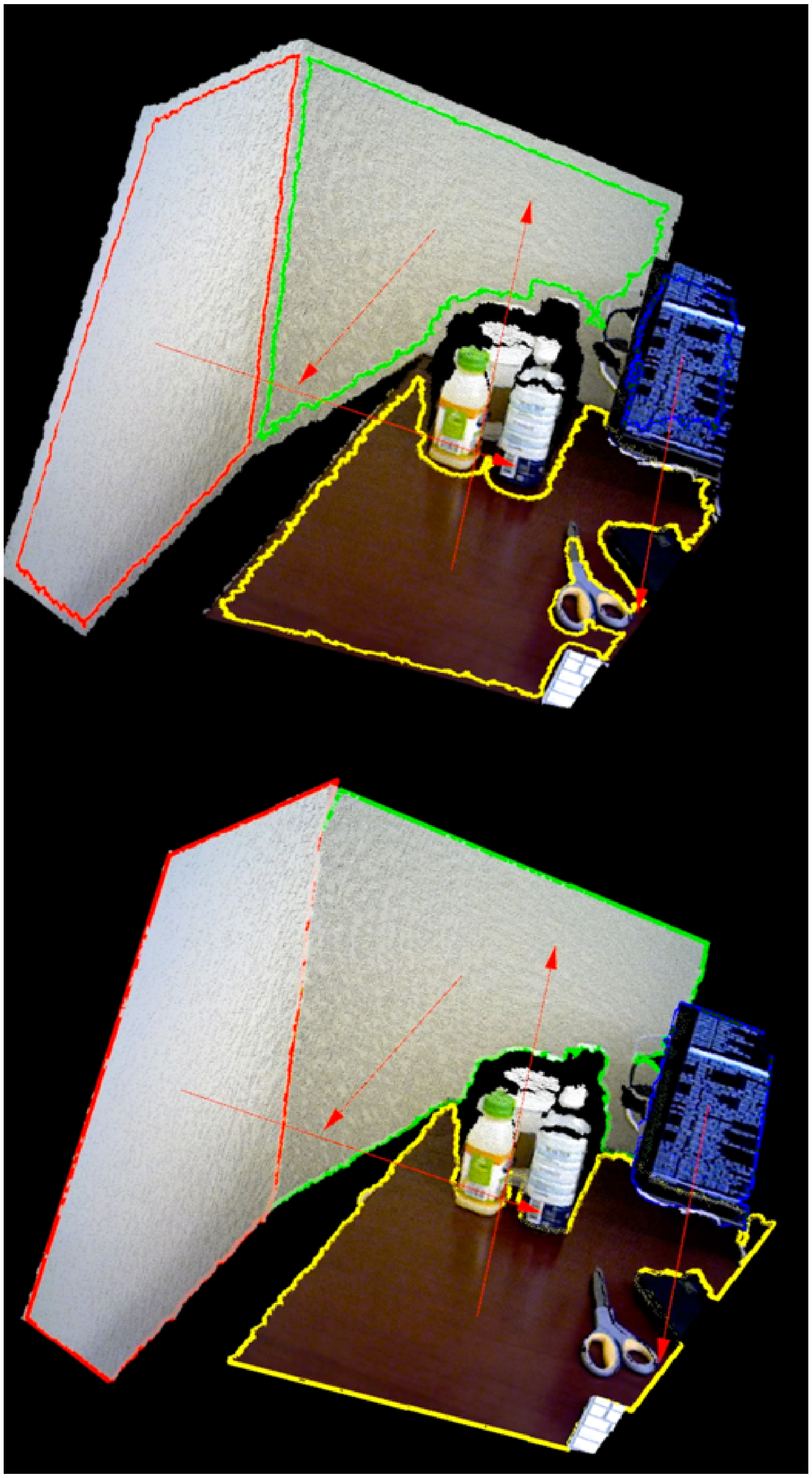


Figure 13: Top: a scene segmented without using the planar refinement approach described in Section 3.4.4. Bottom: a similar scene segmented with the planar refinement approach.

plane distance is given by:

$$dist_{ptp}(p, eqn) = \mid n_x * x + n_y * y + n_z * z + n_d \mid$$

The input to this comparison function requires the output labels L of our previous plane segmentation approach, as well as a set of labels $refine\_labels = \{l_1, ..., l_n\}$ which should be considered for refinement. Also required are the plane equations $eqns = \{eqn_1, ..., eqn_n\}$ corresponding to each segment label to be refined. Our planar refinement comparator works as follows:

$$C(p_1, p_2) = \begin{cases} false\ if\ (p_1 \notin refine\_labels \\ \quad \&\&\ p_2 \notin refine\_labels) \\ \quad ||\ dist_{ptp}(p_1, eqn(p_2)) > thresh_{ptp} \\ true\ otherwise \end{cases}$$

As this comparison only extends regions in one direction, two additional passes are required for refinement: a pass to extend planar regions "up and left", and a pass to extend "down and right". This is illustrated in Figure 14. While these additional passes do require computation, most points require very little process

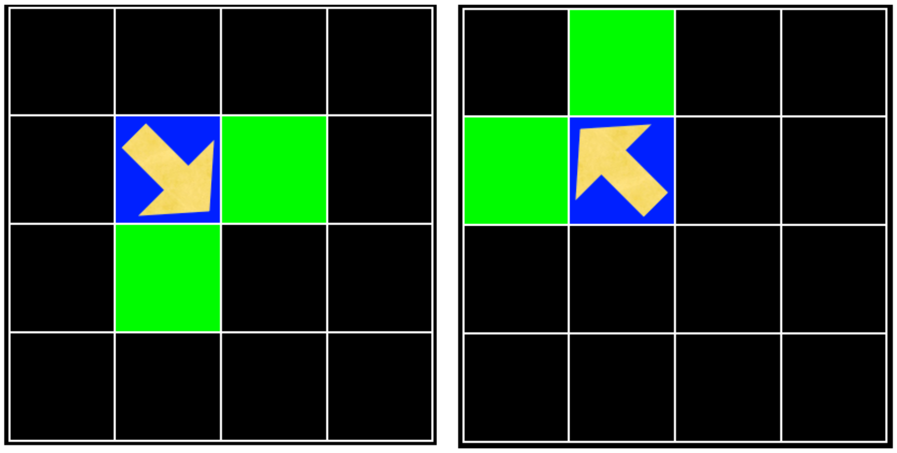


Figure 14: Planar refinement requires two additional passes over the image, one that extends regions "down and right", and one that extends regions "up and left".

3.4.5 Planar Segmentation with Color

The segmentation approaches described above have only made use of depth information and surface normals. However, the approach can be extended to also consider color information in addition to other features. To segment planar regions of similar color, a new comparison function can be defined that is identical to the planar segmentation comparison function, but additionally requires that points have a distance in color space d_{color} below some threshold $thresh_{color}$. While perhaps not the best distance metric in color space, we used euclidean distance in RGB space to demonstrate this approach. An example scene is shown in Figure 15.

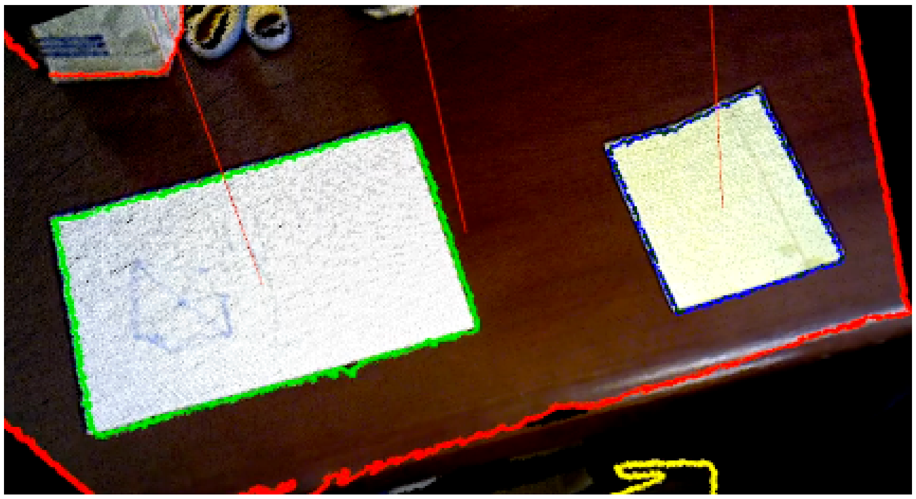


Figure 15: An example scene segmented for planes of similar color. The sheets of paper on the desk are segmented as separate planar regions, as their color does not match the desk.

3.4.6 Euclidean Clustering

Similar to the algorithm described in [124], we take a greedy approach to euclidean clustering. The comparison function used is based on euclidean distance $d_{euclidean}$, given by the well known function.

To be useful for some tasks such as tabletop object detection, this approach also needs to be able to take an image mask. For computational reasons, we instead use a label image L from a previous step, as well as a set of labels $exclude\_labels$ to be included in the mask. For the purpose of tabletop object detection, we first segment planar surfaces, and use the planar regions as our mask – the corresponding labels are used as $exclude\_labels$. One could also construct a binary mask by making a label image with one label set as part of the mask. We can then define a comparison function as:

$$C(P_1, P_2) = \begin{cases} false\ if\ (L(P_1) \in exclude\_labels \\ \quad || \ L(P_2) \in exclude\_labels \\ \quad || \ d_{euclidean}(P_1, P_2) > d_{thresh}) \\ true\ otherwise \end{cases}$$

3.4.7 Road Segmentation

Organized connected component segmentation can also be applied to point clouds generated from a stereo pair. As part of the PCL Honda Research Code Sprint, we developed a comparator for road surface detection from stereo data, for use with the organized connected component segmentation algorithm. One important note here is that this does not assume that the ground is perfectly planar, as most roads are not. Roads that slope towards the edges are quite common, primarily for drainage purposes. Instead, we assume that the drivable surface has smoothly changing normals, and that the points lie within some expected range (possibly quite wide) from the "expected ground plane". This requires the "expected ground plane" as input, specified in Hessian normal form. The expected ground plane is specified with respect to the left camera pose, and is the ground plane the vehicle's tires would be on, if the ground were perfectly planar, as shown in Figure 16. Tolerances for the normal direction and range are also specified as input to the segmentation. This allows points that are too-far outside the drivable range to be excluded. Given the

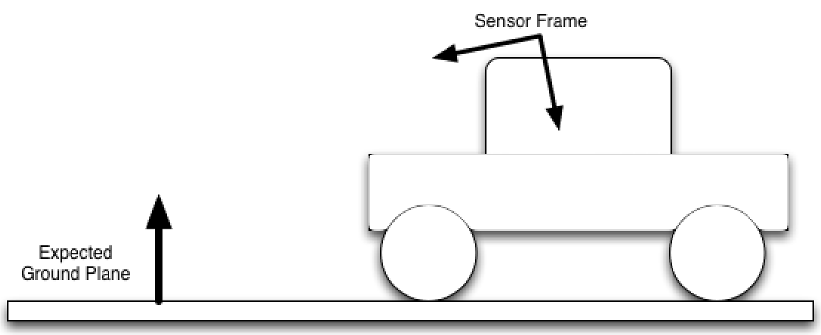


Figure 16: A diagram demonstrating the "expected ground plane" – the plane relative to the sensor pose that the vehicle's wheels should rest on.

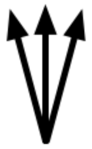


Figure 17: Given the expected ground plane normal in the center, the normals at the sides indicate the acceptable range of normals given an angular threshold of 15 degrees.

expected ground plane equation $G = \{a, b, c, d\}$, and thresholds $road\_angular\_thresh$ and $distance\_thresh$, to be considered part of a road segment, a point P with normal N must satisfy the following condition:

- $N \cdot \{a, b, c\} > cos(road\_angular\_threshold)$

An example demonstrating this can be seen in Figure 17.

To enforce smoothness, points must also have a similar surface normal to their neighbors. Recall that points are compared in a 4-connected sense in our algorithm. To be considered part of the same segment, a point P_1 with normal N_1 and a neighboring point P_2 with normal N_2 must satisfy the following condition:

- $N_1 \cdot N_2 > cos(angular\_threshold)$

An example demonstrating this is shown in Figure 18.

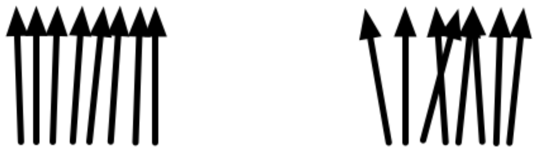


Figure 18: At left, adjacent normals with acceptably smooth variation given an angular threshold of 2 degrees. At right, normals without smooth variation. Note that all normals shown are within the acceptable range from the example above.

The result of using this comparator with the organized connected component algorithm is a set of zero or more regions that fit the above criteria: they are connected in the image, have surface normals within the specified range of the expected ground plane, have smoothly changing normals within the segment, and are connected within the image. Only regions with more than $min\_inliers$ points are returned.

Surface normals can be quite noisy on stereo data, which can result in several disjoint regions labeled as ground. To alleviate this problem, we perform an additional segmentation run on the point cloud to grow the set of ground regions to include points that do not match the surface normal requirements described above, but have a point-to-plane distance less than a specified threshold, and are adjacent pixels to one of the identified road surface regions. The "planar refinement" comparator as described in Section 3.4.4 can perform this task, and can be called by using the "refine" method of the organized connected component segmentation algorithm.

As the resulting points do not meet the more strict requirements described above, we consider these to be lower confidence ground points. These are denoted in yellow in the resulting images, rather than green as is done for road regions that do match the surface normal requirements.

Limited experiments were also performed for detection of nearby obstacles. Once again, we make use of the organized connected component class, but with a different comparison function. To detect obstacles, we detect sets of pixels that have valid depths, where points are within some distance of each other in euclidean distance, and are not part of the previously labeled groundplane. Segments that have both a sufficient number of points, and have centroids that are sufficiently far away from the plane are returned. An example is shown in Figure 19.

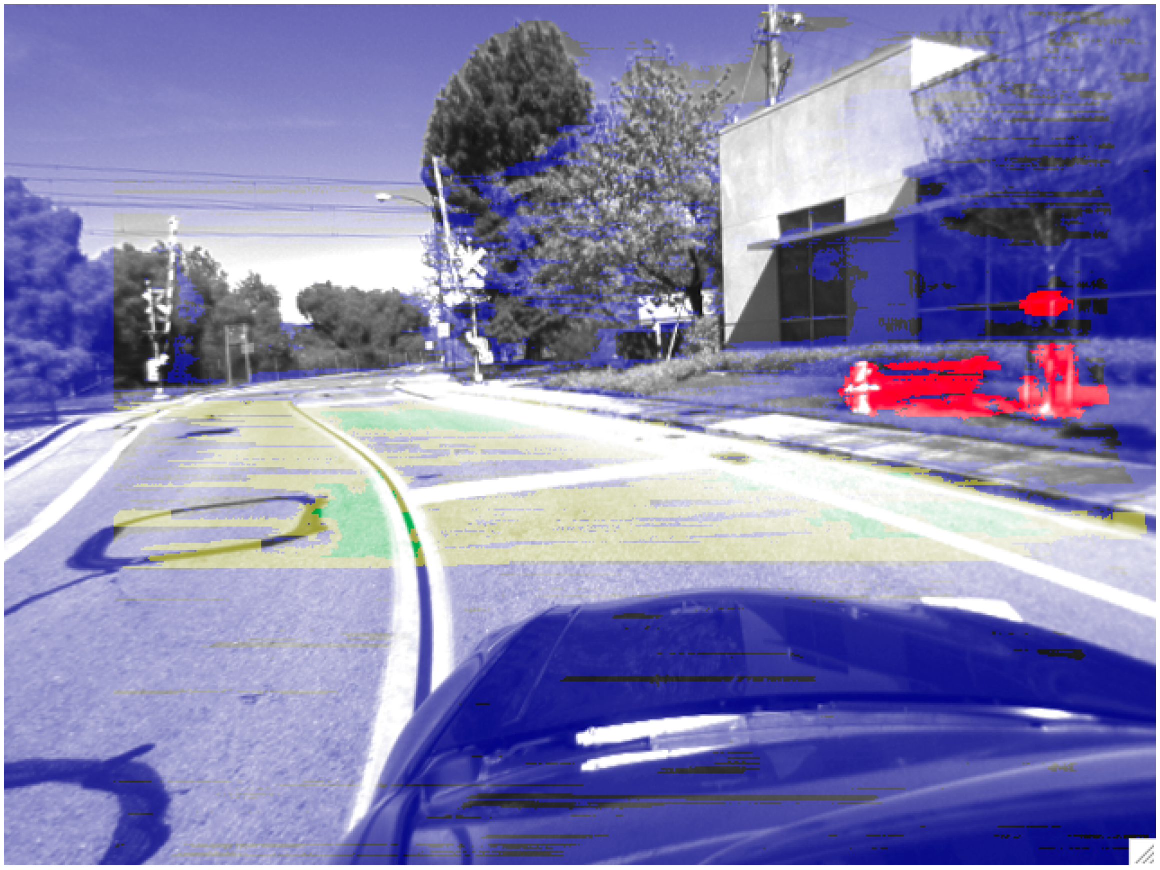


Figure 19: An example of the obstacle detection. Any connected component not part of the road is considered as an obstacle.

3.4.8 Implementation

The above described approach is available in the Point Cloud Library (PCL) [123] as the Organized Connected Component Segmentation module. This class performs the segmentation algorithm described in Section 3.4.2 given some comparison function, such as the one described in Section 3.4.3. To use a different comparison function, different comparator classes are used. A comparator contains a compare function that takes two points as parameters, and returns a boolean *true* if the points should be members of the same segment, and $false$ otherwise. The class may contain additional information such as thresholds or a mask, as used in the clustering approach described in Section 3.4.6. Detailed documentation for all classes is available on the PCL web site (*www.pointclouds.org*).

To achieve efficient performance, we have parallelized this approach. The planar segmentation approach requires surface normals to have been computed for a given point cloud, so this is performed for the most recent cloud received from the sensor. In parallel with this, planes are segmented for the previous frame, for which normals have already been computed. A system diagram illustrating this processing pipeline is shown in Figure 20.

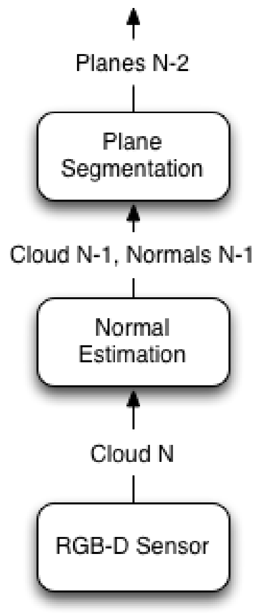


Figure 20: A system diagram demonstrating our planar segmentation processing pipeline. The normal estimation and plane segmentation operations occur in parallel, enabling real-time performance.

3.4.9 Discussion & Evaluation

Our segmentation approach has been applied to several applications, including tabletop object segmentation mapping with planar landmarks. We present quantitative runtime results for several datasets, as well as a qualitative discussion of applications to tabletop object segmentation and SLAM.

3.4.9.1 Runtime Evaluation

The runtime of our approach was evaluated on Kinect data available from the TUM RGB-D Dataset [136] at `http://vision.in.tum.de/data/datasets/rgbd-dataset`. We selected three datasets from this repository with various kinds of scenes. The "freiburg1 desk" dataset contains close-range scenes of an office desktop environment. The "freiburg1 floor" dataset contains scenes of a floor in an indoor environment, so a large plane is present in all frames. The "freiburg2 pioneer_slam" dataset includes scenes from a forward-looking RGB-D camera mounted on a mobile robot moving through a large room, as in a SLAM scenario. These datasets were converted to PCD format, and replayed from files at a rate of 30Hz. These experiments were performed on a 2.6 GHz Intel Core i7 CPU. A multi-threaded evaluation tool using the tools described in PCL implementation from Section 3.4.8 was designed for this application.

Table 2 presents runtimes for planar segmentation, as well as the individual steps of normal estimation and planar segmentation. Normal estimation and plane segmentation were run in parallel, which is necessary for real-time performance. However, these steps can also be run sequentially if only a single core is available, at the expense of frame rate. The frame callback timings are the average time elapsed since the previous frame was fully processed.

Table 2: Average running times for normal estimation and plane segmentation without planar refinement.

	fr1 desk	fr1 floor	fr2 pioneer_slam
Normal Estimation	21.56 ± 2.07ms	22.98 ± 1.93ms	21.97 ± 3.91ms
Plane Segmentation	26.13 ± 3.19ms	21.28 ± 3.09ms	23.17 ± 5.62ms
Frame Callback	33.51 ± 3.13ms	33.62 ± 2.71ms	33.98 ± 4.03ms
Callback Rate	29.83 Hz	29.73 Hz	29.42 Hz

Table 3: Average running times for normal estimation and plane segmentation with planar refinement.

	fr1 desk	fr1 floor	fr2 pioneer_slam
Normal Estimation	21.91 ± 2.36ms	23.99 ± 2.57ms	22.00 ± 3.86ms
Segmentation+Refinement	32.55 ± 3.29ms	29.83 ± 3.38ms	29.53 ± 6.34ms
Frame Callback	35.79 ± 3.20ms	34.08 ± 3.23ms	35.74 ± 4.35ms
Callback Rate	27.93 Hz	29.34 Hz	27.97 Hz

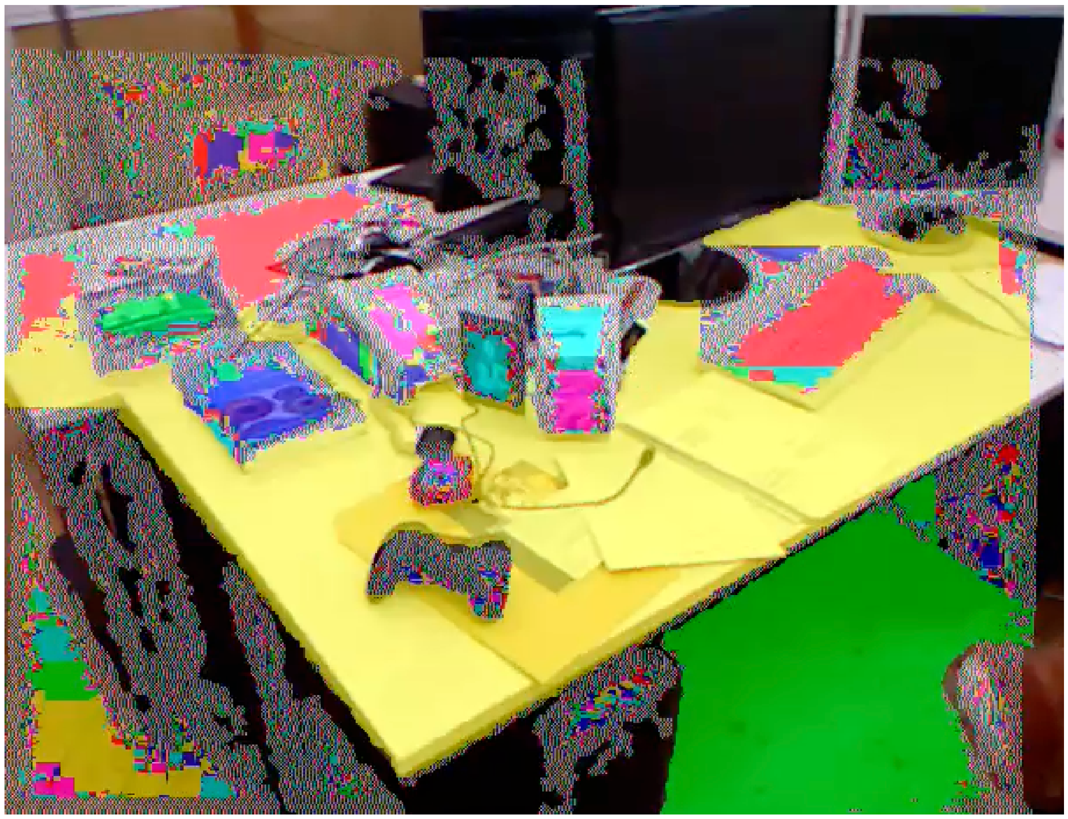


Figure 21: An example image from the desk1 dataset, with a colorized representation of the label image superimposed on top. The displayed label image was generated using the planar segmentation approach with refinement.

3.4.9.2 Timing Comparison to RANSAC-based Segmentation

We additionally performed an informal timing comparison to our previous approach of RANSAC-based plane segmentation, as presented in Section 3.3. We used a 508 point-cloud dataset collected by a Microsoft Kinect RGB-D camera mounted on a PR2 robot, as it navigated through a typical indoor environment. In this case, the environment was the "Yesterday's Sushi" testing area as described at [1]. The area contains several walls, tables, and shelves, as is common in restaurants, homes, and offices.

Three approaches were run on frame of this dataset, and the runtime of each algorithm was recorded. The proposed organized multi-plane segmentation approach was run using a full-resolution VGA (307,200 point) point cloud data, using the "segment and refine" comparator as described in Section Summary statistics are shown in Table 4, and a box plot is given in Figure 22.

Table 4: Comparison of running times to our previous approach. Note that RANSAC based approaches were only run on down-sampled data.

Approach	mean (milliseconds)	standard deviation
Multi-Plane Segmentation	0033.2	0002.3
RANSAC (downsampled, single plane)	0320.0	0340.0
RANSAC (downsampled, all planes)	1520.0	1790.0

3.4.9.3 Tabletop Object Segmentation Discussion

The approach described above has been used for tabletop object segmentation from RGB-D camera. An example scene is shown in Figure 10, and a video example is available on YouTube at `http://www.youtube.com/watch?v=Izgy99WHFBs`. Segmented planar surfaces are displayed with colored boundary points, and large red arrows indicating their normal direction. Segmented objects are shown as colored point clouds.

As can be seen, well-separated objects above a planar surface can be segmented

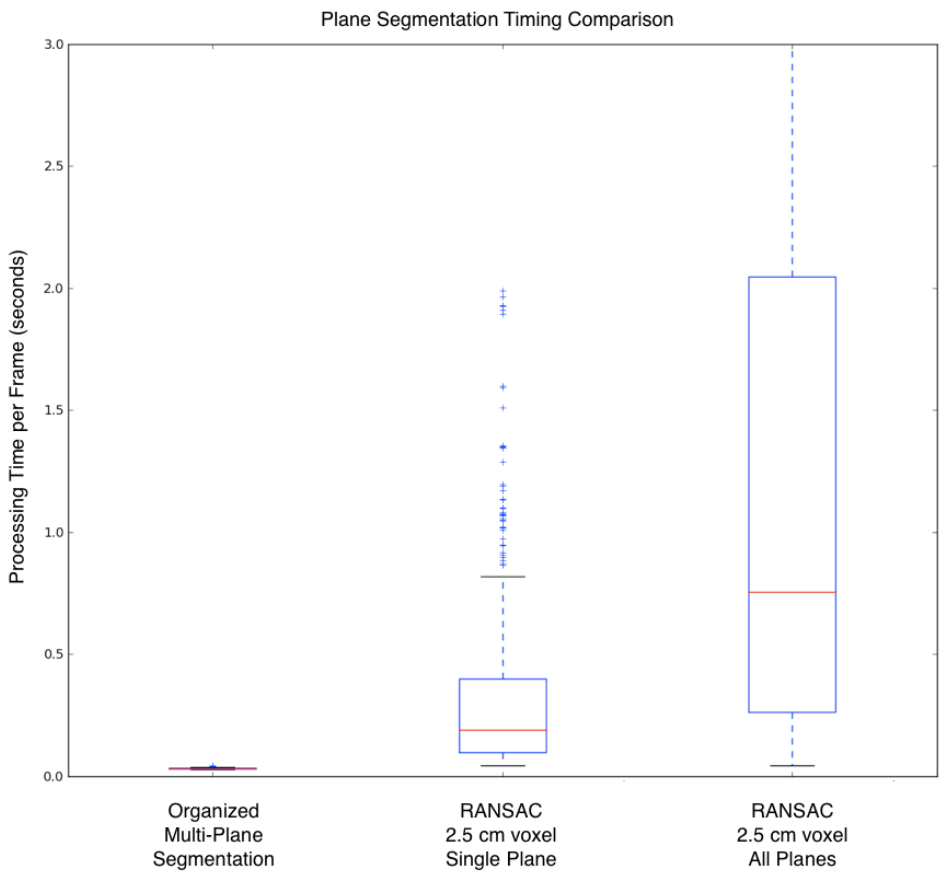


Figure 22: Comparison of running times to our previous approach. Note that RANSAC based approaches were only run on 2.5cm voxel grid down-sampled data both for the dominant plane only (center) and for all planes (right), while full resolution data was used for our approach (left).

in close-range scenes. Highly cluttered scenes pose a challenge, as these tend to be under-segmented. The key to this approach is that using the planar surface as a mask enables the clustering to separate the objects, rather than connecting them via the tabletop surface. In our experience, using the planar refinement step produces a much better mask, leading to a better segmentation.

3.4.9.4 Mapping Discussion

Our segmentation approach has also been used in the context of a 3D mapping task, in which planar surfaces were segmented from an RGB-D camera mounted on a mobile robot moving through an indoor environment. Multiple observations of each surface were merged to create a map such as the one shown in Figure 23. This is similar to the approach using in our previous planar mapping work [149], which was based on RANSAC segmented planes. Qualitatively, we found the planes segmented by the proposed approach to be preferable to the previously used RANSAC based approach. RANSAC necessitated downsampling the data to achieve reasonable performance, which degrades the quality of the planar region boundaries. Additionally, with the RANSAC-based approach, we computed either a convex hull or alpha shape to represent the boundary, which does not always accurately represent the true shape of the region. In contrast, this approach produces the exact boundary points from the image, producing a better representation of the planar boundary. Usage of these segmented landmarks in the context of SLAM and semantic mapping will be given in Chapter 4.

3.4.9.5 Ground & Road Segmentation Results

Limited experiments were also performed to evaluate the road segmentation segmentation of stereo data both quantitatively and qualitatively. Video results are provided for each sequence, which are described below. Additionally, images were labeled for use as ground truth, and compared to the segmentation results.

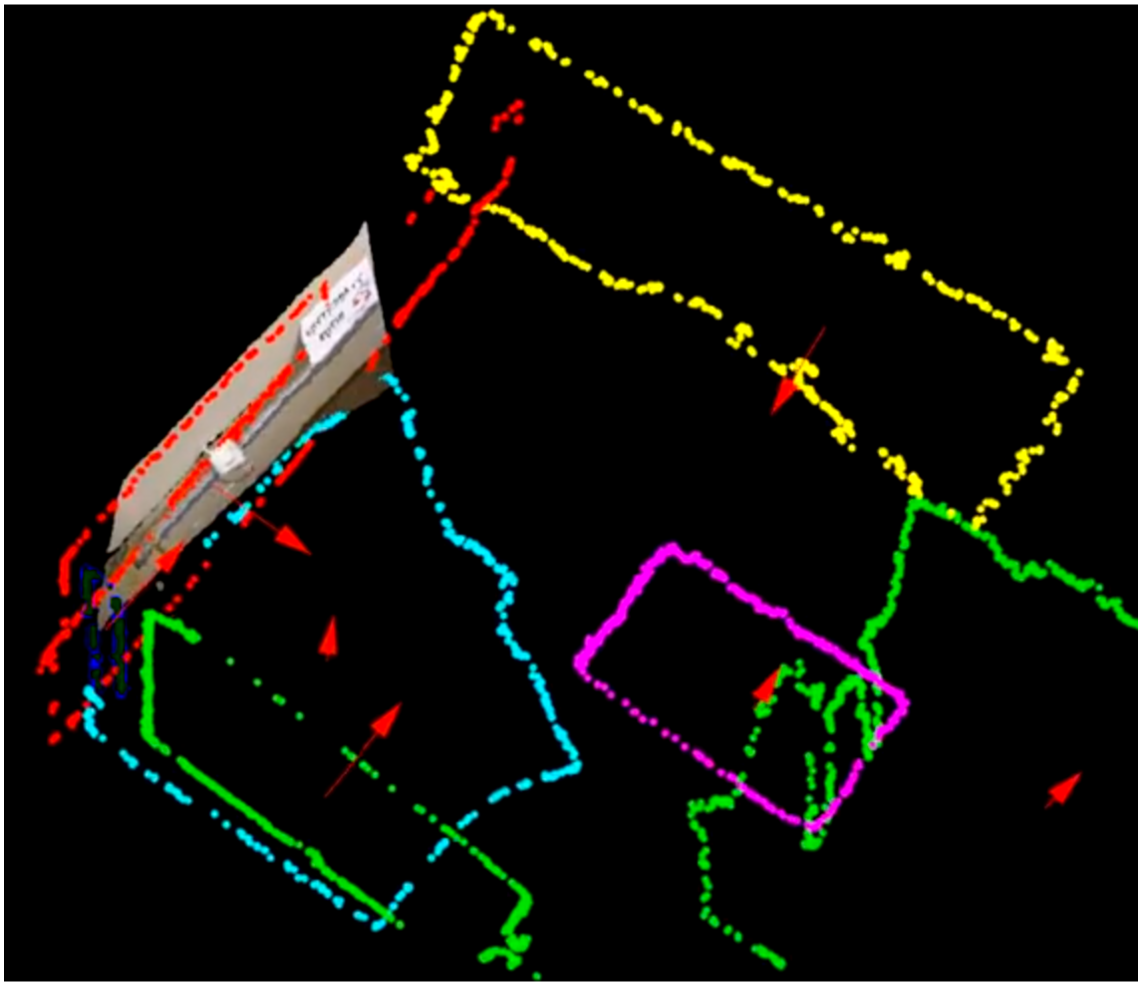


Figure 23: An example of an indoor mapping tasks using planar landmarks. The colored points represent boundaries of planar surfaces segmented from RGB-D data, and merged from multiple observations.

An initial verification of the ground segmentation approach was performed on data collected from a Microsoft Kinect on the Jeeves mobile robot. Point clouds generated from the Kinect are much denser and less noisy than point clouds generated from stereo, but this was suitable for an initial verification of the ground segmentation approach. An example is shown in Figure 24, and the full video sequence is available on Youtube at the following URL: `http://www.youtube.com/watch?v=nHvRGaDO72c`.

Several stereo datasets were provided by Honda for evaluating this technique. Point clouds were generated from provided stereo image pairs using PCL's stereo tools, also developed as part of this code sprint. Several example results are shown in Figure 25. Video results for the castro dataset are available on Youtube at the following URL: `http://www.youtube.com/watch?v=ZCuOxw3thDE`. Video results for the national dataset are available at: `http://www.youtube.com/watch?v=7uGPbg7XZOE`. Video results for the maude dataset are available at: `http://youtu.be/DU5G-dHOlog`.

To quantitatively evaluate the segmentation approach, a small subset of the provided images from were manually labeled, and compared with the segmentation result. For these quantitative results, every 500th frame from the castro dataset was used (12 images). Roadways were labeled in green. In addition to the roadways, sidewalks, curbs, and other surfaces that the vehicle could (but probably should not) drive on were labeled in yellow, to allow us to provide more informative statistics regarding false positives. An example of such a labeled image is shown in Figure 26.

An evaluation script was written to load both the ground truth image and segmentation result for comparison. Results are given in Table 5. The values indicates the percentage of image pixels for which valid depths were computed, and the second indicates the percentage of ground truth pixels with valid depths – this is the percentage of the image that resulted in valid points. Reasons for invalid depths include areas not in view of both cameras, and low texture areas such as sky or the car hood. We should also note that points at the same or closer distance to the camera as the

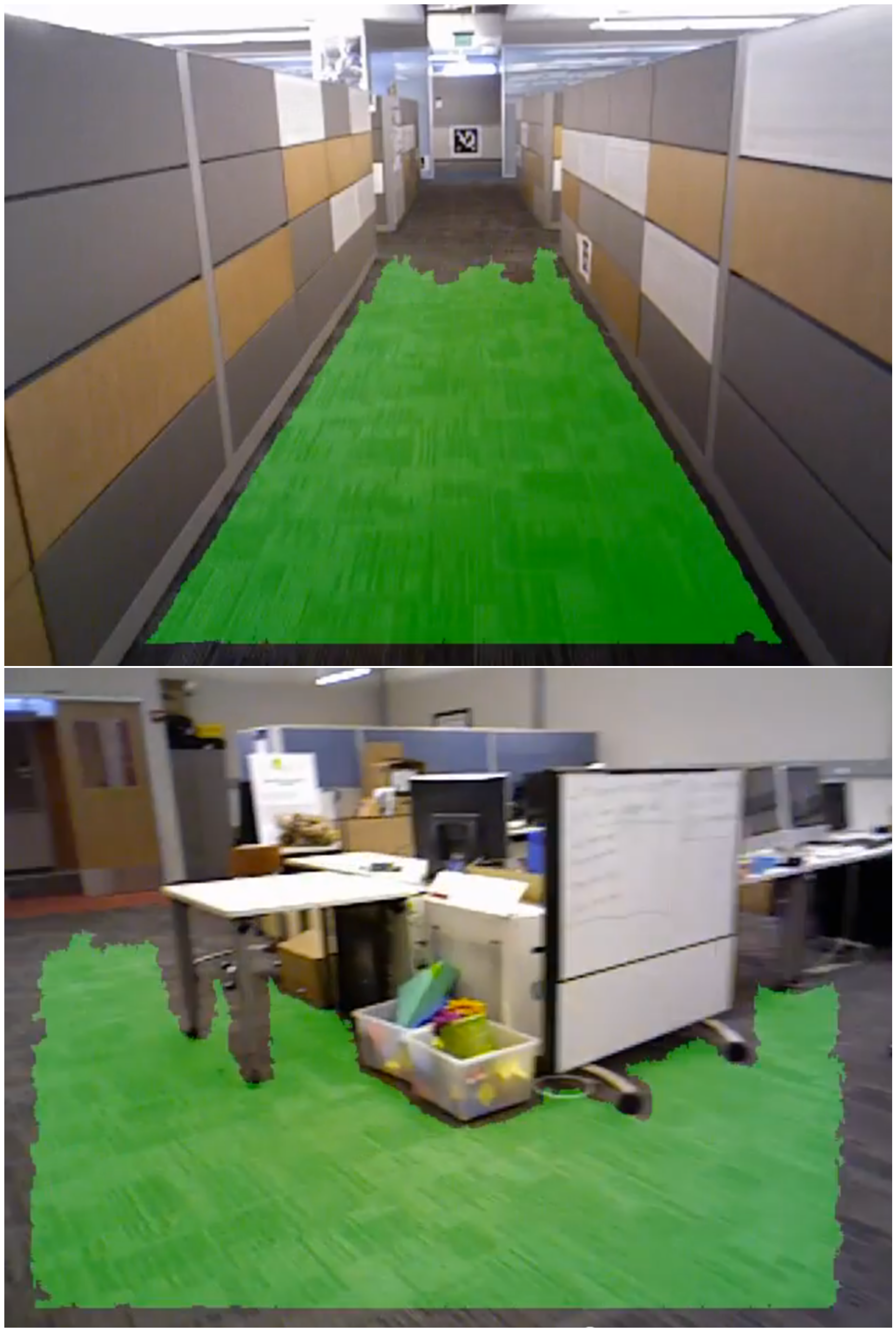


Figure 24: Example scenes of Kinect data segmented using our approach. This was performed to validate the segmentation approach during the development of the stereo tools.

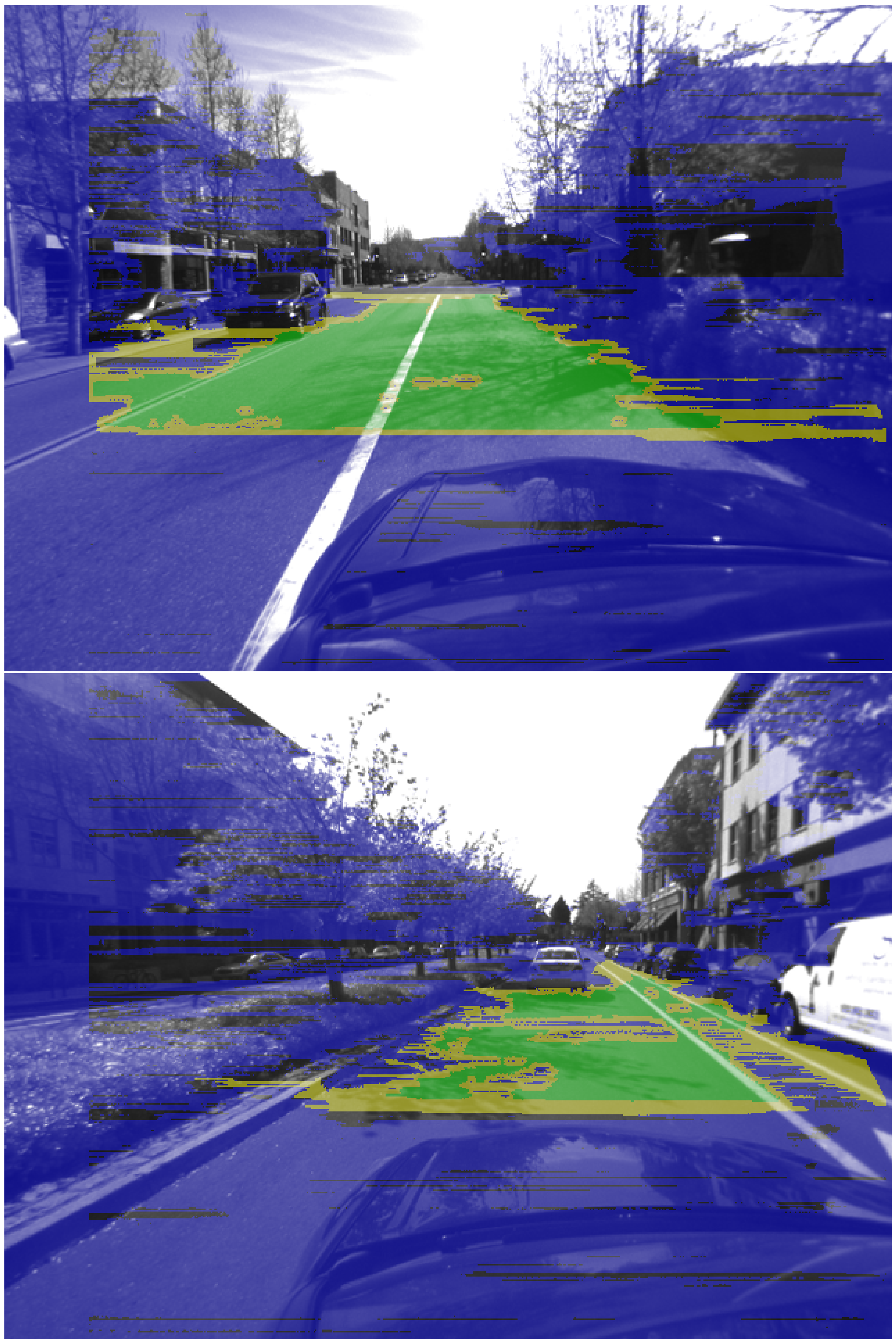


Figure 25: An example of our segmentation results on the Castro dataset. Segments detected using the road surface comparator are shown in green, and pixels added by the road region growing are shown in yellow. Pixels without valid depth data are shown in blue.

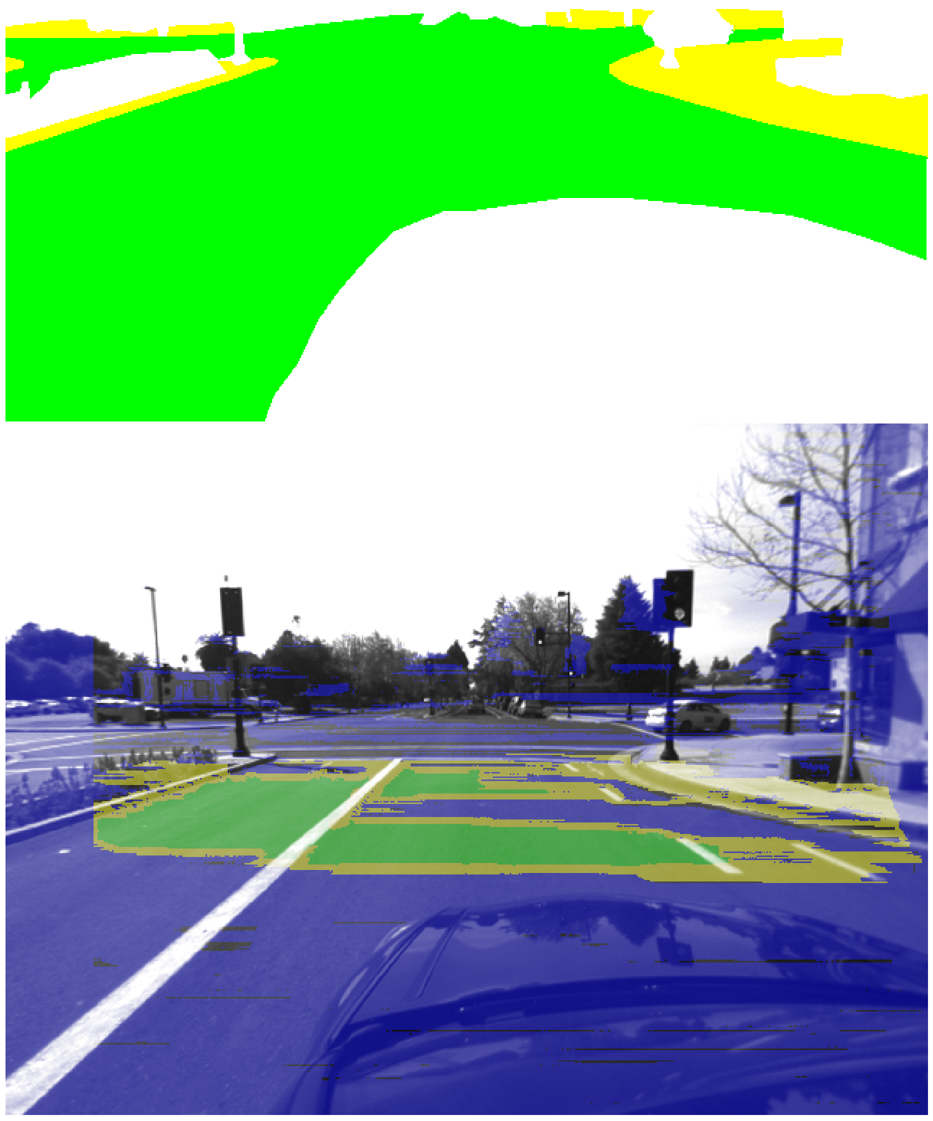


Figure 26: Top: an example of the ground truth segmentation for frame 2500 of the Castro dataset. Roadway is labeled in green, while sidewalks, curbs, and other low-lying areas are labeled in yellow. All other pixels are labeled with white. Bottom: our segmentation result for this image.

car hood will not have valid depths, as the disparity range used for these experiments was limited to disparities smaller than this. This design decision was made because most of these points are not visible anyway (due to the car hood in one or both images), and the complexity of most matching algorithms scales with the size of the disparity range. Since the points are the input to the segmentation system, all subsequent statistics are reported with respect to input points (valid depths), rather than all pixels. We report separate values for both the regions segmented using the road surface comparator (shown in green), and regions extended with the plane refinement comparator (shown in yellow).

Table 5: Ground Segmentation Results

% pixels with valid depths (points)	29.7539
% ground pixels with valid depths (points)	33.6766
% points correctly classified (High Confidence / Green)	32.4700
% points correctly classified (High and Low Confidence / Green +Yellow)	77.5866
% non-ground points classified as ground (High Confidence / Green)	0.7390
% non-ground points classified as ground (High and Low Confidence)	5.2089
% non-ground+sidewalk points classified as ground (High Confidence / Green)	0.0844
% non-ground+sidewalk points classified as ground (High and Low Confidence)	1.5273

The results shown demonstrate that the drivable ground surface can be detected in many circumstances using our approach, but there are also a number of shortcomings. The results on the Kinect dataset indicate that the base algorithm works well on dense and clean data, but current stereo techniques do not yet produce point clouds of comparable density and quality. The most apparent difference is the relatively large number of pixels without valid depths, giving us many fewer pixels to work with. Another challenge is that it is often difficult to distinguish roadways from sidewalks when using purely geometric cues – this seems to be the most common failure mode in the results, as can be seen in the above quantitative results.

3.5 Object Discovery & Recognition

Service robots that interact with objects in the world need to be able to detect objects and recognize them. These can be objects that the robot will need to interact with and manipulate, or can be objects that aid in localization and mapping. This Section describes several techniques for detecting and recognizing objects.

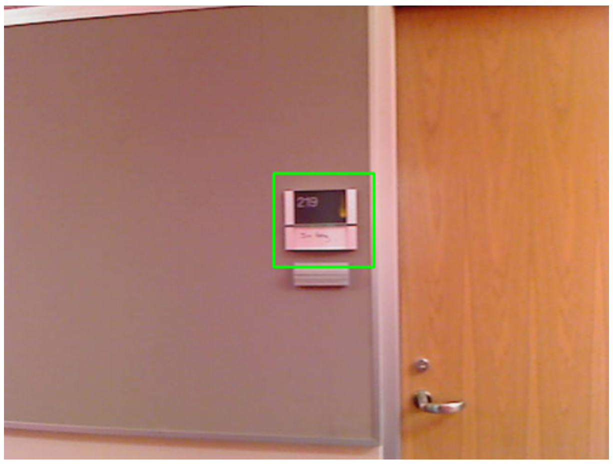
3.5.1 Object Detection

For many approaches to object recognition, searching all sensor data in a brute-force manner can be intractable. To address this, possible objects are often detected using other means, so that potentially expensive feature computation can be focused on areas that are likely to be objects. We describe two approaches to this here.

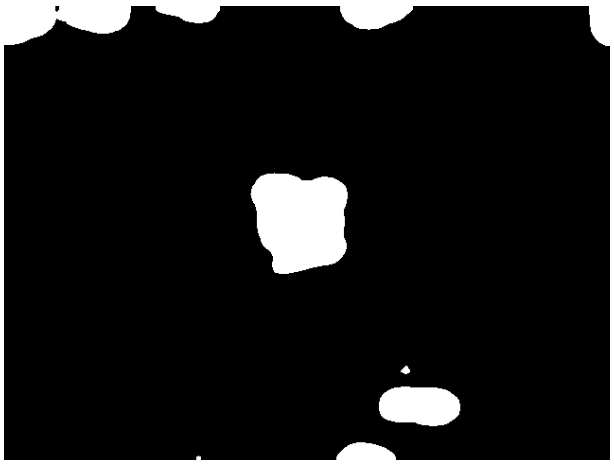
3.5.1.1 Image-based Saliency

Visually salient regions in images can serve as a cue for detection of objects of interest. This approach was used in our work on door sign recognition and mapping [120].

One approach to detection of visually distinctive regions of images is the spectral residual saliency method of Hou and Zhang [65]. An example of a detected region is shown in Figure 27(b). Connected components are detected within the saliency image,



(a) Example image seen by robot



(b) Saliency mask

Figure 27: An image of a door sign seen by the robot is shown in figure 27(a) and the resulting saliency mask is shown in figure 27(b)

and appropriately sized regions are considered as potential objects. One limitation of this approach is that it assumes objects will be more textured than the rest of the scene, which may not always be the case. However, we found this to work well for our application of detection of door signs on walls, as these signs tend to have more texture than the walls they're attached to.

Recognition of door signs detected using this approach is described in Chapter 3 Section 3.5.3, while usage as landmarks in SLAM is described in Chapter 4 Section 4.3.

3.5.1.2 Point Cloud-based Clustering

When depth information is available, this can also be of use for object detection. Section 3.4.6 describes an approach to detecting spatially connected point clusters in point clouds. Such clusters are useful as potential objects, and the following sections will describe methods for attempting to recognize such clusters as known objects. Depending on the application, further filtering of these clusters may need to be performed. For modeling objects of interest to a mobile manipulator, as in our work [24], we chose to filter clusters that are either too small or too large. This can effectively exclude noise and large furniture. Only clusters that include at least 1000 points and have a bounding box volume of less than 1 cubic meter were considered in that paper. These values were determined empirically.

Usage of these clusters for online modeling and subsequent use as landmarks in SLAM is described in Section 4.4.

3.5.2 3D Cluster Recognition

Given a cluster considered as a potential object as described in Section 3.5.1.2, we can take a feature-based approach to recognizing the object. One way to do this is to use visual features such as SURF or SIFT extracted for only this region. For RGB-D sensors, the image data for only inliers to the cluster can be accessed directly, but even this approach also works with separate 3D and 2D sensors, such as a 3D laser scanner and a 2D camera. This approach was used in our previous work [153]. The point cloud corresponding to the object region was projected into the image from the DSLR camera, and this region was used as a mask. SURF features were detected within this image region, and used for recognition.

More recently, features that leverage both 2D and 3D information have become popular. One such approach is the SHOTCOLOR descriptor [3, 141], which we used in our work [24]. A summary of the recognition approach from that work is provided here.

Given an object point cloud detected via the object discovery method described in Section 3.5.1.2, a set of keypoints $\mathbf{K}$ are extracted from a 3cm voxelized version of this cloud. The downsampled / voxelized cloud is used only for keypoint detection, to speed computation – all other operations occur on the full resolution cloud candidate cloud. Surface normals for the object cloud were then computed with a 1 cm radius using the normal estimation approach of Holzer [62]. CSHOT descriptors $\mathbf{D}$ are then computed for the keypoints. Object candidate centroids $\mathbf{C}$ are also computed. All points are represented in the map coordinate frame. An object can then be denoted:

$$\mathbf{O} = \{\mathbf{K}, \mathbf{D}, \mathbf{C}\} \tag{1}$$

In Section 4.4, we will discuss in detail how to use this object discovery and recognition approach to use recognized objects as landmarks in SLAM, and how new objects can be discovered, modeled and mapped online during SLAM.

3.5.3 Door Sign Recognition

Given visually detected features, an alternative approach to recognition is to train a classifier, such as a support vector machine (SVM) trained on Histogram of Oriented Gradient (HOG) features [32]. In our previous work [120], we used this approach to recognize door sign candidates as detected by the salient region detector described in Section 3.5.1.1. The recognition approach and classifier training method is described here, while usage as landmarks in a SLAM problem is given in Section 4.3.

3.5.3.1 Histogram of Oriented Gradients

HOG features, as their name suggests, capture patterns of image gradients, as described by a 2D histogram. Because humans use door signs for navigation, they are typically designed to be visually distinct from their surroundings, making them easy to see. This usually results in sharp contrast with respect to the background, producing strong edges that are easily detectable. HOG features are perhaps best known for use in person detection, as proposed by Dalal and Triggs in [32]. Here we apply them to door sign recognition.

HOG features are computed on a region of interest by first normalizing image contrast over a set of overlapping sub-regions. A set of oriented edge filters is then convolved with the image in a regular grid, the responses of each edge filter being accumulated to fill the histogram bins.

The algorithm includes several tunable parameters, including the number of bins (orientations of edge filters), size of detection window, and contrast normalization window. We performed a coordinate ascent on these parameters to select appropriate values for our application, using 3-fold hold-out cross validation. This resulted in using a 16x26 pixel window size, and 4x4 histogram cells per window, with 9 orientations. We did not detect an advantage for varying the contrast normalization, so we use only one contrast normalization window. We believe this is due to the relatively small size of the door sign regions, so contrast within these typically does not vary significantly. The OpenCV [15] implementation of HOG is used in this work.

3.5.3.2 Support Vector Machines

The previous Section described how to compute a histogram of oriented gradient (HOG) feature for an image region of interest. Given these histograms,a classifier must be trained to determine if the feature represents a door sign or not. As in [32], we use a Support Vector Machine (SVM), a discriminative classifer which finds the maximum margin decision boundary on a kernel function for the input feature. The data points nearest to the descision boundary, histograms in this case, are a set of *support vectors* which will be used to perform the classification. An example HOG feature is classified by evaluating the kernel function with respect to each support vector according to Equation 2, where $y(x)$ is the predicted label, n is the number of support vectors, ω_n is the weight of the n-th support vector, $k(\cdot,\cdot)$ is a kernel function, and b is an offset.

$$y(x) = \sum_{n=1}^{N} \omega_n k(x, x_n) + b \tag{2}$$

Several kernel functions were tested for this application, including linear kernels, polynomials, and radial basis function (RBF) kernels. SVMs using RBF kernels performed well in our testing for cross-validation and generalization, so we chose to use these for our classifier. Coordinate ascent was performed to tune the γ parameter; a value of 1.0 was found to perform best. The SVM implementation in OpenCV [15] was used for this work.

The SVM training was performed using regions extracted using the saliency detection technique from Section 3.5.1.1, which were manually labeled. Positive examples were regions that correctly correspond to signs (see Figure 28), while negative examples were regions corresponding to other structures, such as doorknobs, fire extinguishers, posters, door hinges, etc. Our training set included images of signs from two buildings on the Georgia Institute of Technology campus: 105 images from the College of Computing building and 133 images from the Klaus Advanced Computing building. Example signs are shown in Figure 28. Each image contained approximately 10 to 20 regions.

We then performed 3-fold cross validation with each classifier, the results of which are given in Table 6. While generalization to sign types not present in the training set is low, the false positive rate is also low. False positives are highly undesirable for the SLAM problem, so this is acceptable performance. Experiments are given in Section 4.3 that use these classifiers for SLAM. The classifiers used were trained on images of the same types of signs as the area mapped, but from a different part of the building (so the exact signs in the test set were not present in the training set).

Table 6: Door Sign classifier SVM cross validation results.

Dataset	Positive	Negative	Num Vectors
CoC	0.953	0.985	262
Klaus	0.774	.963	1195
CoC-Klaus	0.795	0.955	1495

Table 7: SVM confusion matrix results on trained buildings. True positive rate is listed on the left, true negative rate is listed on the right.

	CoC		Klaus	
CoC	0.954	0.960	0.321	0.984
Klaus	0.22	0.939	0.915	0.969
CoC-Klaus	0.908	0.949	0.915	0.974

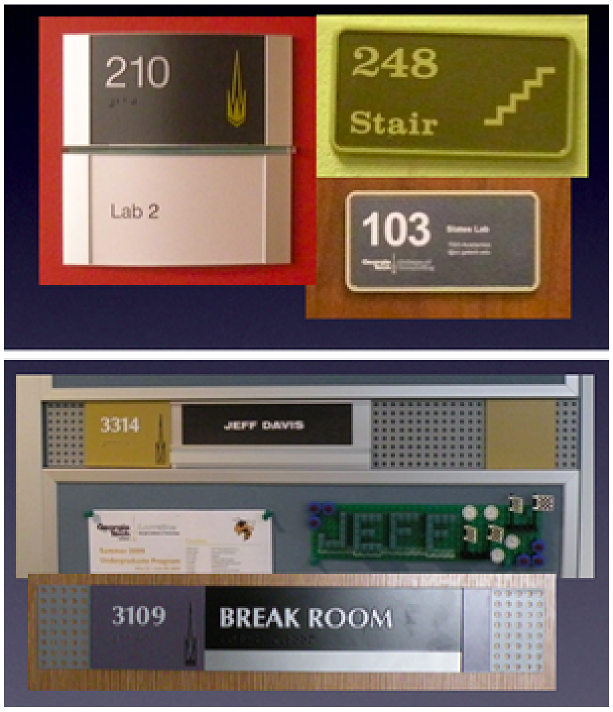


Figure 28: Examples images of signs from our classifier's training set. Signs on the top are from the College of Computing dataset, and signs on the bottom are from the Klaus data set.

3.5.3.3 Optical Character Recognition

In addition to their distinctive shape as recognized by the classifier, most door signs contain text, such as a room number or a name. While the presence of a sign is informative by itself, the text on the sign can be much more useful, as this is often a name or number that is meaningful to humans. As the text on these signs tends to be unique in a building, we can exploit the text for data association purposes if the robot is capable of reading it. Our approach was to send the image as a request to the GoogleGoggles server, which returns any text found in the image. For images taken with our robot, we found this approach to be superior to optical character recognition (OCR) software designed for images produced by a scanner, such as Tesseract [131].

While this approach generally works well for text recognition, errors can still be present in the resulting text. Sources of such errors include recognizing other symbols, graphics, or borders present on the signs as characters, and missing characters due to under segmentation of the salient regions. To match resulting strings against previously observed signs, we process the strings further by extracting numbers and attempting to match the number or text alone. If a number cannot be matched, then we compute the longest common subsequence with respect to all previously observed signs, and consider it a match if the overlap is greater than 60%.

CHAPTER IV

SLAM & SEMANTIC MAPPING

Chapter 3 introduced several techniques for segmenting point clouds and images, and detecting and recognizing structures and objects within them. These techniques all applied to single measurements or sensor frames taken from a fixed location. To build maps, we will need to handle these appropriately as time passes and the robot moves through the environment.

This Chapter provides an overview of several approaches to the Simultaneous Localization and Mapping (SLAM) problem that we've developed. We begin in Section 4.1 by describing how edge features, rather than dense point clouds, can be used for pose-graph mapping that can be more accurate and more efficient than dense alignment methods. Section 4.2 describes how to use large planar landmarks in SLAM, as observed in 3D point clouds, or as measured by lines as extracted from 2D laser scans. We then describe landmarks that correspond to objects, including visually recognized door signs in Section 4.3, and 3D clusters such as tabletop objects in Section 4.4. Finally, we present a technique for modeling relationships between landmarks in Section 4.5, and a method for detecting landmarks that have moved in Section 4.6.

4.1 3D Edge-based SLAM

Section 3.1 described our approach to detecting several types of 3D edges using RGB-D sensors. Edge features are particularly interesting because they are applicable in both textureless and textured environments. We now present a method for utilizing these for pairwise registration and pose-graph SLAM, as originally proposed in [23].

Registration methods for dense point clouds such as Iterative Closest Point (ICP) are commonly used in SLAM. However, input data must often be downsampled to achieve efficient runtimes. For many types of scenes, especially indoors, edges can capture most of the important details for registration, allowing us to use only these edge points rather than a full point cloud. This way, edge detection can be considered a way to intelligently downsample the cloud in a way that preserves an informative set of points. We will demonstrate that this approach can be faster and more accurate than some alternate approaches.

4.1.1 Edge-based Pair-wise Registration

We begin by describing and analyzing pairwise registration of 3D edges. This is the process of aligning two point clouds representing different views of the same scene that include a significant overlapping region. One of the most well-known methods of pairwise registration is the Iterative Closest Point (ICP) algorithm [11], of which many variants exist, such as point-to-plane ICP [127]. ICP can be very computationally expensive for large point clouds, which usually necessitates downsampling. Other approaches to pairwise registration of RGB-D data include extracting 2D keypoints such as SIFT [93] and back-projecting these to the 3D point cloud, generating a sparse cloud for use with pairwise registration [57, 43].

We propose to using extracted edge points, as described in Section 3.1, with the ICP algorithm as implemented in the Point Cloud Library [122]. We compared edge types to Generalized ICP [127], and also RGB-D SLAM [43], a state-of-the art SIFT keypoint based method.

As evaluation metrics, [136] introduced the relative pose error (RPE):

$$\mathbf{E}_i := (\mathbf{Q}_i^{-1}\mathbf{Q}_{i+\Delta})^{-1}(\mathbf{P}_i^{-1}\mathbf{P}_{i+\Delta}) \tag{3}$$

where $\mathbf{Q}_i \in SE(3)$ and $\mathbf{P}_i \in SE(3)$ are i-th ground truth and estimated poses respectively. When the length of camera poses is n, $m = n - \Delta$ relative pose errors are calculated over the sequence. While [136] used the root mean squared error (RMSE) by averaging over all possible time intervals Δ for the evaluation of SLAM systems, we fix $\Delta = 1$ since we are only interested in pair-wise pose errors here.

SIFT keypoints, full-resolution point clouds, and downsampled point clouds are compared with our edge points for use with ICP algorithms. For SIFT keypoint-based ICP, we used the implementation of [43] that employed SIFT keypoint and Generalized-ICP [127] and kept their best parameters for a fair comparison. They [43] originally reported their performance with SIFTGPU [165] for faster computation, but for a fair comparison we ran with SIFT (CPU only) because our edge detection does not rely on parallel processing or a GPU. For points, Generalized-ICP was also employed since it shows the state-of-the-art performance on general point clouds. As the size of original resolution point clouds from RGB-D camera is huge ($640 \times 480 = 307200$ in maximum), we downsampled the original point clouds via voxel grid filtering to evaluate how it affects its accuracy and computation time. Two different leaf sizes, 0.01 and 0.02 m, were tested for the voxel grid. For edge-based ICP, Generalized-ICP does not outperform the standard ICP [11] since locally smooth surface assumption is not valid for edge points, and thus the standard ICP is employed for edges. Except the SIFT-based ICP, all ICPs are based on the implementation of PCL. To ensure fair comparison, we set the same max correspondence distance (0.1 m) and termination criteria (50 for the maximum number of iterations and 10^{-4} for the transformation epsilon) for all tests.

Table 8: RMSE of translation, rotation, and average time in Freiburg 1 sequences

Sequence	**SIFT keypoints**	**Points**	**Points down 0.01**	**Points down 0.02**	**Occluding edges**	**RGB edges**	**HC edges**
FR1 360	14.1 ± 8.6 mm	18.6 ± 9.2 mm	21.7 ± 10.7 mm	22.9 ± 11.8 mm	23.0 ± 19.3 mm	**11.2 ± 7.2**[†] mm	20.5 ± 14.1 mm
	1.27 ± 0.84 deg	0.95 ± 0.50 deg	0.95 ± 0.48 deg	1.01 ± 0.52 deg	1.03 ± 0.81 deg	**0.55 ± 0.33**[†] deg	0.77 ± 0.40 deg
	410 ± 146 ms	14099 ± 6285 ms	3101 ± 2324 ms	913 ± 787 ms	**132 ± 112**[†] ms	321 ± 222 ms	606 ± 293 ms
FR1 desk	13.2 ± 8.6 mm	16.3 ± 8.0 mm	17.7 ± 8.9 mm	17.9 ± 8.6 mm	**7.3 ± 4.5** mm	8.6 ± 5.2 mm	10.5 ± 6.1 mm
	1.15 ± 0.78 deg	0.95 ± 0.50 deg	0.96 ± 0.51 deg	0.96 ± 0.51 deg	0.82 ± 0.48 deg	**0.70 ± 0.42** deg	0.89 ± 0.48 deg
	743 ± 223 ms	9742 ± 4646 ms	923 ± 824 ms	269 ± 265 ms	**100 ± 67** ms	427 ± 270 ms	230 ± 124 ms
FR1 desk2	13.2 ± 8.2 mm	18.8 ± 9.9 mm	20.7 ± 10.7 mm	20.7 ± 10.7 mm	**6.7 ± 3.6** mm	8.9 ± 5.4 mm	17.2 ± 13.8 mm
	1.16 ± 0.74 deg	1.12 ± 0.60 deg	1.12 ± 0.59 deg	1.12 ± 0.60 deg	0.73 ± 0.37 deg	**0.70 ± 0.43** deg	0.98 ± 0.55 deg
	620 ± 209 ms	12058 ± 6389 ms	1356 ± 1089 ms	380 ± 328 ms	**111 ± 71** ms	436 ± 268 ms	298 ± 161 ms
FR1 floor	18.1 ± 15.5 mm	16.9 ± 13.6 mm	16.9 ± 13.4 mm	16.9 ± 13.5 mm	112.9 ± 108.8 mm	**15.7 ± 15.2** mm	124.8 ± 122.2 mm
	0.76 ± 0.58 deg	0.65 ± 0.49 deg	0.65 ± 0.48 deg	0.72 ± 0.52 deg	8.16 ± 7.97 deg	**0.47 ± 0.39** deg	13.84 ± 13.66 deg
	595 ± 204 ms	6108 ± 3489 ms	474 ± 495 ms	125 ± 83 ms	**46 ± 28** ms	232 ± 129 ms	140 ± 32 ms
FR1 plant	14.8 ± 8.7 mm	14.3 ± 7.1 mm	21.6 ± 11.0 mm	21.8 ± 11.2 mm	**5.2 ± 3.0** mm	6.9 ± 3.6 mm	11.2 ± 6.3 mm
	1.04 ± 0.64 deg	0.78 ± 0.38 deg	0.87 ± 0.42 deg	0.86 ± 0.42 deg	0.62 ± 0.32 deg	**0.49 ± 0.27** deg	0.76 ± 0.38 deg
	668 ± 152 ms	10890 ± 5242 ms	2589 ± 1374 ms	851 ± 460 ms	**192 ± 132** ms	515 ± 283 ms	526 ± 234 ms
FR1 room	14.7 ± 11.4 mm	14.6 ± 6.9 mm	16.9 ± 8.3 mm	17.8 ± 9.0 mm	6.5 ± 3.9 mm	**6.2 ± 3.6** mm	10.8 ± 6.7 mm
	0.87 ± 0.59 deg	0.81 ± 0.42 deg	0.81 ± 0.42 deg	0.84 ± 0.44 deg	0.54 ± 0.27 deg	**0.48 ± 0.27** deg	0.68 ± 0.36 deg
	612 ± 201 ms	10410 ± 4482 ms	1703 ± 1353 ms	490 ± 460 ms	**117 ± 82** ms	368 ± 210 ms	340 ± 207 ms
FR1 rpy	14.2 ± 10.3 mm	12.6 ± 7.9 mm	17.5 ± 11.3 mm	17.5 ± 11.7 mm	**6.1 ± 3.4** mm	7.2 ± 4.3 mm	15.8 ± 11.4 mm
	1.05 ± 0.66 deg	1.04 ± 0.55 deg	1.12 ± 0.59 deg	1.17 ± 0.63 deg	0.73 ± 0.37 deg	**0.67 ± 0.39** deg	1.16 ± 0.62 deg
	642 ± 184 ms	13503 ± 6057 ms	2024 ± 1810 ms	677 ± 675 ms	**139 ± 90** ms	501 ± 281 ms	410 ± 251 ms
FR1 teddy	19.2 ± 13.0 mm	20.2 ± 11.4 mm	26.6 ± 14.9 mm	26.9 ± 15.1 mm	21.9 ± 20.4 mm	36.5 ± 35.3 mm	**18.3 ± 12.6** mm
	1.33 ± 0.88 deg	0.98 ± 0.58 deg	1.05 ± 0.61 deg	1.08 ± 0.63 deg	0.99 ± 0.63 deg	**0.92 ± 0.74** deg	1.00 ± 0.54 deg
	682 ± 211 ms	12864 ± 7904 ms	3647 ± 2315 ms	1194 ± 774 ms	**187 ± 116** ms	667 ± 305 ms	676 ± 322 ms
FR1 xyz	8.3 ± 4.9 mm	8.7 ± 5.1 mm	9.8 ± 5.9 mm	11.2 ± 6.2 mm	**4.3 ± 2.2** mm	4.7 ± 2.4 mm	6.3 ± 3.7 mm
	0.66 ± 0.34 deg	0.50 ± 0.24 deg	0.53 ± 0.27 deg	0.58 ± 0.30 deg	0.51 ± 0.25 deg	**0.41 ± 0.22** deg	0.55 ± 0.28 deg
	840 ± 181 ms	8358 ± 3440 ms	699 ± 462 ms	217 ± 157 ms	**96 ± 57** ms	352 ± 210 ms	195 ± 78 ms

[†] For each sequence, the first, second, and third lines represent translational, rotational, and time RMS errors, respectively. The best results are indicated in **bold** type.

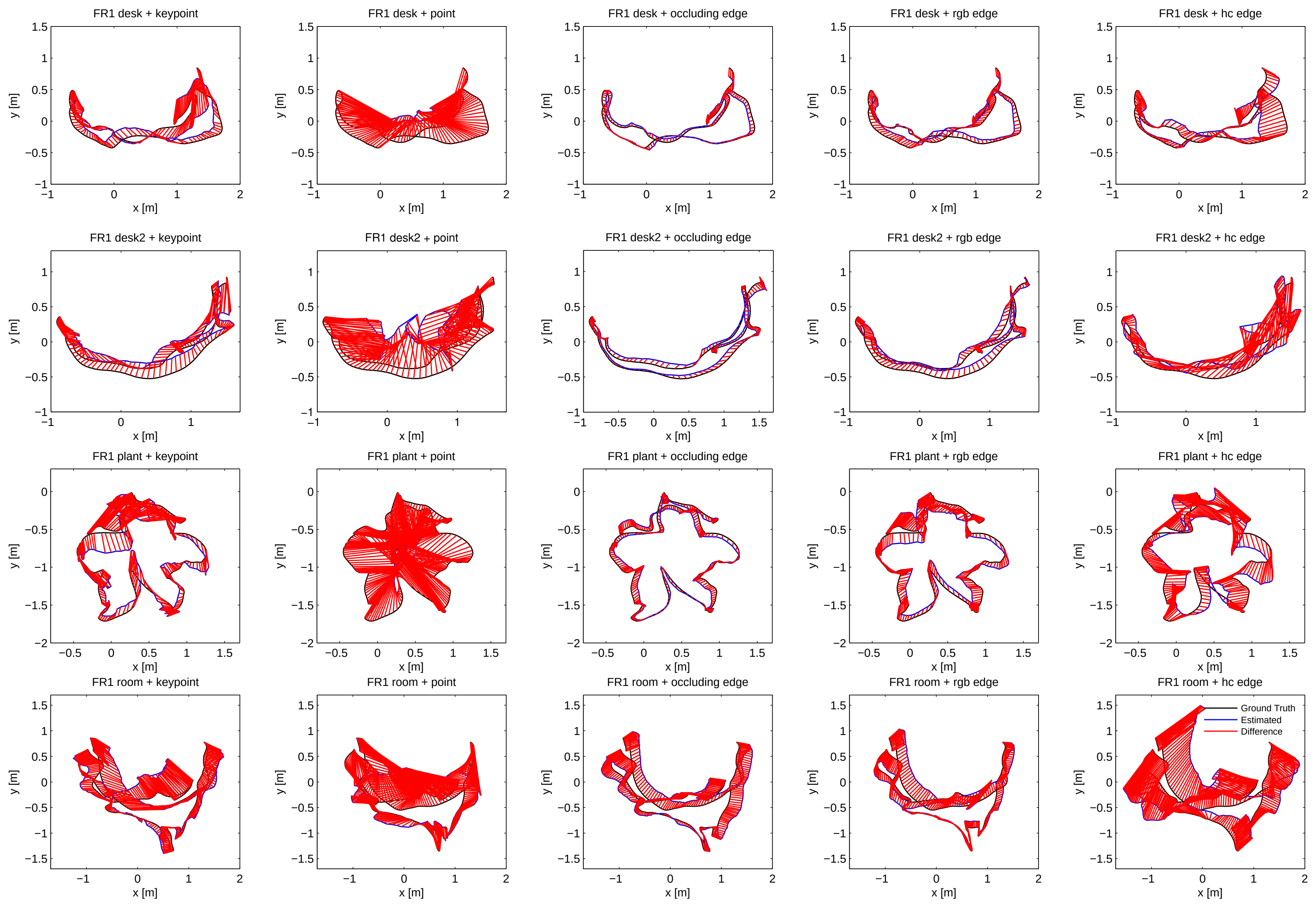


Figure 29: Plots of trajectories from pair-wise registrations.

The RMSE of translation, rotation, and average run time of the seven approaches over the nine Freiburg 1 sequences are shown in Table 8. Plots of these trajectories are shown in Figure 29. For each sequence, the first, second, and third lines show translational RMSE, rotational RMSE, and run time, respectively. The best results among the seven ICP approaches are shown in bold type. As can be seen in the table, our edge-based ICP outperforms both SIFT keypoint-based and point-based ICP. ICP with occluding and RGB edges showed the best results. Interestingly, occluding edges report smaller translational errors, while rgb edges are slightly better for rotational errors. In terms of computational cost, the ICP with occluding edges was more efficient. We found high curvature edges to be slightly worse than the other edges, but this of course depends on the input sequences. Surprisingly, using all available points does not guarantee better performance, and actually reports nearly worst performance over the nine sequences. This implies that some non-edge points are getting incorrectly matched, and increasing the error, yielding less accurate results at high run times. One exception is "FR1 floor" sequence, in which using all points reports reasonable performance compared to other approaches. The sequence is mainly composed of wooden floor scenes in an office, so there are few occluding or high curvature edges. As one would expect, this is challenging for occluding and high curvature edge-based ICP, but the texture from the wooden floor is well suited to RGB edge-based ICP, which yields the best result on this sequence. It is also worth noting that downsampling points greatly speeds up the runtime, but at the cost of a reduction in accuracy. It is even faster than the ICP with RGB edges, but the accuracy is the worst. And yet, it turns out that occluding edge-based ICP is the most efficient pair-wise registration method for the tested datasets.

It is also of interest to compare visual odometry style trajectories from the pair-wise registration to the ground truth trajectories. Although one large error in the middle of the sequence may seriously distort the shape of the trajectory, examining the accumulated errors over time can help us examine differences between these ICP methods. The estimated trajectories of “FR1 desk”, “FR1 desk2”, “FR1 plant”, and “FR1 room” are plotted with their ground truth trajectories in Fig. 29. The plot data was generated via the benchmark tool of RGB-D SLAM dataset [136], in which the ground truth and estimated trajectories are aligned via [63] since their coordinate frames may differ. Based on the plots, the trajectories from points are the worst results and do not correlate with the ground truth trajectories. The trajectories from keypoints reasonably follow the ground truth but exhibits non trivial differences. As the best results in terms of RMSE, occluding and RGB edges result in clear trajectories which are close to the ground truth . These edges are quite powerful features for visual odometry style RGB-D point cloud registration and are also very promising if they are coupled with full SLAM approaches. The high curvature edge results show bigger translational errors, but the trajectories seem reasonable results which are comparable to those of keypoints and much better than those of points.

4.1.2 Edge-based SLAM

The above pair-wise registration algorithm can also be used for SLAM problems by using a pose-graph approach. Only the sensor trajectory will be optimized, by using only constraints between poses. We use the GTSAM library in this work, and in particular we use iSAM2 [72], which allows fast incremental updates to the SAM problem.

Each time a new point cloud is received, we proceed with edge detection as described in Section 3.1 and then perform pairwise registration as described in Sectiond 4.1.1. A new pose $\mathbf{X}_n \in SE(3)$ is added to the pose graph, with a pose factor connecting $\mathbf{X}_{n-1}$ to $\mathbf{X}_n$ enforcing the relative transform between these poses as given by ICP. To improve robustness, we additionally add pose factors between $\mathbf{X}_{n-2}$ to $\mathbf{X}_n$ and $\mathbf{X}_{n-3}$ to $\mathbf{X}_n$.

In addition to adding pair-wise constraints between sequential poses, we also add "loop-closure" constraints between other poses. For each new pose $\mathbf{X}_n$, we examine poses $\mathbf{X}_0, \cdots, \mathbf{X}_{n-2}$, and compute the Euclidean distance between the sensor positions. Pairs of poses are only considered for a loop closure if the relative distance between them is less than a specified threshold, 0.2 m for this work, the angular difference is less than a specified threshold, and the pose was more than a specified number of poses in the past (we used 20 poses). If these conditions are met, we perform pairwise registration between the two poses. If the ICP algorithm converges and the fitness score is less than a given threshold, we add pose factor between these two poses. Along with the above factors to previous poses, this means each pose is connected to at most 4 other poses. Experimental results on some of the Freiburg 1 datasets are given in Section 5.5.1.

4.2 Plane Landmarks

The previous Section introduced edge-based SLAM, using features that are slightly higher level than raw sensor measurements, or corner features. This Section will introduce planar landmarks, measured using 3D sensors or 2D measurements, as can be extracted using the techniques presented in Chapter 3.

As service robots become increasingly capable and are able to perform a wider variety of tasks, we believe that new mapping systems could be developed to better support these tasks. Towards this end, we have developed a simultaneous localization and mapping (SLAM) system that uses planar surfaces as landmarks, and maps their locations and extent. We chose planar surfaces because they are prevalent in indoor environments, in the forms of walls, tables, and other surfaces.

We believe that feature-based maps are suitable for containing task-relevant information for service robots. For example, a home service robot might need to know about the locations of structures such as the kitchen table and countertops, cupboards and shelves. Structures such as walls could be used to better understand how space is structured and partitioned. We describe a SLAM system capable of creating maps of the locations and extents of planar surfaces in the environment using both 3D and 2D sensors, and present experiments analyzing the contribution of each sensing modality.

Our approach involves using multiple sensor types to measure planar landmarks. Planar surfaces can be detected in point cloud data generated by 3D sensors including tilting laser range finders, or range cameras such as the Microsoft Kinect or Asus Xtion, as described in Chapter 3. RGB-D cameras offer extremely detailed information at close ranges, the maximum range is fairly low, and the field of view is limited. This imposes limitations on the types of environments that can be mapped using this type of sensor. In order to address these limitations, our approach also makes use of 2D laser range finders, such as those produced by SICK or Hokuyo. These sensors only provide data in a plane so they are unable to measure landmarks such as tables, shelves, or walls that do not intersect their measurement plane. However, these sensors have wide fields of view, as well as long ranges. A visualization of the fields of view of these sensors is shown in Figure 31. An example of a map produced by our system is shown in Figure 30.

4.2.1 Related Work

An approach to finding horizontal surfaces such as tables was investigated previously by Donsung and Nevatia [79]. The method presented in this work measured the relative pose of tables or desks with an edge-based computer vision approach. This technique was used for finding the relative pose of these surfaces from the robot's current pose, but it was not used for building large-scale maps of surfaces.

Other previous approaches to plane mapping include Pathak *et.al.* [108][107] have removed the dependency on ICP and focus on extracted planar features. ICP based

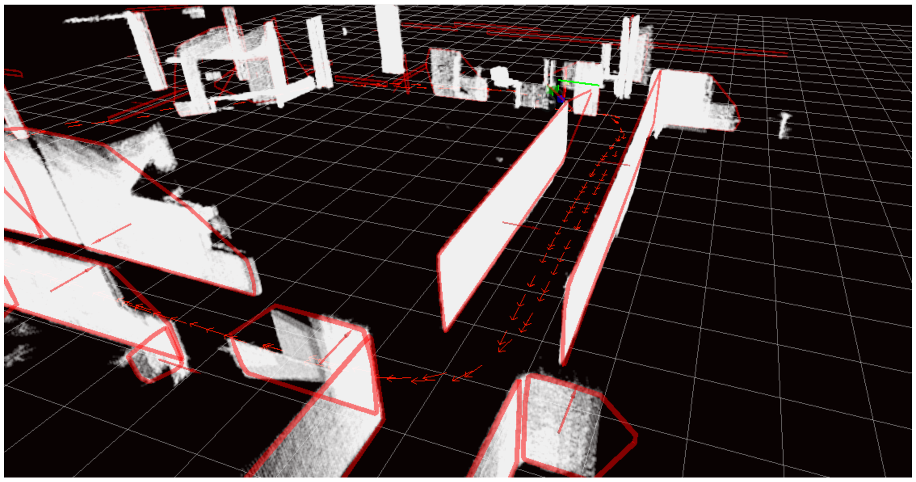


Figure 30: An example of the type of map produced by our system. Planar features are visible by the red convex hulls, and red normal vectors. The small red arrows on the ground plane show the robot's trajectory. The point clouds used to extract these measurements are shown in white, and have been rendered in the map coordinate frame by making use of the optimized poses from which they were taken.

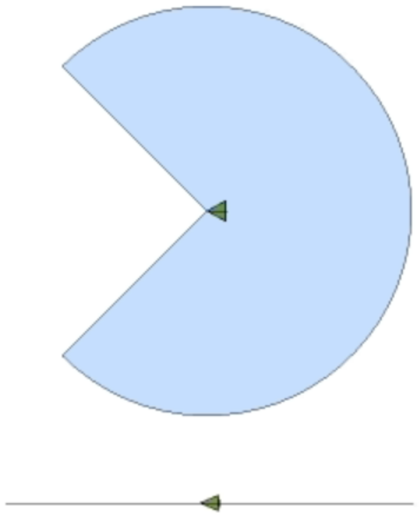


Figure 31: A visualization of the relative fields of view of our Hokuyo UTM-30LX and Asus Xtion range camera. A top-down view is shown on top, with the laser's FOV shown in blue, and the range camera's FOV shown in green. Below this, is a side view of the relative FOVs. The 2D line is the FOV of the laser scanner, while the green triangle is the FOV of the range camera.

techniques are susceptible to local minima particularly when there is insufficient overlap between point clouds. Extracted planar features also exhibit significant compression and computational advantages when compared to full point clouds. In [108], planar features are extracted at each robot pose. These planar features are matched against the features which were seen in prior poses to find correspondences. This technique does not make use of any odometry; therefore, the authors have developed a technique which enables them to sequentially associate planes which is more efficient than typical RANSAC techniques. The authors then compute the least squares rotation and translation which brings the associated planes into alignment. The rotation and translation are used to build a *pose graph* which is optimized. As opposed to building a pose graph, we take the alternative approach of maintaining the planar features as landmarks in the optimization problem, along with odometry.

Perhaps the most related planar SLAM approach is by Weingarten [160]. This work involves using planar features extracted from rotating laser-range finder data as features in an EKF-based SLAM framework, making use of the SPmodel. The features are represented by their normals, and bounded by an alpha shape to represent the extent of the feature, which can be extended incrementally as new portions are observed. We will use a similar feature type in our work, but with several key differences. First, we employ a graph-based approach to SLAM instead, which allows us to solve for the full robot trajectory rather than only the most recent pose. We have found this to be important for accurately mapping the extent of planes, as errors in past poses need to be accounted for and corrected in order to accurately map planar extents. Additionally, we can measure planar landmarks using multiple sensing modalities, by considering both 3D point cloud data as well as 2D laser measurements.

4.2.2 Planar Mapping Approach

In our initial version of this work, presented in [151], we demonstrated how surfaces could be detected in point clouds and mapped in a global map frame, however these surfaces were not used as landmarks. Instead, only 2D landmarks were used, and the surfaces did not affect the robot trajectory. Their positions were fixed with respect to the poses from which they were taken, so their positions in the map frame were dependent on the quality of the reconstructed robot trajectory [151]. We now extend this to include a new feature type, 3D planar patches. A diagram giving an overview of our system is shown in Figure 32.

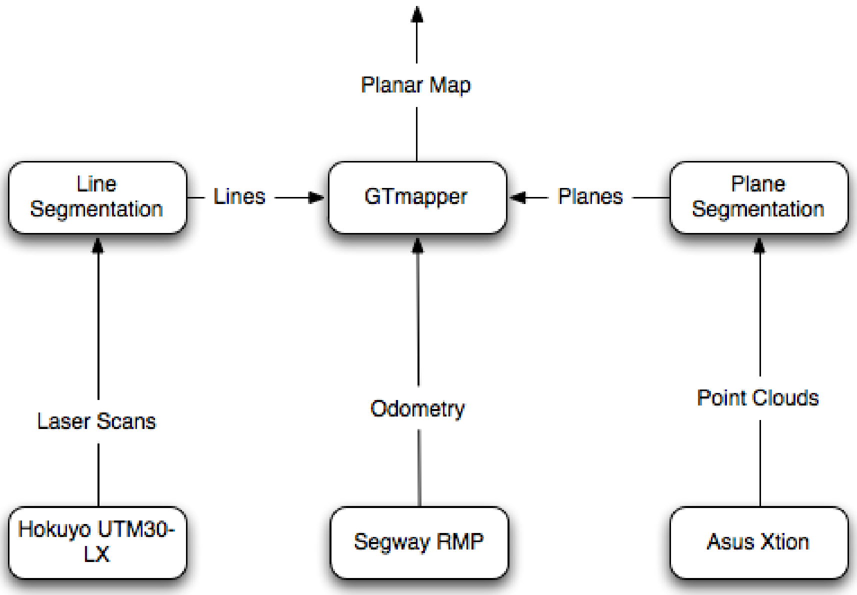


Figure 32: A system diagram, showing an overview of our approach.

4.2.2.1 Plane Representation

A plane can be represented by the well known equation:

$$ax + by + cz + d = 0$$

In this work, we make use of this representation, while additionally representing the plane's extent by calculating the convex hull of the observed points. While only the plane normal and perpendicular distance are used for to correct the robot trajectory in SLAM, it is essential to keep track of the extent of planar patches, as many coplanar surfaces can exist in indoor environments, and we would like to represent these as distinct entities. We therefore represent planes as:

$$p = \left[n, hull\right]$$

where:

$$n = \left[a, b, c, d\right]$$

and *hull* is a point cloud consisting of the vertices of the plane's convex hull. As planes are re-observed, their hulls are extended with the hull observed in the new measurements. That is, the measured hull is projected onto the newly optimized landmark's plane using its normal, and a new convex hull is calculated for the sum of the vertices in the landmark hull and the measurement's projected hull. In this way, the convex hull of a landmark can grow as additional portions of the plane are observed.

4.2.2.2 Data Association

We use a Joint Compatibility Branch and Bound (JCBB) technique for data association as in [99]. We have adapted this algorithm to work with a graph based representation instead of the EKF used in [99]. JCBB works by evaluating the joint probability over the set of interpretation trees of the measurements seen by the robot at one pose. The output of the algorithm is the most likely interpretation tree for the set of measurements. We are able to evaluate the probability of an interpretation tree quickly by marginalizing out the irrelevant portions of the graph of poses and features. The branch and bound recursion structure from the EKF formulation is used in our implementation.

4.2.2.3 Planar SLAM

In this section, we describe our method for mapping planes. Given a robot pose X_r, a transform from the map frame to the robot frame in the form of $(R, \vec{t})$, a previously observed feature in the map frame $(\vec{n}, d)$ and a measured plane $(\vec{n}_m, d_m)$, the measurement function h is given by:

$$h = \begin{pmatrix} R^T * \vec{n} \\ \langle \vec{n}, t \rangle + d \end{pmatrix} - \begin{pmatrix} \vec{n}_m \\ d_m \end{pmatrix}$$

The Jacobian with respect to the robot pose is then given by:

$$\frac{\delta h}{\delta X_r} = \begin{bmatrix} 0 & -n_a & n_b & \\ n_a & 0 & -n_c & [0] \\ -n_b & n_c & 0 & \\ 0 & 0 & 0 & \vec{n}^T \end{bmatrix}$$

The Jacobian with respect to the landmark is given by:

$$\frac{\delta h}{\delta n_{map}} = \begin{bmatrix} [R_r] & \vec{0} \\ \vec{X}_r^T & 1 \end{bmatrix}$$

Using this measurement function and its associated Jacobians, we can utilize planar normals and perpendicular distances as landmarks in our SLAM system. During optimization, the landmark poses and robot trajectory are optimized.

4.2.3 Laser Plane Measurements

Measurements which come from the laser line segmentation algorithm described in section 3.2 can be used to measure full 3D planes. This allows us to use the same formulation for measured landmarks, but with different measurement types. The laser lines measurements are under-constrained, and leave the planar landmarks with a rotational degree of freedom about the measured line, as shown in Figure 33. Landmarks that have only been measured by laser line measurements are also given a very weak prior measurement for being vertical planes, in order to constrain this degree of freedom.

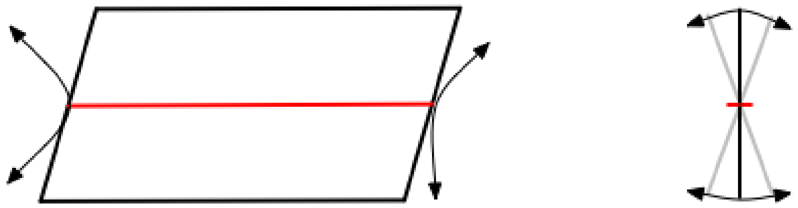


Figure 33: A drawing demonstrating a 2D measurement of a 3D plane. Such measurements are under-constrained, and have a rotational degree of freedom about the measured line.

Line measurements on planes provide only two constraints on a mapped plane feature: a range constraint and an angular constraint. The angular constraint is that the normal to the map plane forms a right angle with a vector along the measured line. Given a robot pose X_r, a transform from the map frame to the robot frame in the form of $(R, \vec{t})$, a previously observed feature in the map frame $(\vec{n}, d)$ and a measured line with endpoints p_1 and p_2 where $\vec{b} = \frac{p_1 - p_2}{|p_1 - p_2|}$ is a unit vector along the measured line and $\bar{p} = \frac{p_1 + p_2}{2}$ is the midpoint of the line, the measurement function for laser lines h is given by:

$$h = \begin{pmatrix} \left\langle R^T * \vec{n}, \vec{b} \right\rangle \\ \langle \vec{n}, \bar{p} \rangle + d \end{pmatrix}$$

The Jacobian with respect to the robot pose is then given by:

$$\frac{\delta h}{\delta X_r} = \begin{bmatrix} \vec{p}_1 & 0 \\ \bar{p} & 1 \end{bmatrix} * \begin{bmatrix} 0 & -n_a & n_b & \\ n_a & 0 & -n_c & [0] \\ -n_b & n_c & 0 & \\ 0 & 0 & 0 & \vec{n}^T \end{bmatrix}$$

The Jacobian with respect to the landmark is given by:

$$\frac{\delta h}{\delta n_{map}} = \begin{bmatrix} \vec{p}_1 & 0 \\ \bar{p} & 1 \end{bmatrix} * \begin{bmatrix} [R_r] & 0 \\ \vec{X}_r & 1 \end{bmatrix}$$

Since an entire family of planes is consistent with a given laser line measurement, additional constraints will be required to fully constrain the graph optimization. These additional constraints can come from: another laser line measurement taken from another point of view at a different height or pitch angle, a 3D plane measurement, or a weak synthetic prior factor. An evaluation of this approach using the OmniMapper is given in Section 5.5.2.

4.2.4 Discussion

We have presented an extension to our mapping system that allows for the use of planar surfaces as landmarks. The resulting maps provide the locations and extent of surfaces such as walls, tables, and counters. These landmarks can also be measured by 2D laser range finders, and maps can be produced using both sensor modalities simultaneously. We presented experimental results demonstrating the effectiveness of both sensor types, and demonstrated that both contribute to the mapping process.

We found that both sensor types have strengths and weaknesses. 2D laser scanners, which have long been popular for robot mapping systems, have long range and a wide field of view (40 meter range with 270 degree field of view, for our sensor), but do not produce 3D information. Landmarks that do not fall within their measurement plane cannot be observed. When mounted parallel to the groundplane, this means that important surfaces such as tables and shelves cannot be observed. Additionally, small amounts of clutter present in the plane can disrupt their ability to extract useful features.

Range cameras provide detailed information up close at a high frame rate, and allow us to map planes in any orientation, including horizontal surfaces such as tables or desks, as shown in Figure 71. However, these sensors also have drawbacks, including a narrow field of view and and low maximum range (58 degrees horizontal, 45 degree vertical, 0.8m - 3.5m range for our sensor). These sensors were designed for gaming and user interaction rather than mapping, so while these limitations are suitable for that application as well as mapping smaller scale areas with many features, they can be problematic for mapping larger, more open spaces.

By designing a mapping system that can utilize multiple sensing modalities, we can take advantage of the strengths of each type of sensor, and ameliorate some of their weaknesses. We have found that more information can be collected by our system when using both types of sensor, which can lead to better mapping and localization performance.

4.3 Object Landmarks: Door Signs

The previous section introduced planar landmarks for SLAM, and described how these can enable SLAM systems that use multiple sensors to concurrently measure the same landmarks. These additionally provide some semantic information about the space, as vertical planes correspond to walls, showing how space is partitioned, while horizontal planes correspond to tables and shelves, where objects of interest may occur. This section will describe how higher level objects such as door signs can be used as landmarks in SLAM, and also how text on these can help data association, and could be used to aid communication with humans.

Complex and structured landmarks such as objects have many advantages over low-level image features for semantic mapping. Low-level features suffer from viewpoint dependent imaging conditions such as boundary occlusion (where the feature is on a boundary and will appear different in subsequent frames due to motion parallax), insufficient invariance to robot motion, and specular reflections.

Data association is a problem for most SLAM algorithms operating in unstructured environments. Low-level features make use of validation gates and joint compatibility to mitigate this problem; however, the use of higher level features reduces the significance of this problem, since each landmark might have uniquely identifiable characteristics. Signs, for example, often contain text which can be read by the robot to give a unique string which could be used as an unambiguous data association cue.

Semantic mapping also offers an advantage for robots to understand task assignments given to them by human users. Non-technical users will prefer human terms for objects and locations when assigning tasks to robots instead of whatever indices or coordinates the robot uses to represent them in its memory. A text string which could be read from a sign, such as "Room 213" provides semantic information both as a label, associating "213" with the present region and denoting the place as a "room".

In this section, we present a method for using a learned object classifier in a SLAM context to provide measurements suitable for mapping. To demonstrate this, we present a classifier for recognizing door signs and a data association technique based on reading text in a graphical SLAM framework for an office environment.

4.3.1 Related Work

Castle *et. al.* has incorporated known planar objects in [19] as part of visual SLAM. This technique extracts SIFT features [93] from the image and periodically finds inliers to a homography from a canonical view of the known objects. Our work differs from this technique in that the object recognition module is not finding matches to a small set of known objects but is based on classification of objects which may not have been seen before.

Recognition and reading of door signs was proposed by Tomono *et. al.* in [142]. This work recognized and read specific door signs in a building and estimated their relative pose with respect to the robot. More recently Tomono *et. al.* have developed an object recognition scheme which they used with a mapper to build object maps [143]. In contrast to this work, our approach uses a machine learning technique for recognition which could be extended to other types of objects.

4.3.2 Door sign landmarks

Our robot makes use of the Robot Operating System (ROS) developed by Willow Garage [115] for control of the flow of data. Our technique uses three new software modules: the *laser-line-extractor*, the *door-sign-detector*, and the *mapper*. These modules will be explained in the following subsections.

4.3.2.1 Laser-line-extractor

Walls are extracted from straight lines in the laser scan, as described in Section 3.2. In Figure 34, example data is shown with extracted walls overlaid. We use a RANSAC [46] technique to extract lines from the laser data. This technique was adapted from the comparative analysis paper by Nguyen *et.al.* [102].

Pairs of points are uniformly selected from the laser point cloud (the laser range data rendered into a set of 2D points). Laser range data is analyzed to find collinear points to this line. If there are gaps in the laser line, these are used to break up

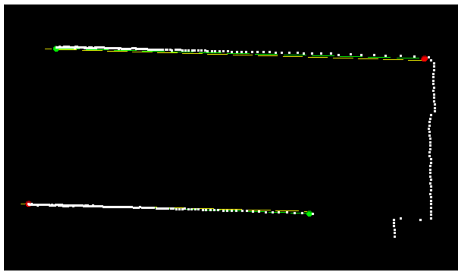


Figure 34: Laser scanner data from the robot. The structure of the hallway is recognized by the laser line extractor as two walls on either side, shown with green lines. The wall at the end of the hall is too small in this view to be recognized as a line. Raw laser points are shown in white.

one line into multiple lines. Only lines which are longer than a certain threshold are passed to the mapper as measurements.

4.3.2.2 Door-sign-detector

The door-sign-detector module makes use of the classifier described in Section 3.5.3 to recognize door signs in images taken from the robot's camera. If an image region is classified as a sign by the SVM, then a query is made from this image region to the GoogleGoggles server. If GoogleGoggles is able to read any text on the sign, then it will be returned to us in a response packet. Detected signs with decoded text are then published as measurements that can be used by the mapper. The measurements consist of the pixel location in the image of the detected region's centroid, the image patch corresponding to the detected region, and the text string returned from GoogleGoggles.

4.3.2.3 Mapper

Our SLAM implementation makes use of the GTSAM library [34]. This library represents the graph SLAM problem with a *factor graph* which relates landmarks to robot poses through factors. These factors are nonlinear measurements which are generated by the door sign detector and laser line extractor modules described above.

GTSAM builds a factor graph of nonlinear measurements. Each of these nonlinear measurements must support a *linearize* function which returns a *GaussianFactor* which is the linearization of its measurement function at the current configuration. The GaussianFactor takes an error vector and Jacobians relating the error vector to each of the variables involved in this factor. In our case, most measurements involve the pose of the robot and the pose of a landmark such as a wall or a visual feature in the environment.

4.3.2.4 Wall features

For the case of walls, our linearization gives a measurement which is based on the line's range and orientation. Note that this differs slightly from the formulation used in Section 4.2.3, as here we are measuring 2D lines directly, instead of making under-constrained measurements on 3D planes. The only errors in the line measurements are the angle of the line, and the perpendicular distance of the line to the origin. These parameters are $\eta = (\phi, \rho)$. The Jacobians which are needed by the linearization process to generate a GaussianFactor are $\frac{\delta\eta}{\delta x_r}$ and $\frac{\delta\eta}{\delta x_f}$ where x_r is the robot pose and x_f is the global landmark pose.

The Jacobians are broken down in terms of local parameterizations of the landmarks (x_o) to $\frac{\delta\eta}{\delta x_r} = \frac{\delta\eta}{\delta x_o}\frac{\delta x_o}{\delta x_r}$. The global representation x_f is transformed into the local representation x_o by transforming it into the robot's coordinate frame. Since the underlying representation of this feature is two points, this operation is a simple rigid body transformation. The Jacobian of this transformation is the derivative of these local representation x_o with respect to the robot pose x_r which is

$$\frac{\delta x_o}{\delta x_r} = \begin{bmatrix} -1 & 0 & x_{o_y} \\ 0 & -1 & -x_{o_x} \end{bmatrix}$$

The Jacobian $J_{\eta o}$ for walls is more complicated. Let $\left(x^1_{o_x}, x^1_{o_y}, x^2_{o_x}, x^2_{o_y}\right)$ be the coordinates of the endpoints of the walls. dy be the y-coordinate difference in the endpoints of the wall $x^2_{o_y} - x^1_{o_y}$, and dx be the x coordinate difference $x^2_{o_x} - x^1_{o_x}$. Also, let L be the length of the wall $L = \sqrt{dx^2 + dy^2}$.

$$J_{\eta o}^T = \frac{\delta \eta}{\delta x_o} = \begin{bmatrix} \frac{-dy}{L^2} & \frac{-dy*\left(dx*x_{ox}^2+dy*x_{oy}^2\right)}{L^3} \\ \frac{dx}{L^2} & \frac{dx*\left(dx*x_{ox}^2+xy*x_{oy}^2\right)}{L^3} \\ \frac{dy}{L^2} & \frac{dy*\left(dx*x_{ox}^1+dy*x_{oy}^1\right)}{L^3} \\ \frac{-dx}{L^2} & \frac{-dx*\left(dx*x_{ox}^1+xy*x_{oy}^1\right)}{L^3} \end{bmatrix}$$

and $\frac{\delta \eta}{\delta x_f} = \frac{\delta \eta}{\delta x_o} \frac{\delta x_o}{\delta x_f} \tilde{B}$ where $\tilde{B}$ is the binding matrix which projects corrections from the point representation x_f to the M-space representation (ϕ, ρ). In this way, the corrections will only be applied to the direction and distance of the line, but not to the endpoints themselves.

4.3.2.5 Door sign features

Measurements are made on the 3D coordinates of the back-projected image location directly. Range is recovered by finding the laser beam from the head laser which is projects most closely to the image coordinates of the sign. This technique approximates the true range which we will eventually get from the use of a 3D camera like the Kinect. This factor also incorporates an additional variable which corresponds to the transformation between the robot base and the camera. By keeping track of the transformation when each measurement is taken, we are now able to move the camera on the pan-tilt unit during a data collection run.

To implement this factor in GTSAM, we must specify an error function and the error function's derivatives in terms of all of the variables which contribute to it. The error function is the difference in the 3D position of the predicted location of the sign from the measured value given by the recognition module.

$$h(x_r, T_{bc}, x_f) = z - T_{bc}^{-1} * x_r.transform_to(x_f) \tag{4}$$

The three Jacobians of this error function are computed by combining various primitive derivative operations using the chain rule. The Jacobian of the error function with respect to the robot pose, $\frac{\delta h}{\delta x_r}$ is found as the product of the derivative of the projection operation from the camera with respect to the camera pose, times the derivative of the 3D pose composition operator with respect to its first entry. The second Jacobian, $\frac{\delta h}{\delta T_{bc}}$ is the same as the first Jacobian, with the pose composition derivative taken with respect to the second parameter. The final Jacobian, $\frac{\delta h}{\delta x_f}$ is just the derivative of the projection operation with respect to the object pose. Experimental results validating these features for use in SLAM appear in Section 5.5.3.

4.4 Object Landmarks: SLAM with Object Discovery, Modeling, and Mapping

Section 4.3 presented how to use door signs as landmarks in SLAM, but this approach required training a classifier a-priori on the objects to be used. In this section, we will instead show objects of interest can be discovered, modeled, and used as landmarks online during the SLAM process. The work in this section was originally published in [24].

One motivating reason to discover objects online is that each environment may have a unique set of objects, and a set of object models may not be available a-priori. Object discovery approaches have been proposed to address this need [74, 27] . Simultaneous Localization and Mapping (SLAM) is also required for service robot to be able to map new environments and navigate within them. We propose an approach to online object discovery and modeling, and show how to combine this with a SLAM system. The benefits are twofold: an object database is produced in addition to the map, and the detected objects are used as landmarks for SLAM, producing improved mapping results.

In contrast, maps augmented with objects confer a number of advantages for mapping the environment. Features, e.g., objects and planes, can be represented using a compact representation and provide a richer description of the environment.

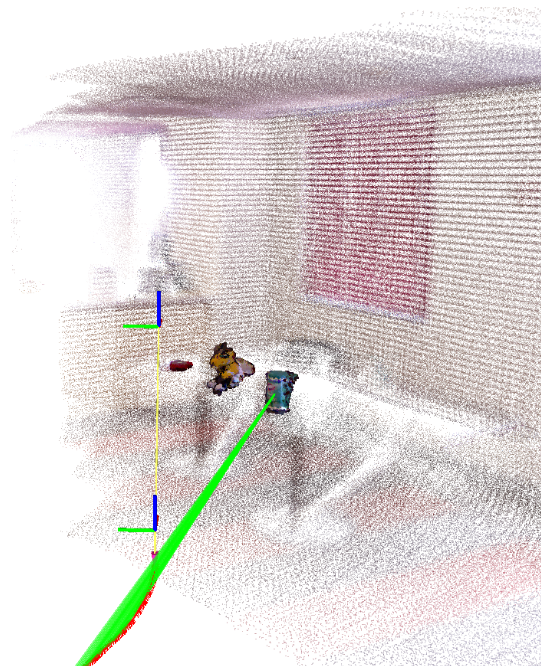


Figure 35: Snapshot of the process at one instant. The robot trajectory is shown in red. Green lines add constraints between the object landmarks and the robot poses they are seen from. Light background shows the aggregated map cloud generated using the current SLAM solution.

It is simpler to include prior constraints at the object level than at the lower feature level. The semantic information available for an object provides better cues for data association as compared to a 3D point cloud. An object would not be represented by a 3D point but rather by a 3D point cloud. Joint optimization over all the camera poses and objects is computationally cheaper than the joint optimization over all the 3D points and cameras since, as there are many fewer objects compared to the number of 3D points in a map.

Object discovery and scene understanding are also related to our approach. Karpathy et al. [74] decompose a scene into candidate segments and ranks them according to their objectness properties. Collet et al. used domain knowledge in the form of metadata and use it as constraints to generate object candidates [27]. Using RGB-D sensor, Koppula et al. [82] used graphical models capturing various image feature and contextual relationship to semantically label the point cloud with object classes and used that on a mobile robot for finding objects in a large cluttered room. Hoiem and Savarese [59] did a survey of additional recent work in the area of 3D scene understanding and 3D object recognition.

All of these approaches either learn object models in advance, or discover them in a batch process after the robot has scanned the environment and generated a 3D model or a video sequence. In contrast, our main contribution is to integrate object discovery and mapping in a SLAM framework, following an online learning paradigm. Considering the advantages of object augmented maps, we too use objects as landmarks in addition to other features.

However, we discover objects in an incremental fashion as the robot moves around in the environment. In contrast to other approaches, we do not train object detectors ahead of time, but we discover objects and train on object representation in an online manner. As the robot moves through the environment, it continuously segments the scene into non planar and planar segments (Section 3.5.1.2). All the non planar segments associated across frames are considered as object hypotheses. Every new object is represented using the 3D descriptors and matched against the other objects in the given map (Section 3.5.2). The transformation between the new object and the matched object is used as a constraint when optimizing the whole map. This is considered as a loop closure constraint when the robot sees the same object after some time and is used to optimize the complete trajectory (non sequential detection). It also helps in reducing the segmentation errors by adding a constraint between an under-segmented object part and the full object model, which results in better object modeling (recurring detection). When the new object does not match any of the previous objects, the object representation for the new object is saved for future use. Figure 35 shows a screenshot from our system that illustrates our object landmarks.

To demonstrate the effectiveness of our approach, we show results on the object discovered in the process and the resulting object augmented map generated by it

(Figure 82). We also compare the robot trajectory estimated with and without the use of object landmarks during loop closure (Figure 85).

4.4.1 Object Mapping Approach

As the robot moves around in an indoor environment, we continuously map the environment using the SLAM system described in Chapter 5. ICP along with odometry information available during each frame is used to infer the robot pose trajectory. We use the Georgia Tech Smoothing and Mapping (GTSAM) library to optimize the robot poses and landmarks [35]. In GTSAM, the SLAM problem is described as a factor graph where each factor represents a constraint between the variables (robot poses, objects, planes etc.). We add some additional factors in between the objects and pose-objects as we discover and recognize them. This is described in Section 3.5.1.2 and 3.5.2. It is jointly optimized by performing maximum a-posteriori (MAP) inference over the factor graph.

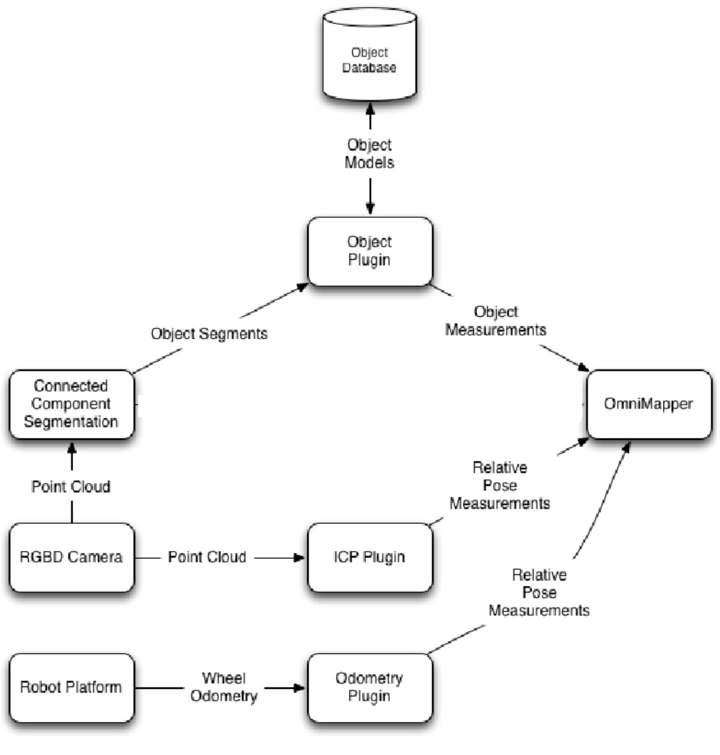


Figure 36: Flowchart of the data processing pipeline.

Each RGB-D frame is segmented using connected component segmentation to generate planes and non-planar segments as in Section 3.4. We consider all the non-planar segments as the object segments. Each non planar segment from each frame is propagated across frames to generate object hypothesis (Section 3.5.1.2). The proposed object hypothesis **O** are represented using 3D feature descriptors augmented with map information (Section 3.5.2). Object representation helps in identifying loop closure when the robot sees the same object and for producing better object models when an object part undersegmented in a few frames matches to the full object model. Object recognition is done by finding the object representation with the maximum number of inlier correspondences with the current object (Section 3.5.2). If an object does not find the minimum number of inlier correspondences with any of the saved representations, it saves the representation of the current object. This follows an online learning framework, where the robot identifies potential objects and matches to other objects in the map, if none of them matches it hypothesizes that the potential object is a new object and therefore saves the representation. Figure 36 shows the flowchart of the complete system. A detailed description of the mapping framework for integrating different sensor plugins is given in Chapter 5.

4.4.1.1 Per-Frame Segmentation

Each RGB-D cloud is segmented as described in Section 3.4. Additional filtering is done to exclude regions that are too large or too small, as in Section 3.5.1.2.

4.4.1.2 Frame-to-Frame Tracking

To track segments across sequential frames, we use a graph matching strategy similar to the method proposed by Couprie et al. [29]. Our approach works as follows. During each frame we use the segmentation result from the previous frame S_{t-1} and the segmentation result from the current frame S_t to produce the final segmentation in the current frame S_t^*. This is done by matching segments in the previous frame and the current frame. We represent each segment as the centroid of the point locations corresponding to that segment. In contrast to RGB images, depth is an important component of RGB-D images. For each segment centroid in the current frame, we find the nearest centroid in the previous frame. If the centroid in previous frame matches to more than one segment in the current frame, we choose the segment in the current frame which is the closest to the segment in the previous frame. This results in a *two-way matching* of segments. The segments rejected by this strategy are given new labels.

Since the robot is also mapping the environment and estimating its trajectory using other sources of information like odometry and ICP algorithm, we have a prior estimate of the camera pose available at each frame. We use the camera pose information during each frame to transform the current frame with respect to the previous frame according the relative transformation between the two frames. The centroids are then computed in the transformed frame. This ensures better correspondence estimation, which is geometrically consistent to the transformation.

In addition to the previous approach we also experiment with reasoning in the 3D world to propagate segment information. Since we already have the camera pose available at each frame, we transform the current frame according to the estimated pose. The segments from all the previous frames are transformed according to their respective camera poses and aggregated into one point cloud. The current transformed frame is then matched against the aggregated cloud to propagate segment labels. However, since we are matching against the whole map, a new segment in the current frame can match against a far away object because we match each segment to its nearest segment centroid in the map. To avoid this, we perform an additional check by computing the bounding box $\mathbf{B_t}$ of the current segment and the bounding boxes of the matching object $\{\mathbf{B_o}\}$ in the map. If the intersection of the two bounding boxes is small, we do not match the current segment to the same object in the map and instead give it a new label. The intersection is computed using Jaccard index which is given by Equation 5.

$$J(\mathbf{B_t}, \mathbf{B_o}) = \frac{|\mathbf{B_t} \cap \mathbf{B_o}|}{|\mathbf{B_t} \cup \mathbf{B_o}|} \tag{5}$$

We found that matching against the whole map gives better results than matching against the previous frame even though the computation time required to match against the whole map is more than matching against the previous frame. Given the set of segments having the same label, we consider the object corresponding to the set of segments to be completely modeled if no new segment is added to the same label and the object is outside the view frustum of the camera. The object point cloud is created by aggregating the segments from all matched frames transformed according to their respective camera poses.

Once the object is modeled, non-linear constraints are added between the object landmarks and the corresponding robot poses. In the SLAM system, each object $\mathbf{O}$ is represented by the centroid of its point cloud $\mathbf{C} \in \mathbb{R}^3$ (in the map frame). Given a matching object segment S having the centroid $\mathbf{C_s} \in \mathbb{R}^3$ (in the robot frame) and the corresponding robot pose $\mathbf{X} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$, the object measurement function $f(\mathbf{X}, \mathbf{O})$ is given by

$$f(\mathbf{X}, \mathbf{O}) = R\mathbf{C} + t$$

$f(\mathbf{X}, \mathbf{O})$ transforms the object centroid in the robot frame. Assuming a Gaussian noise Λ, $\mathbf{C_s}$ is given by

$$\mathbf{C_s} = f(\mathbf{X}, \mathbf{O}) + \Lambda$$

Constraints added according to the above measurement function jointly optimizes for the object location and the robot trajectory. The object recognition (Section 3.5.2) considers all such objects given by the object discovery and adds constraints between the matching objects.

Loop closure detection is handled in a separate thread, where we merge different objects given the current robot trajectory and object location estimate. Each object represented using the CSHOT descriptor (Section 3.5.2) [141] is matched against other nearby objects given the current estimated map. Each object is matched against the nearby objects whose Jaccard index (Equation 5) is non-zero when compared with the bounding box of the current object. The matching objects are merged to form one object. This step helps merging the under-segmented object parts not matched by segment propagation. We don't add object object constraint or perform optimization in this step and it only helps in improved object model generation given the current state of the SLAM solution.

4.4.2 Object Discovery, Recognition & Modeling

Object clusters are tracked across frames, and recognized and modeled using CSHOT [3, 141]. Individual frames are processed as described in Section 3.5.2.

Every new object $\mathbf{O}_i$ estimated using object discovery (Section 3.5.1.2) is matched to all the nearby objects from the current map. For every new object we compute the object representation $\{\mathbf{K}, \mathbf{D}, \mathbf{C}\}$. Since the object $\mathbf{O}_i$ is considered as a landmark, we can estimate its current location $\mathbf{C}$ and uncertainty in the estimate $\mathbf{\Sigma}$ given the current SLAM solution. We utilize this information $(\mathbf{C}, \mathbf{\Sigma})$ to find out the set of potential objects $\mathbf{\Omega}$ that are likely to match to the new object. We consider all the objects $\mathbf{\Omega}$ whose centroid lie within twice the covariance radius $\mathbf{\Sigma}$ of the new object $\mathbf{O}_i$ as the set of potential matching objects. More formally, the set of potential objects $(\mathbf{\Omega})$ is given as:

$$\mathbf{\Omega} = \forall_{\omega \in \mathbf{\Theta}} \left\{ (\mathbf{C}_\omega - \mathbf{C})^T \mathbf{\Sigma^{-1}} (\mathbf{C}_\omega - \mathbf{C}) < 2 \right\} \tag{6}$$

Here the ω and $\mathbf{C}_\omega$ represents a potential matching object and the corresponding centroid. $\mathbf{\Theta}$ represents the set of all objects. $\mathbf{C}$ and $\mathbf{\Sigma}$ represents the centroid of the new object and its covariance matrix, respectively.

The new object $\mathbf{O}_i$ is matched to a potential object $\omega \in \mathbf{\Omega}$ by finding the correspondences between keypoints $\mathbf{K}$ of the new object and the keypoints $\mathbf{K}_\omega$ of the potential object ω. The correspondences between $\mathbf{K}$ and $\mathbf{K}_\omega$ are estimated by finding the nearest neighbors in both directions resulting in a two-way match. First of all, we compute the nearest neighbors of feature descriptors $\mathbf{D}$ with respect to all the feature descriptors $\mathbf{D}_\omega$ belonging to the potential object ω. The same is done in reverse for all the descriptors $\mathbf{D}_\omega$ in the potential object with respect to $\mathbf{D}$. Let the nearest neighbor of a keypoint $k \in \mathbf{K}$ in ω is ζ. If the nearest neighbor of keypoint $\zeta \in \mathbf{K}_\omega$ in $\mathbf{O}_i$ is k, we accept that correspondence between the two objects (two-way match). Those correspondences are then further refined using geometric verification. If the number of refined correspondences is less than 12, the object is not considered as a match to the new object. We found that using 12 reduces the number of false positive matches. Among all the matching object representations, we find the object $\mathbf{O}^*$ which has the maximum number of inlier correspondences with respect to the new object $\mathbf{O}_i$.

Assuming that most of the objects don't move during the mapping time, we use the spatial context stored with the object to make the search more efficient. The representations are searched in the order of geometric distance $\|\mathbf{C} - \mathbf{C}_\omega\|$ from the new object location. In case the new object does not match to any of the stored representations, we assume that it is an unseen object and save its representation to disk.

In case we find a match to the new object, the two major use cases are as follows:

- If the matching object is seen after a certain period of time when the robot returns to the same location, we declare it as *loop closure detection.*
- If the matching object is one of the under-segmented object parts, we add constraint between the under-segmented part and the new object in order to merge them as the optimization progresses. It is used by *object refinement*

(Section 4.4.1.2) thread to generate a refined model.

Given the new object $\mathbf{O}_i$, the matching object $\mathbf{O}^*$, the centroid of the new object $\mathbf{C}$ and the centroid of the matching object $\mathbf{C}^*$, we add a non linear constraint factor between the new object and the matching object. The object-object measurement function $h(\mathbf{O}, \mathbf{O}^*)$ is given by,

$$h(\mathbf{O}_i, \mathbf{O}^*) = \mathbf{C}^* - \mathbf{C} \tag{7}$$

The ideal distance between $\mathbf{C}^*$ and $\mathbf{C}$ should be zero since both the centroids belong to the same object separated apart due to the error incurred by SLAM or object discovery. Assuming Gaussian measurement noise ρ,

$$h(\mathbf{O}_i, \mathbf{O}^*) + \rho = 0$$

In both the cases, loop closure detection or object refinement, the non linear constraint tries to minimize the gap between matching object and the new object. Minimizing the distance between the new object and the matching object gives better object models and at the same time optimizes the robot trajectory.

This approach has been evaluated using the OmniMapper framework. The results are presented in Section 5.5.4.

4.5 Modeling Relationships between landmarks with Virtual Measurements

We've presented how to use a variety of complex features, including planes (Section 4.2), door signs (Section 4.3), and 3D objects detected and modeled during SLAM 4.4. This section will describe how to improve mapping results by including our domain knowledge in the SLAM process, such as our knowledge that door signs are attached to walls, or that walls are often either parallel or perpendicular. This work was originally presented in [150].

Features detected by the robot can often be related to each other. For example, if we detect a wall using a laser scanner, and detect some visual features at a depth quite near to this wall, it is likely that these features are actually attached to the wall. Another example of a constraint between features is two walls that intersect at a corner and are perpendicular to each other. Introducing these types of constraints into our SLAM problem is a way of injecting knowledge about our environment into our map estimation, and can improve the mapping result. This type of relationship between landmarks can be thought of as a *virtual measurement* between the landmarks.

In this section, we introduce a method for including such virtual measurements between landmarks, as a way to use our domain knowledge about the environment to improve mapping results.

Geometric constraints between features have also been considered by the vision community, for example in the structure from motion problem. Szeliski and Torr considered the relationship between points and planes in [138].

4.5.1 Virtual Measurements

In order to represent relationships between landmarks, we allow for the addition of virtual measurements between two landmarks. These measurements might be to suggest that point landmarks close to walls are coplanar with that wall, or two walls that meet one another are perpendicular. Here, we demonstrate how to add these virtual measurements between a point feature and a wall feature that says the point should lie on that wall. Note that here we use 2D robot poses, and 2D wall measurements as in Section 4.3.2.4, and 2D point measurements, as opposed to 6DoF robot poses and 3D planes and 3D points.

We identify points that should be constrained to walls by using a threshold in the likelihood of the point being on the wall given its current posterior distribution. In the case of visually detected features, we can set this threshold based on the camera's uncertainty. If we would like to add a constraint that a point (x_c, y_c) should lie on a wall x_l with endpoints (x_{o_1}, y_{o_1}) and (x_{o_2}, y_{o_2}), we have the point to line distance metric:

$$\eta = \frac{-\delta y_o x_p + \delta x_o y_p + \delta y_o x_{o^1} - \delta x_o y_{o^1}}{\sqrt{\delta x_o^2 + \delta y_o^2}}$$

where $\delta x_o = x_{o^2} - x_{o^1}$ and $\delta y_o = y_{o^2} - y_{o^1}$ which are the feature coordinates of the wall. The point feature is (x_p, y_p).

The Jacobian of this measurement with respect to the point coordinates is:

$$\frac{\delta\eta}{\delta x_{point}} = \begin{bmatrix} \frac{-\delta y_o}{\sqrt{\delta x_o{}^2 + \delta y_o{}^2}} & \frac{\delta x_o}{\sqrt{\delta x_o{}^2 + \delta y_o{}^2}} \end{bmatrix}$$

The Jacobian of this measurement with respect to the wall coordinates is:

$$\frac{\delta\eta}{\delta x_{wall}} = \begin{bmatrix} \frac{-\delta y_o(\delta x_o(x_p - x_o{}_2) + \delta y_o(y_p - y_o{}_2))}{(\delta x_o^2 + \delta y_o^2)^{\frac{3}{2}}} \\ \frac{\delta x_o(\delta x_o(x_p - x_o{}_2) + \delta y_o(y_p - y_o{}_2))}{(\delta x_o^2 + \delta y_o^2)^{\frac{3}{2}}} \\ \frac{\delta y_o(\delta x_o(x_p - x_o{}_1) + \delta y_o(y_p - y_o{}_1))}{(\delta x_o^2 + \delta y_o^2)^{\frac{3}{2}}} \\ \frac{\delta x_o(\delta x_o(x_p - x_o{}_1) + \delta y_o(y_p - y_o{}_1))}{(\delta x_o^2 + \delta y_o^2)^{\frac{3}{2}}} \end{bmatrix}^T$$

These Jacobians represent the $\frac{\delta\eta}{\delta x_{wall}}$ in the global reference frame. This is now multiplied with the $\tilde{B}$ matrix for wall measurements to relate the computed δx_f to a change in the measurement space value δx_p. These Jacobians are inserted into the measurement matrix A so that they relate x_p from the wall and point landmark. The residual distance η is placed in the b vector at this measurement.

With this virtual measurement, the optimization routine will try to pull the point onto the wall, and also it will pull the wall towards the point. Note that this method does not require that the point lie exactly on the line, but instead pulls the point measurement closer to the wall measurement. This allows for objects that lie very near the wall but not actually on the wall.

4.5.2 Virtual Measurements Results

To test our system, we performed two experiments: a simulated experiment so that we can investigate our system while having the benefit of ground truth, as well as an experiment with data from our mobile robot.

4.5.2.1 Robot Platform

To test our system, we collected data with a Mobile Robotics Peoplebot, shown in Figure 37. Our robot is equipped with a SICK LMS-291 laser scanner, as well as an off-the-shelf webcam. As measurements, we detect AR ToolKit Plus [157] markers in the camera images, and extracted line features from laser scans using a Hough transform. The measurements of the AR ToolKit Plus markers give the relative pose of each marker with respect to the camera. Wheel odometry is also logged, and is used as the input for the motion model. Because our algorithm is a batch algorithm, data was logged to a file and processed offline.

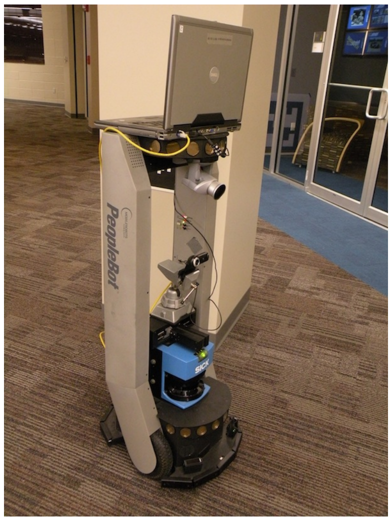


Figure 37: The robot platform used for these experiments. The camera attached to the laptop is the one that is used to detect the AR ToolKit Plus markers.

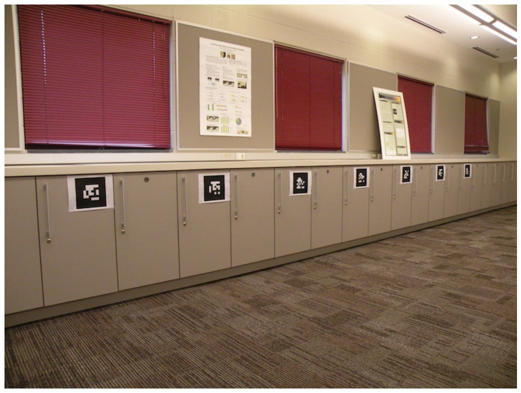


Figure 38: The experimental setup. The black and white squares on the cabinets are the AR ToolKit markers used as landmarks.

4.5.2.2 Procedure

The environment used for our experiment was a portion of a corridor in our lab, shown in Figure 38. The corridor has a long wall, on which we placed several AR ToolKit markers. The robot's laser scanner was used to detect lines corresponding to walls in the laser scan, and images from the camera were recorded. We recorded data at 15 poses, and used this as input for our SLAM system.

4.5.2.3 Results

While we do not have ground truth available for the experiment performed with our robot, we can qualitatively evaluate the results. The raw data is plotted and shown in Figure 39. We then performed our SAM optimization as described in Section 4.6.2, both with and without adding constraints between points and walls. The result with no constraints added is shown in Figure 40, and it can be seen that many of the detected landmarks do not fall on the line detected by the laser scanner. We then performed the SAM optimization again, this time adding constraints that any points detected within $0.4m$ of the wall should be on the wall. The result, shown in Figure 41, shows the points being much closer to the line, resulting in a map that appears more accurate than the result obtained without the constraints.

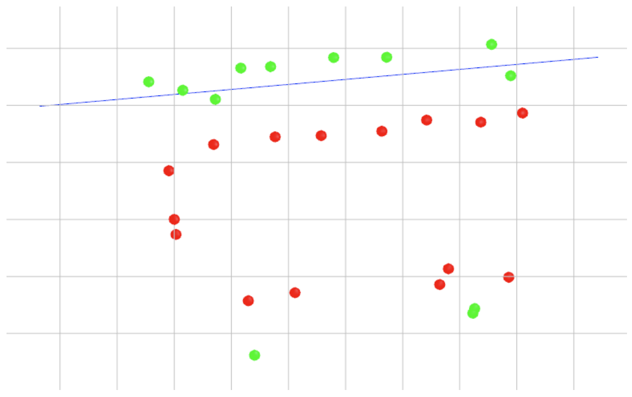


Figure 39: The initial state of the map, before optimization. Poses are shown in red (dark), static landmarks are shown in green (light), and walls are shown in blue.

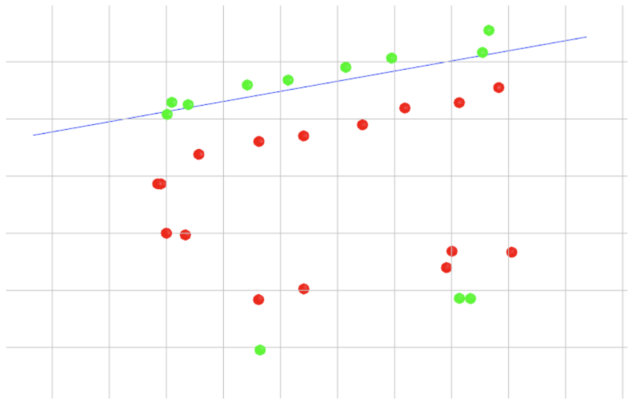


Figure 40: The map after optimization, but without using constraints between features. Poses are shown in red, static landmarks are shown in green, and walls are shown in blue.

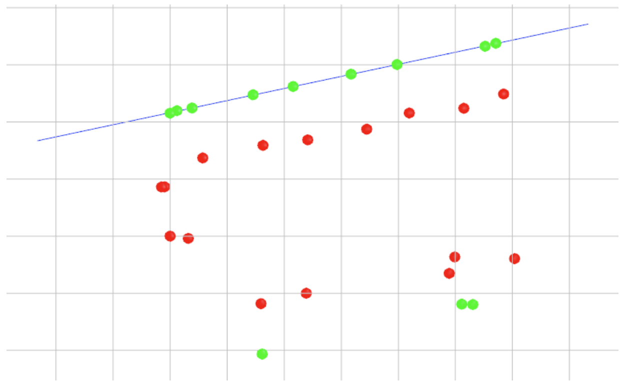


Figure 41: The map after optimization, including constraints between features. Poses are shown in red, static landmarks are shown in green, and walls are shown in blue.

Table 9: Simulated results both with and without virtual measurements, with respect to the ground truth from the simulator.

	Without VMs	With VMs
Average Pose Error:	$0.044654m$	$0.044426m$
Pose Error Variance:	$0.000665m^2$	$0.000727m^2$
Average Landmark Error:	$0.063480m$	$0.043084m$
Landmark Error Variance:	$0.001266m^2$	$0.001044m^2$

4.5.2.4 Simulated Experiment

In order to better evaluate the effects of using this type of constraint in our map estimation, we also performed an experiment in simulation. This allowed us to empirically compare the map estimation both with and without the use of constraints against the simulated ground truth. Robot poses were simulated, with Gaussian noise, according to the motion model described in Section 4.6.2. Measurements of point landmarks were also simulated, also corrupted by Gaussian noise. Wall landmarks, parameterized by their two endpoints, were also simulated, with the two endpoints being corrupted by Gaussian noise.

The environment we set up was a $6m$ by $7m$ box consisting of four walls, with landmarks placed along the walls, and some within the environment, as shown in Figure 44. The measurement noise on the line measurements was set to 0.02, while the points had a variance of 0.1. The simulated robot path was a circular trajectory with 40 poses. The experiment was performed both with and without the use of virtual measurements. The raw data is shown in Figure 42, the result without including virtual measurements is shown in Figure 43, and the result with virtual measurements is shown in Figure 44. The average error on the robot poses and landmark poses was then calculated for both cases, as was the variance on the pose error and landmark error. The results are summarized in Table 9.

While the pose error was similar between the two cases, the landmark error was reduced when virtual measurements were added. We were able to use our domain knowledge that points detected near walls using our noisy point sensor should actually be placed on the wall, which has been measured with a much less noisy wall sensor.

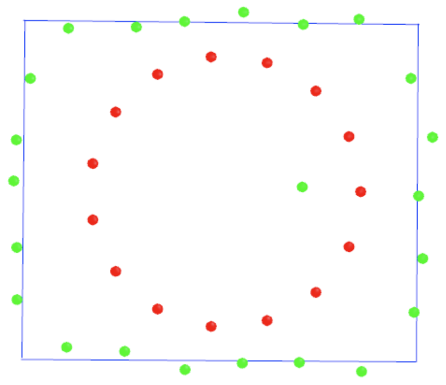


Figure 42: The initial state of our simulated experiment. Note that the robot's initial belief (shown) is that it moved in a perfect circle. The "actual" poses in the simulator were corrupted by Gaussian noise. Poses are shown in red, static landmarks are shown in green.

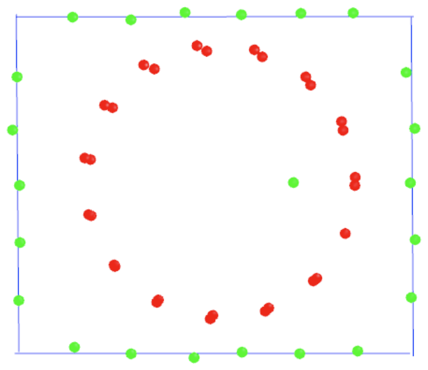


Figure 43: The simulated experiment after optimization, without inclusion of virtual measurements. Poses are shown in red, static landmarks are shown in green.

4.5.2.5 Discussion

We can consider this approach of adding constraints as a way to use our domain knowledge to improve our map estimation results. Indoor human environments are highly structured, and so they have a variety of constraints that we might wish to

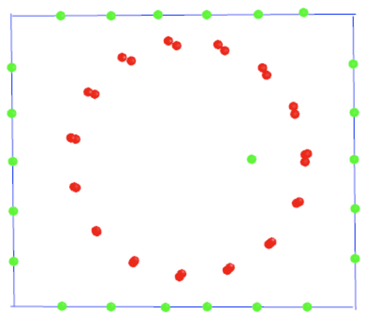


Figure 44: The simulated experiment after optimization, including constraints between features. Poses are shown in red, static landmarks are shown in green, and walls are shown in blue.

encode in our estimation problem. In addition to the collinearity constraint demonstrated here, the general approach used could be applied to add many different types of constraints to our map estimation. As we discussed in the related work, some previous work has focused on adding parallel or perpendicular constraints between walls. This type of constraint could also be added using our approach.

While this approach can improve results when we add virtual measurements that correctly relate two features, one of the shortcomings of this method is that we would introduce significant error into our maps if we were to add virtual measurements between features that should not be related. During informal testing, we tested the effect of adding a virtual measurement between features that were not actually related. As one would expect, this resulted in very high error on the map. However, due to the graph based approach used here, such mistaken virtual measurements can be undone if we re-optimize without them.

4.6 Moveable Objects

Service robots that create and use maps over long time periods will need to handle changes in the environment. Our approach uses structures and objects for localization and mapping, but the approach presented thus far does not account for objects that move. This work was originally presented in [119].

Many SLAM algorithms rely on the assumption that the environment is static, and will perform poorly or fail if mapped objects move. However, to operate in real world dynamic environments, algorithms will need to recognize moveable objects. Consider, for example, a robot that makes a map of a room, and then returns several days later. Some of the features in its map might correspond to immobile objects, such as walls, while some might correspond to objects that may have moved, such as furniture. If someone has moved some of the objects in the room that were part of the robot's map, it will be potentially catastrophic because the SLAM system will make an inconsistent map out of incompatible measurements.

4.6.1 Related Work

A common assumption in SLAM is that the environment being mapped is static. There are two main research directions which attempt to relax this assumption. One approach partitions the model into two maps; one map holds only the static landmarks and the other holds the dynamic landmarks. Hähnel *et. al.* use an Expectation Maximization (EM) based technique to split the occupancy grid map into static and dynamic maps over multiple iterations in a batch process. This technique is shown in [53] and [54] to be effective in generating useful maps in environments with moving people. Biswas *et. al.* take a finite set of snapshots of the map and employ an EM algorithm to separate the moving components to generate a map of the static environment and a series of separate maps of the dynamic objects at each snapshot. Wolf and Sukhatme [164] [163] [162] are able to separate static and dynamic maps with an online algorithm. Stachniss and Burgard developed an algorithm in [133] which identifies dynamic parts of the environment which engages a finite number of states. The map is represented with a "patch map" that identifies the alternative appearances of the portions of the map which are dynamic, like doors.

The second direction with respect to relaxing the static world assumption is to track moving objects while mapping the static landmarks. Wang *et. al.* in [158] and [159] are able to track moving objects and separate the maps in an online fashion by deferring the classification between static and dynamic objects until several laser scans can be analyzed to make this determination.

Bibby *et.al* [12] use an EM based technique over a finite time-window to perform dynamic vs. static landmark determination; however, their approach differs from ours in several important ways. First, they use a finite sliding window after which no data associations can be changed. Our approach has no such limitation as we are attempting to determine which aspects of the environment are static vs dynamic over the entire mapping run. Bibby's technique does a good job of detecting *moving* objects but it will not be able to detect *moveable* objects over longer time intervals, that might move when they aren't being observed by the robot. Additionally, Bibby addresses the static data association problem by maintaining a distribution of data associations across the sliding window; the data association decision is made permanent at the end of the window (6 steps in Bibby's implementation). An infinite sliding window would be computationally intractable in this implementation due to exponential growth of the interpretation tree; however, our approach offers an alternative solution to the static data association problem which does not suffer from finite history.

4.6.2 Moveable Object Approach

Our algorithm is based upon the Square Root SAM of Dellaert [33]. We have modified this algorithm with a per-landmark weighting term which enables discrimination between stationary and mobile landmarks to allow for more reliable localization. The remaining landmarks which have a low weight are classified as being moveable and are now tracked by the robot without influencing the robot's trajectory.

4.6.2.1 Mapping Approach

Our implementation of Square Root SAM finds the assignment for the robot trajectory and landmark positions that minimizes the least squares error in the observed measurements. As is common in the SLAM literature, our motion and measurement models assume Gaussian noise. Each adjacent pose in the robot trajectory is modeled by the motion model in equation 8.

$$x_i = f_i(x_{i-1}, u_i) + \nu_i \tag{8}$$

where $f_i(.)$ is the nonlinear motion model and u_i is the observed odometry from the robot, and ν_i is the process noise. In our case, we use a differential drive robot which has a three-dimensional pose (x_i, y_i, θ_i). The model is shown in equation 9.

$$\begin{pmatrix} \Delta x_i \\ \Delta y_i \\ \Delta \theta_i \end{pmatrix} = \begin{pmatrix} u_0 \cos(\tilde{\theta}) - u_1 \sin(\tilde{\theta}) \\ u_0 \sin(\tilde{\theta}) + u_1 \cos(\tilde{\theta}) \\ u_2 \end{pmatrix} \tag{9}$$

where u_0 is the forward motion, u_1 is the side motion, u_2 is the angular motion of the robot, and $\tilde{\theta} = \theta_{i-1} + \frac{u_2}{2}$. The measurement model determines the range and bearing to the landmarks. It has the form shown in equation 10.

$$h(x_i, l_j) = \begin{pmatrix} \sqrt{(x_i - l_j^x)^2 + (y_i - l_j^y)^2} \\ \tan^{-1}\left(\frac{(l_j^y - y_i)}{(l_j^x - x_i)}\right) - \theta_i \end{pmatrix} \tag{10}$$

The linearized least squares problem is formed from the Jacobians of these motion and measurement models as is seen in [33]. By organizing the Jacobians appropriately in matrix A and collecting the innovation of the measurements and odometry in vector b we can iteratively solve for the robot trajectory and landmarks which are stacked in Θ as seen in equation 11.

$$\Theta^* = \arg\min_{\Theta} \|A\Theta - b\|^2 \tag{11}$$

After each iteration, the Jacobians are re-linearized about the current solution Θ. The solution to this minimization problem can be found quickly by direct QR factorization of the matrix A using Householder reflectors followed by back-substitution. The source paper for this technique [33] exploits sparsity to vastly improve performance; however, we are currently using dense matrices. The optimization currently runs in approximately one second per iteration for a SAM problem of around 50 poses and 100 measurements with dense matrices. We anticipate achieving much better performance once we utilize sparse matrices.

4.6.2.2 Expectation Maximization

To establish an EM algorithm for SAM with moveable objects, we first must express the joint probability model in equation 12.

$$P(X, M, Z) = \prod_{poses} P(x_i|x_{i-1}, u_i) * \prod_{landmarks} P(z_k|x_{i_k}, l_{j_k}) \tag{12}$$

where $P(x_i|x_{i-1}, u_i)$ is the motion model and $P(z_k|x_{i_k}, l_{j_k})$ is the sensor model. We add a hidden variable ω_k to each landmark measurement, which changes the joint probability model to equation 13.

$$P(X, M, Z, \Omega) = \prod_{poses} P(x_i|x_{i-1}, u_i) * \prod_{landmarks} P(z_k|x_{i_k}, l_{j_k}, \omega_k) \tag{13}$$

With a Gaussian representation for the sensor model, this new set of parameters ω_k results in the sensor model in equation 14.

$$P(z_k|x_{i_k}, l_{j_k}, \omega_k) \propto \exp -(\omega_k((z_k - h(x_{i_k}, l_{j_k})^T \Sigma^{-1}(z_k - h(x_{i_k}, l_{j_k}))) \tag{14}$$

With the interpretation that ω_k is the likelihood that this measurement comes from a static landmark, if the landmark is not static (i.e. $\omega_k = 0$), then the measurement does not affect the joint likelihood since $P(z_k|x_{i_k}, l_{j_k}, \omega_k) = 1$ for all assignments to the robot poses and landmark positions. When the landmark is static (i.e. $\omega_k = 1$), then this weighting term makes this measurement behave like normal. Obviously, the hidden variables ω_k cannot be directly observed by the robot and must instead be estimated from multiple observations of each object. The M step selects new assignments for the ω_k to maximize the joint likelihood. Since the likelihood can be trivially maximized by setting all $\omega_k = 0$, we introduce a Lagrange multiplier to penalize setting too many moveable landmarks. The non-constant portions of the log likelihood for the measurements is now seen in equation 15.

$$l(Z, X, L, \Omega) = \sum_{measurements} (-\omega_k(\eta_k^T \Sigma_k^{-1} \eta_k)) - \lambda(1-\omega)^T(1-\omega) \tag{15}$$

where η_k is the innovation of the k-th measurement (the k-th measurement minus its predicted value). This log likelihood is maximized when

$$\frac{\delta l(Z, X, L, \Omega)}{\delta \Omega} = 0 \tag{16}$$

For each ω_k we get the equation 17

$$-(\eta_k^T \Sigma_k^{-1} \eta_k) + 2\lambda - 2\lambda\omega_k = 0 \tag{17}$$

so

$$\omega_k = 1 - \frac{\eta_k^T \Sigma_k^{-1} \eta_k}{2\lambda} \tag{18}$$

We have made the additional modification that the ω_k is not assigned per measurement, but instead since these measurements come from objects we would like to treat the objects as the things that are moveable instead of the measurements being unreliable. This is a simple modification which changes equation 15 into equation 19.

$$l(Z, X, L, \Omega) = \sum_{measurements} (-\omega_{l_k}(\eta_k^T \Sigma_k^{-1} \eta_k)) - \lambda(1-\omega)^T(1-\omega) \tag{19}$$

where ω_{l_k} is the weight of landmark l involved in the kth measurement. For each ω_{l_k} we get the equation

$$\sum_{k \in K_l} -(\eta_k^T \Sigma_k^{-1} \eta_k) + 2\lambda - 2\lambda\omega_l = 0 \tag{20}$$

so

$$\omega_l = 1 - \frac{\sum_{k \in K_l} \eta_k^T \Sigma_k^{-1} \eta_k}{2\lambda} \tag{21}$$

where K_l is the set of measurements of landmark l. The Lagrange multiplier λ can be assigned to trade off the penalty for having moveable landmarks. This was the M step of EM.

The E step of EM computes the robot trajectory and the landmark positions with the current estimates of the weighting terms ω_l. The least-squares problem solved by Square Root SAM to find the most likely map is simply the weighted least squares problem, which is to add a weighting term $\omega_l \in [0, 1]$ to each landmark measurement row i.e.

$$H * x_i + J * l_{j_i} = z_{ij} - h(x_i, l_{j_i}) \tag{22}$$

becomes

$$\omega_{l_{j_i}} * (H * x_i + J * l_{j_i}) = \omega_{l_{j_i}} * (z_{ij} - h(x_i, l_{j_i})) \tag{23}$$

where $H = \frac{\delta\nu}{\delta x_i}$ and $J = \frac{\delta\nu}{\delta l_{j_i}}$ with $\omega_{l_{j_i}}$ is the weight assigned to the landmark which we are measuring in this row. In the least squares formulation, this will have the effect of scaling the contribution of this measurement to the overall solution.

4.6.2.3 Moveable Landmark Tracking

After the terminal iteration of the EM algorithm, landmarks which have a weight factor falling below a specific threshold are removed from the SAM optimization and collected in a separate data structure. The final map is optimized once again with the moveable landmarks removed. Measurements on the dynamic landmarks are used with the final trajectory to compute global locations for the dynamic landmarks. These landmarks are now moved into a separate list of moveable landmarks where they could be referred to later to find a list of observed positions. If the moveable landmarks were tagged with semantic information, the robot would then be able to use this data structure as a candidate list of search locations for the object for a retrieval task. The robot can start with the most recently seen position for this object and then try the other places that the object has been seen in the past. Currently, the distribution of positions of the moveable landmarks is being represented as a list of observed locations.

4.6.3 Moveable Object Results

To verify the performance of our algorithm, we performed a series of experiments with moveable landmarks in our office environment.

4.6.3.1 Robot Platform

To test our system, we collected data with a Mobile Robotics Peoplebot. Our robot is equipped with a SICK LMS-291 laser scanner, as well as a Logitech webcam. As measurements, we detect ARToolKit Plus [157] markers in the camera images. The ARToolKit was used in these experiments because the emphasis here is on the detection of static and mobile landmarks. Having landmarks with trivial data association helps us focus on the key contribution of this paper; however, there is no loss of generality and natural landmarks will be considered in future work. The resulting measurements give the relative pose of each marker with respect to the camera. Laser data was also logged, but is only used for visualization purposes. Wheel odometry is also logged, and is used as the input for the motion model.

4.6.3.2 Experimental Setup

The environment used for our experiment was a portion of our lab, consisting of two student offices and the corridors connecting them. ARToolKit markers were placed throughout the environment to serve as landmarks. Some of the markers were pinned to the walls, while others were held by moveable frames which facilitated their movement during the experiments.

4.6.3.3 Procedure

Our first experiments were to move the robot in a circle in one of our student offices measuring 4.5 meters on a side. This office had 12 ARToolKit markers pinned to the walls. In addition to these static landmarks, this office had a total of 10 moveable landmarks which were placed upon the desks and shelves. We performed tests of our implementation of the standard SAM algorithm by moving the robot in this office to collect measurements of landmarks without moving them during the test run. The next experiment was to move the landmarks to a second location within this same cubicle midway through the data collection. We performed a larger scale experiment in which the robot moved between both of our group's student offices. Each of these offices is 4.5 meters on a side, and they are separated by about 12 meters of corridors. We left the 12 ARToolKit markers in the first office from the small scale experiment, and placed 9 markers in the second office. Additionally, we pinned 8 markers to the walls in the corridor between our offices. Several data sets were collected in this setup with varying numbers of moveable landmarks. In each run, the robot was moved in the first office so that each landmark was observed multiple times and then the robot was driven down the corridor. While the robot was being moved down the corridor, the moveable landmarks were transferred from the starting office to random locations in the second office. The robot was then maneuvered in the second office so that each landmark was observed multiple times and then it was driven back to the first office. The robot was driven in the first office in a few loops and was finally placed as close as possible to its starting location. Each test run of this type featured similar trajectories, but always the moveable landmarks were placed in arbitrary positions in the two offices.

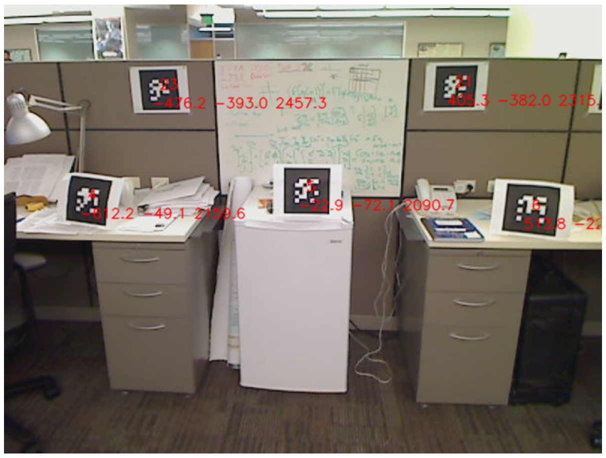


Figure 45: One of the images used as part of our experiments, as taken from the robot.

The logs were used as input for our algorithm. While the ARToolKit Plus measurements provide the relative pose of the landmark in Cartesian space with respect to the camera, we instead converted this to a range and bearing measurement. As described in section 4.6.2, the algorithm first considered all measurements, and iteratively adjusted the weights to determine which landmarks were moveable, and which were static. The algorithm iterated until the state converged and the updates were below a specified threshold. Once a stable configuration had been found, the landmarks which had weights below a certain threshold were removed from the SAM problem and were tracked separately. At this point, a final map was generated.

4.6.3.4 Results

We present the longest test run in detail. The initial state of the problem can be seen in Figure 46. This corresponds to the raw odometry and sensor measurements. The robot starts out in the upper rightmost corner of the left office, facing up in the image. The initial iteration of the EM algorithm will have 1.0 in each ω_k, so each landmark is initially assumed to be static. The SAM optimization is iterated until convergence with these parameters, resulting in a very poor quality map as can be seen in Figure 47. It is apparent in this figure that the moveable landmarks

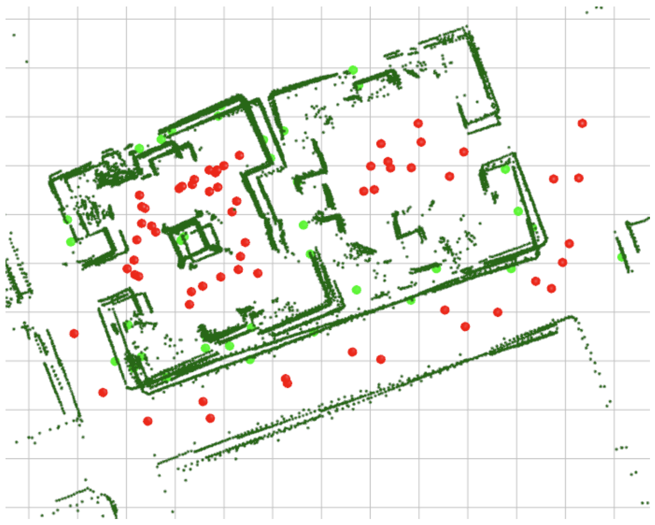


Figure 46: The initial state of the map, before optimization. Poses are shown in red, static landmarks are shown in green. The dark green points are laser scans, and are for visualization purposes only.

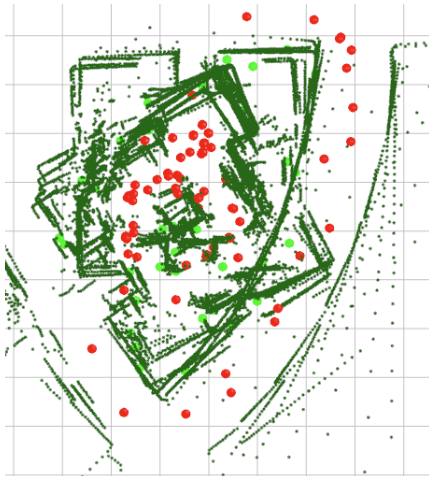


Figure 47: The resulting map with all measurements (including moveable objects) prior to the first weight assignment.

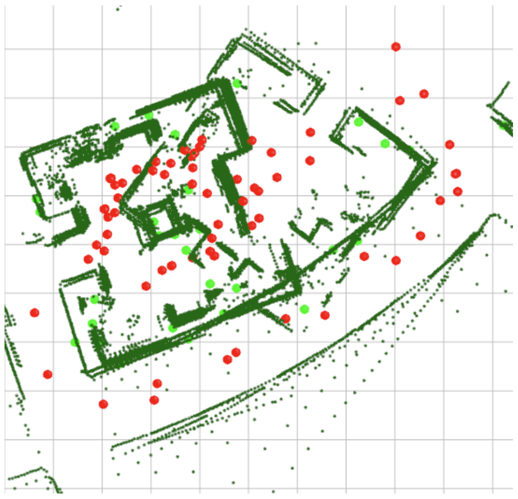


Figure 48: The resulting map after two iterations of the EM algorithm.

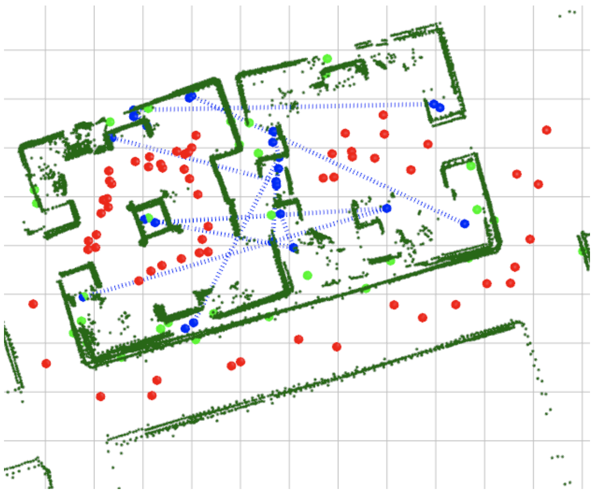


Figure 49: The resulting map after all iterations and thresholding to exclude moveable objects. Poses are shown in red, static landmarks are shown in green, and moveable landmarks are shown in blue, connected by a blue dotted line showing their movement. The dark green points are laser scans, and are for visualization purposes only.

have resulted in incorrect loop closures causing the two offices to intersect almost completely. After two iterations of the EM algorithm the map can be seen in Figure 48. This map is clearly better than the initial state; with two additional iterations of the EM algorithm the low weight landmarks can be thresholded to generate the final map in Figure 49. In this particular run, there were 10 moveable landmarks, 6 of which were detected as moveable. No static landmarks were mistakenly detected as moveable. The remaining 4 moveable landmarks which were mistakenly classified as static were only observed in one of the two offices. Without the observation of the landmark in its second position, the algorithm cannot determine that the landmark had moved. The missing observations can be explained by our use of a webcam and some poor lighting, or the missing landmarks do not appear with a front aspect view which ARToolKit Plus can detect. We have performed two additional test runs of this length and four runs of the single office test, with similar results.

We performed an additional test run where we ignore measurements from the static landmarks. This test was generated by running one of our normal test runs with measurements suppressed from markers that we know to have been static. In this test run, the EM operation was able to correctly identify all of the landmarks as moveable. The final output appears the same as the initial odometry solution. This makes sense because the SAM problem has no measurements between landmarks which affect the trajectory since all of the landmarks had moved.

4.6.4 Discussion

While our algorithm worked well for the scenarios we tested, there are some cases that could be more problematic for this technique. Here we provide a brief discussion of some scenarios in which this technique might not perform well.

One case that could cause difficulties for this technique is if several landmarks moved together in a coherent manner. For example, if many landmarks were to move one meter in the same direction, the algorithm might not factor these out, as it might be more likely that the error could be ascribed to poor odometry. This case is of particular interest, because this is what would happen if we were to track many features that were part of a single object. As the object moved, all of the features would undergo the same rigid body transform, moving them in a coherent manner. A possible solution to this would be to track the entire object, instead of individual features on the object.

Because we determine which objects moved based on their residuals, if a landmark were to move only by a small amount, the residual would probably not be large enough to cause it to be excluded. In this case, it would be used for our algorithm, and would add error to the final map and trajectory.

Another potentially troubling scenario is if most or all of the landmarks we detect are moveable. In our experiment, temporary visual features were used as landmarks for the robot. In practice, some more permanent architectural landmarks should be chosen in addition to potentially moveable landmarks, such as walls.

Also, data association was not an issue because ARToolKit markers were used, and so different markers that appeared near each other were never considered as the same landmark. If we were to use natural features as landmarks, the moveable landmark detection problem becomes much more complex. However, if most of the landmarks we see are static and only a few have moved, a similar technique should still be applicable.

CHAPTER V

OMNIMAPPER: A MODULAR MULTI-MODAL MAPPING SYSTEM

Chapter 4 introduced several new feature types and mapping approaches. These were developed for separate experiments, each one requiring a slightly different configuration of the mapper, which in turn required significant developer effort to create. This motivated us to develop a modular mapping framework to more easily configure a mappers for different feature types, sensors, and robot platforms as needed for a specific application. We call this system "OmniMapper", and describe it in this chapter. Much of this work was originally published in [155].

The literature includes a variety of Simultaneous Localization and Mapping (SLAM) approaches targeted at platforms with different sensors, environments, CPU budgets, and applications. It is clear from this diverse set of solutions that SLAM is not a problem with a one-size-fits-all solution. Applications have varying requirements, and so require different solutions. Many of the existing solutions have limited applicability, and are not open. This implies that doing comparative experiments using different types of sensors, features or data association methods becomes a challenge. Concurrently utilizing data from multiple sensors, such as laser scanners and RGB-D sensors, is usually not supported.

Many factors can determine what type of mapping system is suitable for a given application. Environments vary in their appearance, structure, and texture; this can be seen in Figure 50. Due to this variety, a given type of feature may be better suited

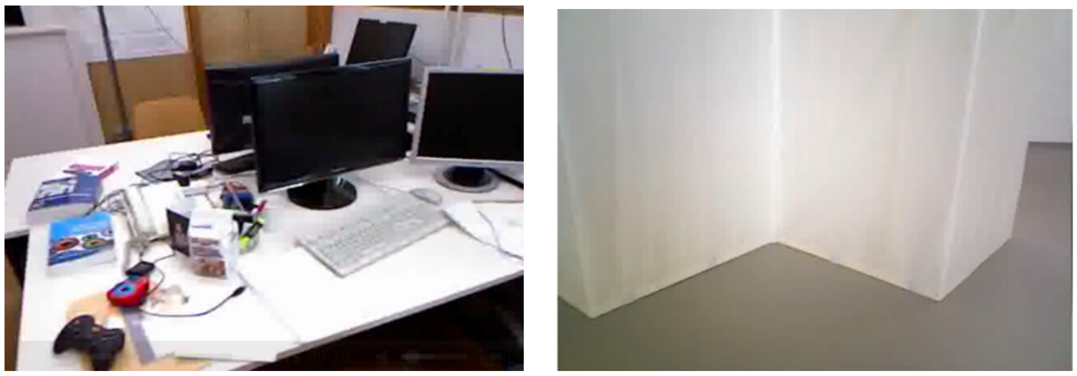
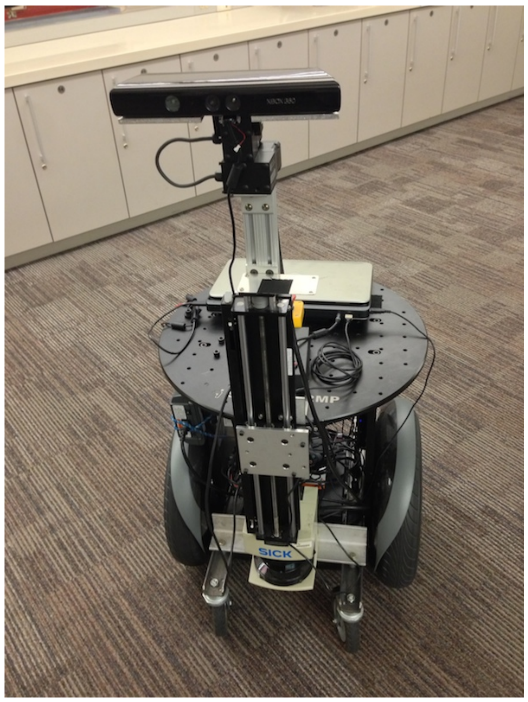

(a) (b)

Figure 50: Examples of environments with different properties. Left: an environment with a lot of visual texture. Right: an environment with little visual texture. Images are from the TUM RGB-D Dataset.

to some environments than others. Robotic platforms may also be equipped with different types of sensors and have varying computational capabilities, which impose requirements on what algorithms and feature types can be used. Different mapping applications also require differing information to be stored in a map, ranging from basic occupancy information to semantic maps that include place names and object information.

Robotic platforms also have differ in their computational capabilities, which impose requirements on what algorithms and feature types can be used (Figure 51). For example, a large service robot may be equipped with multiple state-of-the-art PCs, enabling computationally expensive features and map representations to be used. However, a small unmanned aerial vehicle (UAV) may only be equipped with a limited CPU, necessitating usage of efficient feature types.

Mapping applications also require differing information to be stored in a map. For example, some robots may only need to be able to navigate in a collision-free manner between predefined waypoints, for which a 2D occupancy-grid style map may


(a)

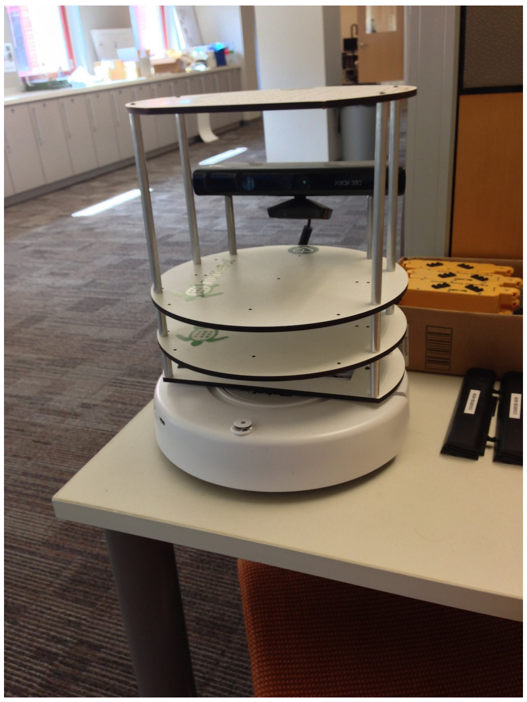
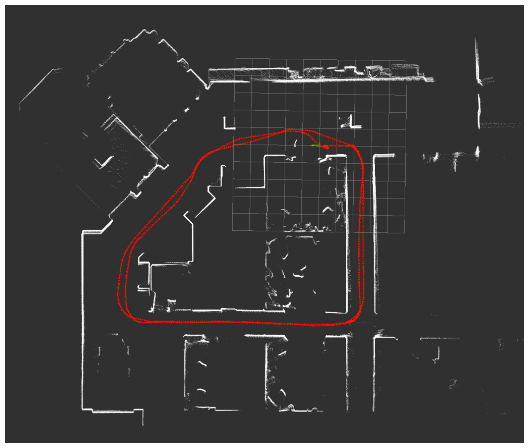

(b)

Figure 51: Examples of robots with different sensors, configurations, and computational abilities. Left: The Jeeves service robot platform, equipped with a laser scanner and an RGB-D sensor, and a laptop with a multi-core CPU. Right: a Turtlebot robot with an RGB-D sensor and a low-end netbook.


(a)

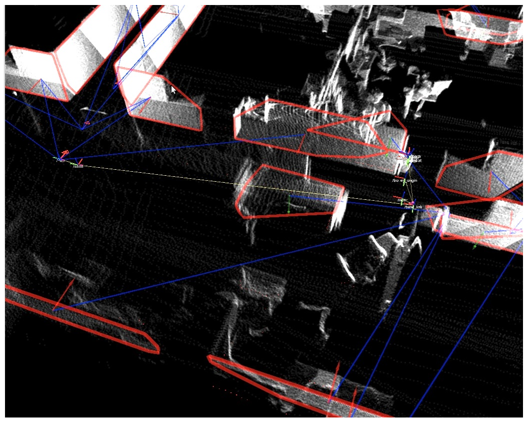

(b)

Figure 52: Examples of different map representations taken in the same environment. Left: an occupancy grid map. Right: a map that includes planar surfaces such as walls and tables.

suffice. On the other hand, a home service robot such as the Willow Garage PR2 may additionally require higher level semantic knowledge of the environment, such as names of places, object locations, and more. See Figure 50 for some examples of different environments, platforms, and map types.

Most current approaches to SLAM support only one or two particular types of measurements. For example, the Iterative Closest Point (ICP) algorithm is popular for pose-graph solutions to the SLAM problem. Kinect Fusion [100] uses 3D Point-to-Plane ICP matching against a Truncated Signed Distance Function (TSDF) map representation. RGB-D SLAM [44] uses SURF features back-projected to a depth image from an RGB-D Camera. For some environments and applications, it can be advantageous to use multiple sensors concurrently. For example, may service robots are equipped with both laser scanners and RGB-D sensors, and these complement each other well for mapping purposes. Lasers typically have long range and wide field of view, while providing only 2D range measurements. RGB-D sensors have limited field of view and range, but provide rich sensor information. Combining these can lead to improved maps. In Section 4.2, we showed the benefits of using 2D line measurements from laser range finder concurrently with 3D planar landmarks extracted from an RGB-D sensor to take advantage of the different qualities of these two sensing modalities.

To better address the diverse range of SLAM problems and to enable easy multimodal mapping, we propose the OmniMapper, a modular multimodal mapping framework. Rather than providing a single mapping system with a fixed choice of features, we have developed a modular plugin-based software framework that can be easily configured to use different feature types, sensors, and produce different types of maps. Much as toolkits such as the Point Cloud Library (PCL) [122] and OpenCV

[15] provide perceptual tools for addressing a wide range of 3D perception and computer vision problems, and “MoveIt!” [137] provides a framework for motion planning, we aim to provide tools for addressing the set of SLAM problems. By providing a *framework* rather than a library, we additionally impose some structure on what a solution should look like, while maintaining generality. The plugin architecture allows plugins to operate in one or more threads, so as to take advantage of modern multi-core CPUs. The key contribution is an approach to integrating data from multiple sensors and measurement types into a single factor graph, so all measurements can be considered jointly.

The remainder of this section is structured as follows: Section 5.1 describes some related works. Section 5.2 describes the overall approach to the SLAM problem, and how multiple sensor types are used concurrently. Section 5.3 provides details on the OmniMapper software framework, which implements this approach. Mapping results are provided in Section 5.4 for four different platforms, each with different sensors, operating in different environments and different configurations of the tools provided by our framework.

5.1 Related Work

Several libraries and toolkits for SLAM are currently available. The GTSAM library [34] provides tools for constructing factor graph based maps, and performing optimization and inference on these models, and is used in our work. Other recent graph optimization techniques include Toro [51], g^2o [88], and the Ceres Solver [2]. These tools are SLAM “back-ends”, focusing on performing the optimization required for the SLAM problem, but providing limited or no support for feature extraction, data association, or interfacing with sensors.

Mapping approaches with multiple sensors and multiple feature types have also been proposed, some of which are highlighted here. Henry *et. al* [57] introduced RGB-D Mapping, which combined visual features extracted from the RGB camera with Iterative Closest Point (ICP) on the dense point cloud from the depth camera. Visual information and laser information have also been combined, as in the work of Newman *et. al* [101] where visual information was used for loop closure detection, and laser was used for building a geometric map. Visual measurements have also been combined with inertial measurement unit (IMU) data in previous work, such as the work of Leutenegger *et. al* [89]. These approaches were designed with specific platforms, sensors, or environments in mind. In contrast to these approaches, we provide a mapping framework in which different sensors and feature types can be changed or combined freely depending on the application.

The approach presented here builds on our previous work on SLAM and multi-sensory mapping. We demonstrated the usage of laser scanners, robot odometry, and visually detected door signs in [120]. Planar landmarks observed from both RGB-D cameras and 2D laser range finders were combined in [154], along with robot odometry. In this work, we build upon and extend these mapping approaches by making a more modular framework from mapping, allowing these approaches to be used interchangeably or combined with other techniques.

5.2 Mapping Approach

This section provides a brief overview of some of the key concepts used in our mapping system. The underlying factor graph representation and optimization tools are described, followed by our approach to building such factor graphs from a variety of different measurement sources concurrently.

5.2.1 Factor Graphs & Optimization

To optimize the robot trajectory and landmark positions, we employ a factor graph-based approach called Square Root Smoothing And Mapping ($\sqrt{SAM}$) [36]. In $\sqrt{SAM}$, the full robot trajectory is optimized, as opposed to a filtering approach in which only the most recent pose is present in the solution, and past poses are marginalized out. More specifically, we use the iSAM2 approach, which enables efficient incremental updates to the SLAM problem thereby enabling online operation. The details of iSAM2 are available in [72].

A key advantage of factor graph based SLAM is that a variety of different factor types can be used jointly in one optimization problem. Factor graphs are composed of variables, which for the SLAM problem represent entities in the world such as robot poses or landmarks, and factors, which provide probabilistic measurements that relate to one or more variables. Using this formulation enables us to jointly optimize robot trajectories and a variety of landmark types, and to use many different measurement types that relate these entities. Our framework provides a means to construct such factor graphs using a variety of measurement and sensor types.

5.2.2 Common Measurement Types

Many different types of measurements can be used for SLAM. We will highlight several of the important measurement types used by our system here.

5.2.2.1 Relative Pose Measurements

Relative Pose Measurements are measurements between two poses, and so are represented in the factor graph as binary factors that impose some constraint between these two poses. In our approach, there is a 1:1 correspondence between poses and timestamps, so this means each relative pose measurement has a corresponding time interval. A relative pose measurement consists of a spatial displacement $T \in SE(3)$, two time points that define the time interval, $time\_start \in \mathbb{R}$ and $time\_end \in \mathbb{R}$, and a noise model C that represents the uncertainty associated with this measurement.

Many sensors can produce this type of measurement. For example, scan matching approaches can provide relative pose measurements between sequential scans, or match to a scan from a previous pose to trigger a loop closure. Robot odometry and visual odometry also produce this type of measurement. A motion model such as a velocity motion model (or even a null motion model representing the expected absence of motion) can also be used to produce such measurements. Another example would be place-recognition measurements such as FABMap [30], which would add measurements indicating that two poses from different times correspond to the same place.

Our approach makes an additional distinction between types of relative pose measurements. Some measurement sources, such as robot odometry and inertial sensors, operate at a high rate, and are suitable for generating relative pose measurements for arbitrary time intervals by using interpolation if the exact requested timestamps are not available. We call this type of measurement source a *continuous pose measurement source*. On the other hand, measurement sources such as ICP or other scan matching approaches can only produce such measurements for certain time intervals where sensor measurements are available. As we will see in Section 5.3.4, we will treat these types of measurements sources differently, as continuous pose measurement sources will be used to link together all sequential poses.

5.2.2.2 Absolute Pose Measurements

Some sensors, such as GPS, external tracking systems such as Vicon systems, and celestial navigation approaches provide absolute position measurements in some fixed coordinate frame. If this external coordinate frame is used as the map frame, or a known transform between the two frames exists, such measurements can be used to directly measure the robot pose at a given timestamp. In contrast to relative pose measurements, these measurements are unary measurement factors, in that they only impose a constraint on a single pose in the graph.

5.2.2.3 Landmark Measurements

Another common type of measurement is a landmark measurement, in which the robot's sensors observe an external entity, usually called a feature or a landmark. The observed landmark observation then becomes part of the SLAM problem, and the landmark's pose is the optimized jointly with the trajectory. Multiple observations of the same landmark from different poses provide information about the robot pose, in addition to updated the landmark pose.

Several landmark types are supported including 2D and 3D points, 6DOF poses, and visual measurements from cameras. We have also introduced some new types, such as measurements of planes (e.g. walls), as described in detail in our previous work [154]. Such measurements typically occur in individual sensor frames, and as such will occur at a single point in time. These measurements are typically realized through binary factors between a pose and a landmark.

5.2.3 Trajectory Management

OmniMapper is designed to track the movement of a single entity such as a sensor or robot, as it moves through an environment. Note that this could be a robot with multiple sensors, but we choose one point on the robot to be the *base frame*. It is the trajectory of this *base frame* pose that will be tracked, and all measurements are transformed to this point.

While the mapper is operating, it adds poses to the trajectory periodically as it moves through space and time. A pose is added for each timestamp at which a measurement is provided, so there is a $1:1$ mapping between pose symbols and timestamps. We require that our trajectory is fully connected (multiple disjoint trajectories are not supported), so there must be at least one relative pose measurement between each sequential pose. Multiple measurements from different sources between sequential poses are supported – for example, we may have both odometric information and a relative pose measurement from ICP between the same pair of poses.

We make a distinction between continuous pose sources (high-rate measurements sources such as odometry, motion models, or inertial sensors) and other measurement types. It is generally not necessary or beneficial to add a factor between each odometric measurement, but instead to aggregate the odometric estimate into single factors for a longer time duration. Instead, the other measurement sources used in the system determine the timestamps at which poses should be created by requesting a pose to be created to correspond to a given timestamp. Given a sequence of timestamps (and therefore poses), these can be linked together by invoking all available continuous poses sources to create a relative pose measurement between these timestamps / poses. This ensures that our trajectory remains connected, and that the information from each continuous pose source is used for the entire trajectory.

Many sensors operate at different rates, and features may take different amounts of time to compute. This can lead to measurements arrive with out-of-order timestamps. This can be a challenge to avoid double-counting information from continuous pose measurement sources such as robot odometry. Two possible solutions to this issue include measurement queueing and pose splicing. We provide a motivating example prior to describing these.

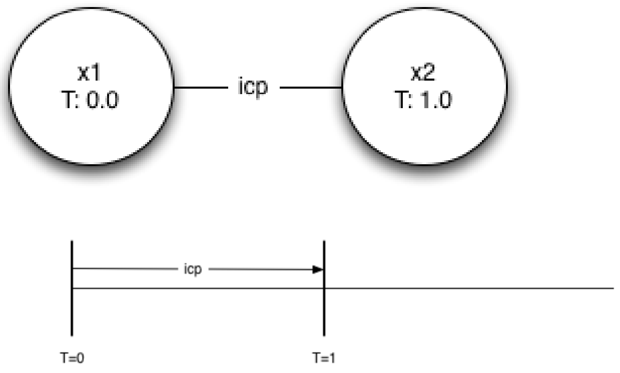


Figure 53: An example of a relative pose measurements between two time stamps. Top: a representation of the graph. Bottom: a timeline representing the time intervals.

5.2.3.1 Trajectory Management Example

Consider an example where we have three measurement sources: Iterative Closest Point (ICP) measurements, which give a relative pose measurements between two scans, and planar landmark measurements, which is a feature-based technique, and robot odometry, which is a reliable pose source. We begin by adding an ICP measurement between two consecutive scans occurring at times 0.0 and 1.0, resulting in the graph and timeline shown in Figure 53.

We can then continue by invoking our odometry-based reliable pose source, and adding a relative pose measurement between these two times that summarizes the odometry estimate for the same time interval of 0.0 to 1.0. The result is shown in Figure 54.

We next receive a landmark measurement at time 0.75 – note that this occurs in the middle of the previously measured time interval of 0.0 to 1.0. If we simply add this pose to our graph and again invoke our reliable pose measurement source (robot odometry, in this case) to add measurements between time 0.0 to 0.75 and

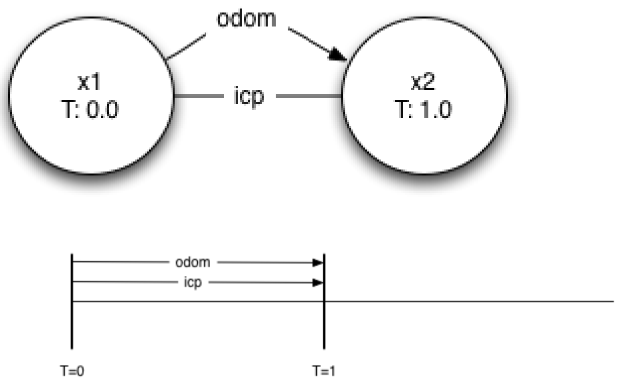


Figure 54: An example of a relative pose measurements between two time stamps after adding odometric information. Top: a representation of the graph. Bottom: a timeline representing the time intervals.

also 0.75 to 1.0, we will have double counted the odometric information – it has already been added as a constraint between times 0.0 and time 1.0. An example of this being handled incorrectly is shown in Figure 55. To handle this correctly, we must either remove the previously added or odometry factor between 0.0 and 1.0, or delay addition of factors from reliable pose sources until all measurement sources have completed. A corrected example is shown in Figure 56.

As you can see, managing a variety of measurement sources can be challenging. Two possible solutions to this issue include measurement queueing and pose splicing.

5.2.3.2 Measurement Queueing

Measurement queueing involves delaying the generation of relative pose measurements and the addition of measurements to the map for some time window, to allow for lower rate sensors and features to be computed. After the time window has passed, the measurements are added to the factor graph, and relative pose measurements from continuous pose sources are generated. This approach is currently implemented in the OmniMapper framework.

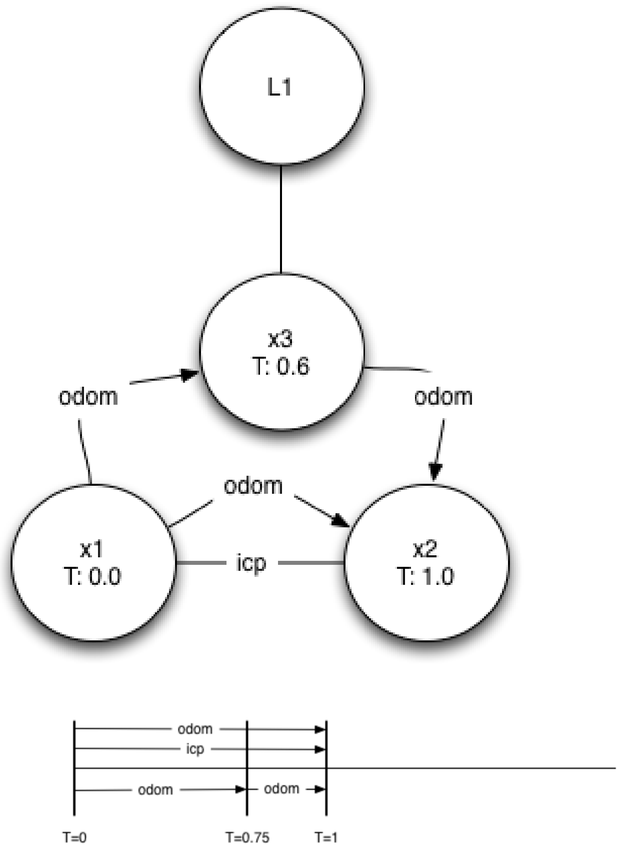


Figure 55: An example of a relative pose measurement, a landmark measurement, and an incorrectly handled relative pose measurements from odometry, because the odometric information has been double counted. Top: a representation of the graph. Bottom: a timeline representing the time intervals.

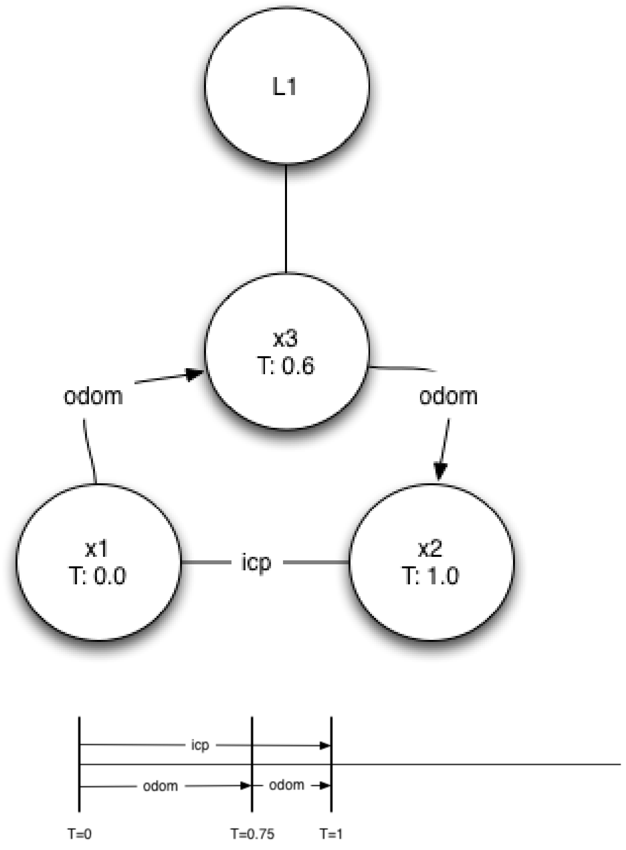


Figure 56: An example of a relative pose measurement, a landmark measurement, and correctly handled relative pose measurements from odometry, with no double counting. Top: a representation of the graph. Bottom: a timeline representing the time intervals.

5.2.3.3 Pose Splicing

An alternative approach would be to perform pose-splicing. In this approach, measurements are added immediately, and if an out-of-order measurement arrives in a time interval that has already been added to the factor graph, the previously generated factors from continuous pose sources are removed, and new ones are added to include the new timestamp, without double-counting information. This approach is not currently implemented in the OmniMapper, but may be added as future work.

5.3 Software Architecture

The mapping approach described in Section 5.2 has been implemented in C++, and is available as open source under the BSD license from `http://www.omnimapper.org`. This section provides some details on this implementation.

Our framework integrates with and builds upon several other open-source software projects, including GTSAM [35], PCL [122], and ROS [114]. Detailed dependency information is available on our web site.

5.3.1 Optimization & SLAM Backend

SLAM systems are often regarded as being composed of two parts: a front-end and a back-end. The front-end provides feature extraction, interfaces to sensors, data association, and more, while the back-end performs the optimization to actually solve the SLAM problem. Our contribution is primarily on the front-end, though we go a bit further than traditional SLAM front-ends in that we also include support for addition of semantic information, as well as interactive semantic mapping applications as in [153]. We make use of the Georgia Tech Smoothing and Mapping Library (GTSAM) [35] as our SLAM back-end. While alternate SLAM back-ends such as g2o or Google Ceres could in theory be used in our framework, only a GTSAM factor-graph based back-end has been implemented as of this writing.

5.3.2 OmniMapperBase

OmniMapperBase is the core of the mapping system. This class is responsible for building the actual factor graph used to solve our Smoothing and Mapping (SAM) problem, and interacting with GTSAM and its iSAM implementation which performs the optimization. OmniMapperBase does not interact directly with any sensors; measurements are provided to OmniMapperBase via a measurement plugin architecture. The main functionality of this class is managing the trajectory and measurements as described in Section 5.2.3.

Figure 57 provides a visual overview of how plugins interact with the mapper. Plugins process timestamped sensor data to generate measurements. These could be landmark measurements, relative pose measurements, or any of the types described in Section 5.2.2. The plugins are responsible for generating measurement factors, which internally provide measurements that involve one or more symbols in the map. Given a timestamp, a plugin can request the pose symbol associated with that time using the $getSymbolAtTime$ function, which returns a $gtsam :: Symbol$. This symbol is then used to construct any factors that reference the pose at this time. Once the plugin has constructed a factor, it can add this to the SLAM problem by calling the $addFactor$ function, which will add it to the SLAM problem. Plugins can operate in parallel, executing in one or more threads each, allowing the system to take advantage of modern multi-core CPUs.

As described in Section 5.2.3, the OmniMapperBase interacts with timestamped data. Different applications may require different methods of getting the current time. For example, when performing real-time SLAM, the system clock is generally the source of time, but processing logged data such as a ROS bag file may require

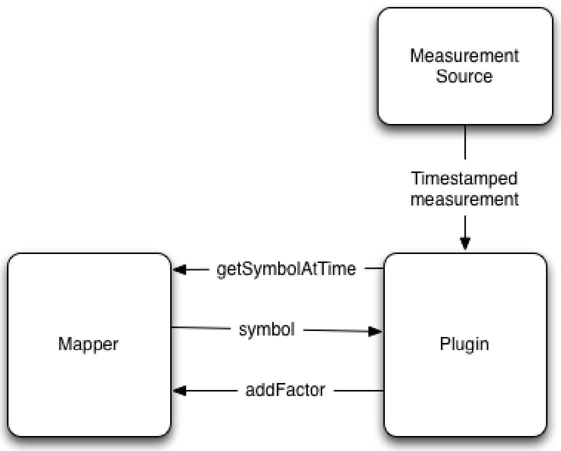


Figure 57: An overview of the plugin-based architecture.

using a different clock for playback at a different rate, or to enable pausing of the logged file. Both use-cases are supported in the current implementation.

5.3.2.1 Inversion of Control & Dependency Injection

OmniMapper makes use of the Inversion of Control *IoC* paradigm, and the related Dependency Injection design pattern. The key concept of the inversion of control paradigm is that rather than a user calling library functions from their code, instead a framework using IoC will define some interfaces that the user implements, and the framework then calls the user code. Dependency Injection is a related design pattern by which this is achieved. This design pattern involves the definition of some class or object that has one or more dependencies, for which interfaces are defined. Multiple implementations that adhere to these interfaces may exist, enabling them to be bound (or injected) and modified at runtime. More details about these techniques are available in an article by Martin Fowler [50].

The main benefit we gain from this design pattern is the ability to load plugins dynamically at runtime – that is, we can easily add / remove plugins either at load time or while the mapper is operating. This is achieved by defining a set of interfaces for the different types of plugins and modules used in the mapper, for which various implementations can be defined. As will be explained in detail, this design pattern is used throughout the mapper, including to enable modularity of pose plugins, measurement plugins, output & visualization methods, time sources / clocks, lookups of transforms from sensor to the base frame, and functions for different mechanisms to trigger measurement updates.

5.3.3 Measurement Plugins

This section highlights a selection of measurement plugins currently available for use with our mapping system. Relative pose measurements are supported for building pose-graphs, and several landmark measurements are supported for feature-based SLAM. Both types can be used simultaneously. As described in Section 5.2.3, the mapping system tracks the trajectory of a single entity, so we require all sensor measurements to be transformed to the *base frame*. Measurement plugins can be provided with a $getSensorToBaseFunctor$, which can return either a static or dynamic transform.

5.3.3.1 3D Iterative Closest Point

Since its introduction in [11], the Iterative Closest Point algorithm and various other versions and adaptations have been popular for registration of point clouds. Our mapping framework includes a plugin for registration of point clouds using either the standard Iterative Closest Point algorithm [11], or the Generalized Iterative Closest Point (GICP) [127] algorithm. Both implementations used are from the Point Cloud Library (PCL) [122].

The plugin can be used to register full point clouds, uniformly downsampled point clouds, or other point sets. We have additionally used this plugin on 3D edge features, as demonstrated in our previous work [23] on edge-based registration and SLAM.

Loop closure with the 3D ICP plugin is handled by a thread which finds the closest place along the robots trajectory which was observed far enough in the past so as to favor large loops. If this closest place is sufficiently close to the robot's current location, then the ICP or GICP routine is used to find the transformation between the two keyframe clouds from these locations. If the ICP match is successful, then the resulting relative pose is added to the factor graph as an additional constraint. This additional constraint corrects any error which has accumulated since the robot visited this place before, thereby *closing the loop*.

5.3.3.2 Laser Scan Matching

Laser range finders such as the SICK LMS series and Hokuyo UTM-30LX are popular sensors for use on mobile robots. We developed a 2D scan matching plugin built upon Censi's Canonical Scan Matching approach [20], which can provide accurate pose registration between laser scans.

As with the 3D ICP plugin described above, in addition to being used for matching sequential scans, the scan matching plugin can also produce loop closures by attempting to match scans from nearby robot poses. If a match is successful, loop closure constraints are added.

5.3.3.3 Planar Surface Landmarks

Large planar surfaces can serve as excellent landmarks for indoor mapping systems. In our previous work [154], we described how such landmarks can be used in SLAM. To fully constrain platform motion, at least three planes in general position (not co-planar) must be matched between frames, which is not always possible. As such, these types of measurements are best used in multimodal mapping, in conjunction with other measurement types. The planar landmarks used can either be extracted from RGB-D data using the organized connected component approach described in Section 3.4, orfrom unorganized point clouds as in Section 3.3.

5.3.3.4 Object-Based Landmarks

Object-level mapping can also be useful both for SLAM and semantic mapping purposes. In previous work, we demonstrated the usage of recognized objects such as door-signs in SLAM [120]. The framework supports this type of measurement as an object plugin, which adds landmark measurements between a robot pose and either a 6D pose or a 3D point representing the object location, depending on whether or not orientation information is measured.

5.3.4 Pose Plugins

5.3.4.1 Robot Odometry

Many robotic platforms can report relative position changes using odometric information computed from wheel encoders. ROS [114] is often used on such mobile robots, and publish their position in an odometric coordinate frame using the TF library available with ROS. We have developed an odometry pose plugin for using odometric information published via TF. Such measurements are treated as a continuous pose source, so odometric information will be summarized in factors between each sequential pose / timestamp as described in Section 5.2.3.

5.3.4.2 Visual Odometry

Visual odometry systems operate by generating relative pose measurements between sequential images. Many such systems operate at relatively high framerates (30 Hz), and are suitable for using in the same way as robot odometry. The odometry plugin described above can be employed with visual odometry systems that publish such results using the TF system in ROS. We have tested this using the TUM Dense Visual Odometry library by Kerl *et. al* [77].

5.3.4.3 Motion Models

In the absence of a continuous pose source, it may be helpful to use a motion-model as a pose plugin, to provide a prior on the type of motion that may occur. The toolkit provides a *null motion model*, which adds a weak prior indicating that no motion occurred between two timestamps. Such factors can serve to keep the trajectory connected in the absence of other measurements, for example if a plugin such as ICP did not converge, and was thus unable to provide a relative pose measurement for a given time interval.

5.3.5 Output Plugins

Output plugins are called after each map optimization, and are used to produce various types of outputs from the mapping system, including visualizations, and publishing or saving various kinds of map data.

5.3.5.1 Visualization Plugins

Visualizations are supported for ROS's RViz Visualization tool, as well as PCL based visualizations. As with the other plugins, these plugins are configurable, and can publish appropriate visualizations based on the content of the map and the user's requirements. Visualizations such as the robot / sensor trajectory, loop closure constraints, planar landmark locations and boundaries, and object landmark locations are supported.

5.3.5.2 Map Output Plugins

Many types of maps may be useful for different applications. One supported type of map output is an aggregate point cloud, in which many point clouds taken at different poses are aggregated in the map frame, using the optimized robot trajectory. Another type of useful map output is a mesh model of the environment. This is supported via a plugin that creates a Truncated Signed Distance Function (TSDF) volume [31] using a set of organized point clouds, and generates a mesh using the Marching Cubes algorithm [92]. Stephen Miller's CPU_TSDF library is used in this plugin [97].

5.3.6 Adding User Defined Plugins

The framework has been designed with extensibility in mind, so users can easily add new plugins of any of the types described above. To create a measurement plugin, all that is required is for a user to create a measurement of one of the types described in Section 5.2.2 and provides these to the mapper base as described in Section 5.3.2. Output plugins need only to install a callback to receive the resulting optimized factor graph.

5.4 Example Mapping Applications

Our mapping system has been applied to a variety of different platforms, environments, and feature types. Several such applications are highlighted here, to demonstrate both the flexibility of the framework and its applicability to multimodal mapping.

5.4.1 3D Semantic Mapping for Service Robots

Service robots can be equipped with a wide variety of sensors. We have employed our system on the Jeeves service robot, which is a Segway RMP base, equipped with a SICK LMS-291 Laser Range Finder and a Microsoft Kinect RGB-D Camera mounted on a pan-tilt unit. Several different feature types have been used with this robot, including 2D scan matching, 3D ICP, and planar landmarks. The mapping system has additionally been used to support semantic mapping applications as in [153]. Figure 58 shows the platform used, as well as a system diagram and an example map.

5.4.2 Large Scale 3D Mapping

We have also used OmniMapper in larger scale operation on an iRobot PackBot platform in a number of test facilities. The experimental platform, system diagram, and example map output can be seen in Figure 59. In this configuration, the robot collects 3D point clouds from a Velodyne 32E laser scanner and uses the 3D Iterative Closest Point plugin described in Section 5.3.3.1 to build a map of the environment. In this example, the robot maps a large loop (0.3 km) around the corridors of an office building.

5.4.3 Mapping with a handheld RGB-D Camera

Creating maps using hand-held RGB-D Cameras has recently become popular, with approaches such as RGB-D Mapping [57], RGB-D SLAM [44], and Kinect Fusion [100]. Our framework has also been applied to this domain using several feature types. The example map shown here uses planar landmarks, edge ICP, downsampled cloud ICP, and dense visual odometry. The sensor, mapper configuration, and example map are shown in Figure 60.

5.4.4 Scan Matching for Mobile Robots in Industrial Environments

SLAM systems based on matching scans from laser range finders are very popular, and have been widely reported on in the literature. An example of such a system is

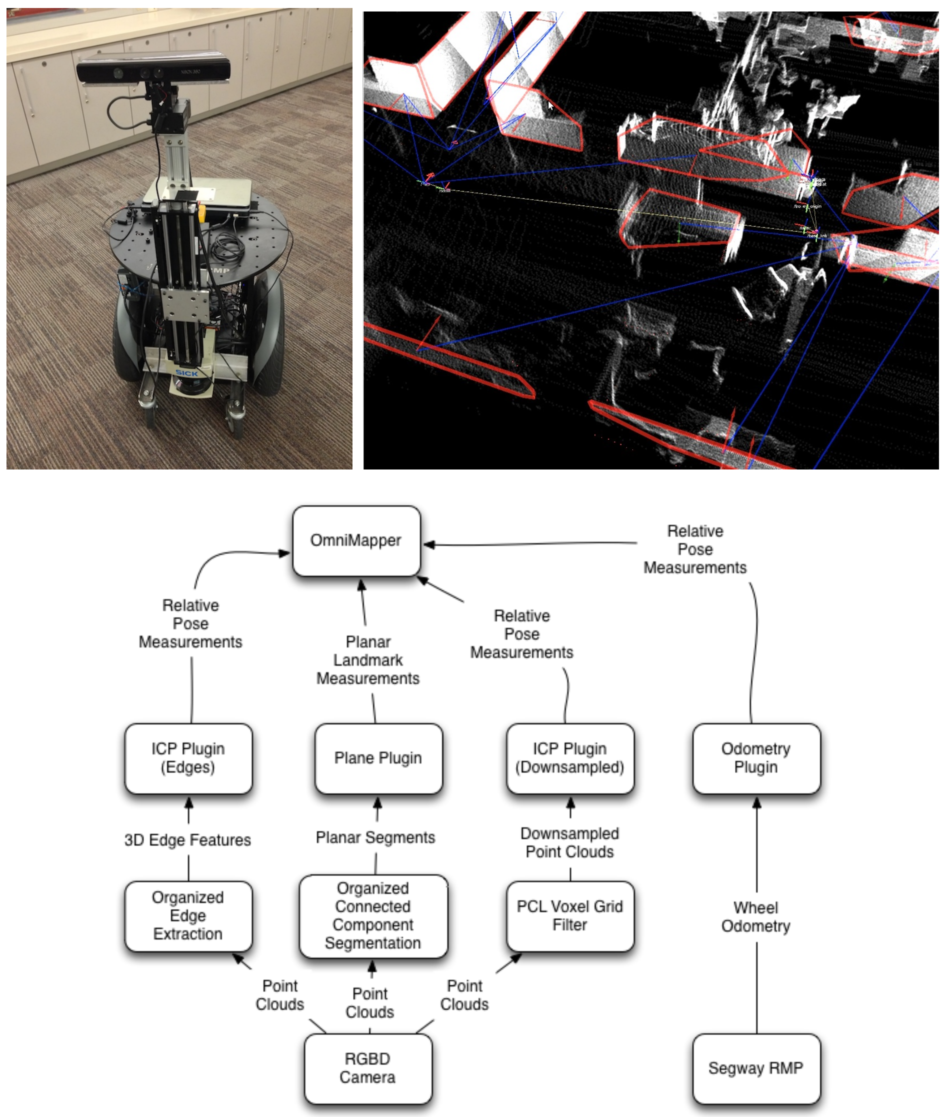


Figure 58: Top Left: The Jeeves service robot platform. Top Right: An example map produced by our system with this platform and configuration. Bottom: A system diagram representing the configuration and plugins used.

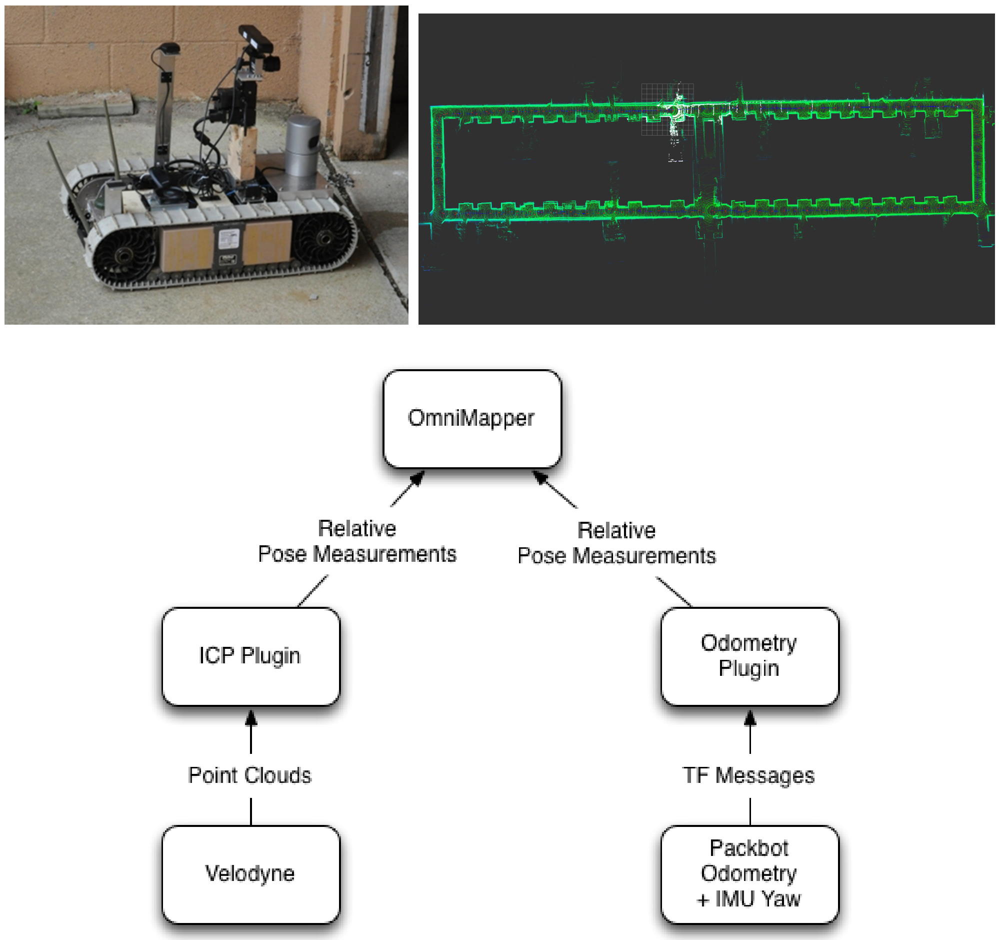

OmniMapper
Relative Pose Measurements
Relative Pose Measurements
ICP Plugin
Odometry Plugin
Point Clouds
TF Messages
Velodyne
Packbot Odometry + IMU Yaw

Figure 59: Top Left: An iRobot PackBot equipped with a Velodyne 32E 3D laser scanner, a MicroStrain GX2 IMU, and an onboard computer. Top Right: An example map produced by our system with this platform and configuration. Bottom: A system diagram representing the configuration and plugins used.

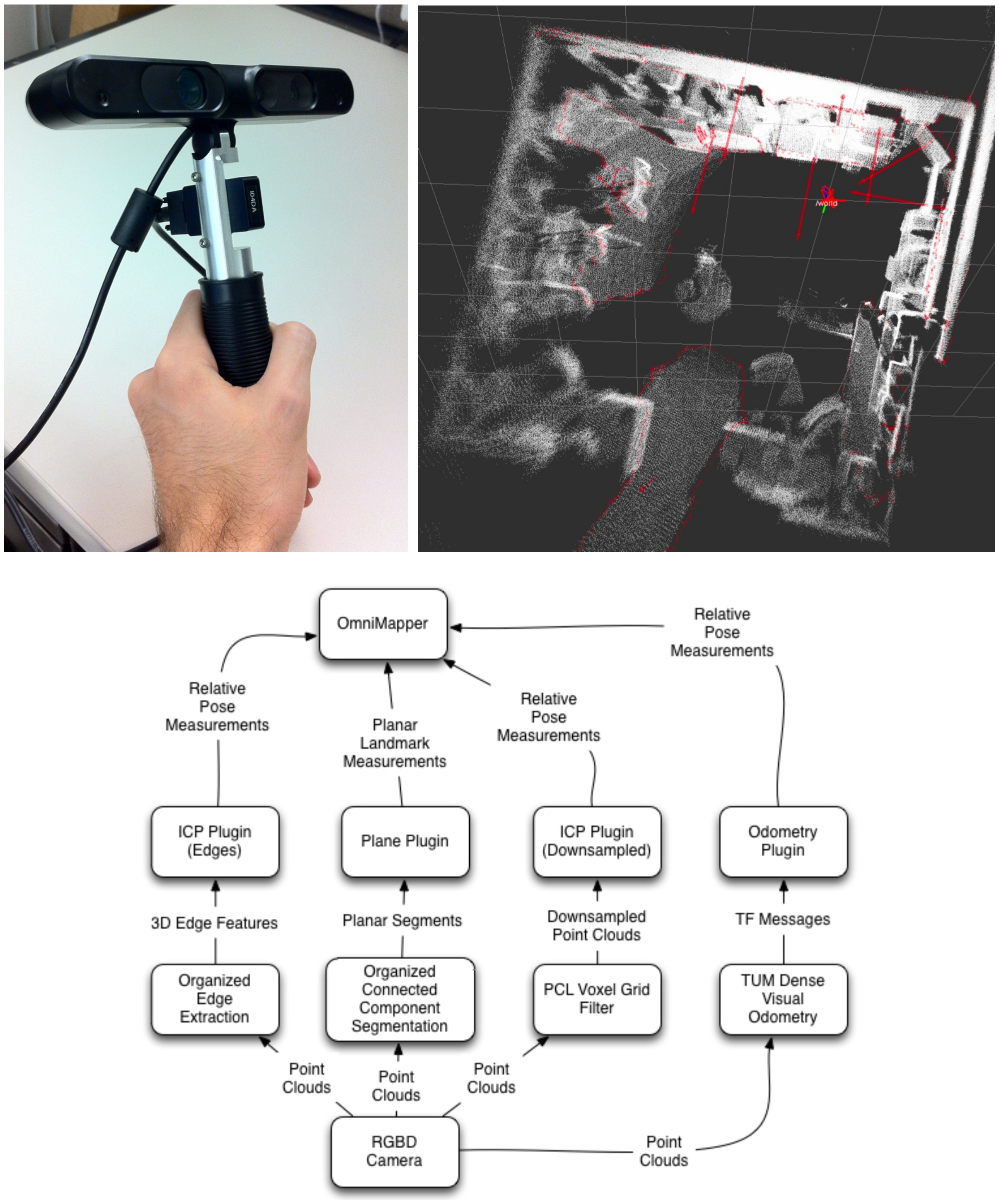


Figure 60: Top Left: A handheld RGB-D camera. The pictured IMU is not used in this work. Top Right: An example map produced by our system with this platform and configuration. Bottom: A system diagram representing the configuration and plugins used.

the gmapping package distributed with the ROS navigation stack, and used on the PR2 robot. While several SLAM approaches can be applied to these types of measurements, such as Extended Kalman Filters and Particle Filters, graphical approaches can also be used.

We have applied the OmniMapper system to a holonomic industrial robot equipped with laser scanners and wheel odometry, shown in Figure 61. The system was used to map and localize the robot for navigational purposes in an industrial environment.

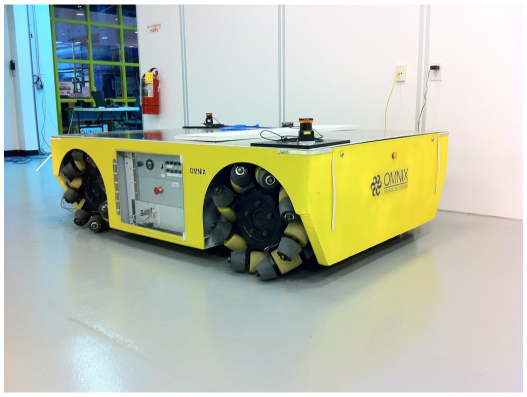
5.5 Mapping Results

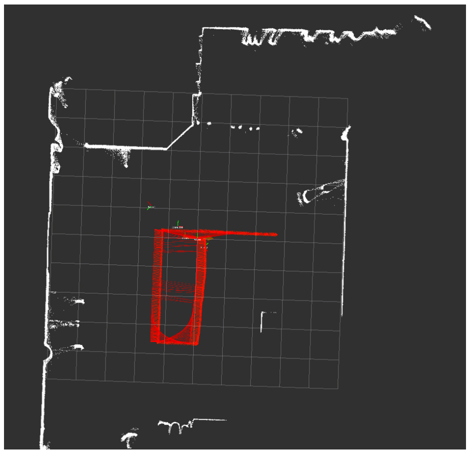
Section 5.4 described several applications of the mapping system to demonstrate its flexibility. Here we describe experiments performed with the mapper. Note that the above chapter describes the mapping system in its most recent form, but not all experiments were performed at the same time, thus some details vary. Relevant changes will be described within.

5.5.1 3D Edge-based SLAM Results

When developing the edge-based features described in Section 4.1.2, experiments were performed to test their utility for SLAM. These results were originally published in [23].

Some of the Freiburg 1 datasets were used for evaluation. For this experiment, we employed only occluding edges. Table 10 presents RMSE and total computation times of both RGB-D SLAM [43] and our edge-based SLAM powered by GTSAM [34]. The results imply that our edge-based pose-graph SLAM is comparable to the state-of-the-art RGB-D SLAM in terms of accuracy. It even reported better results in "FR1 plant" and "FR1 room" sequences. For computational efficiency, the edge-based SLAM is quite efficient. Thanks to the faster pair-wise ICP with the occluding edges,





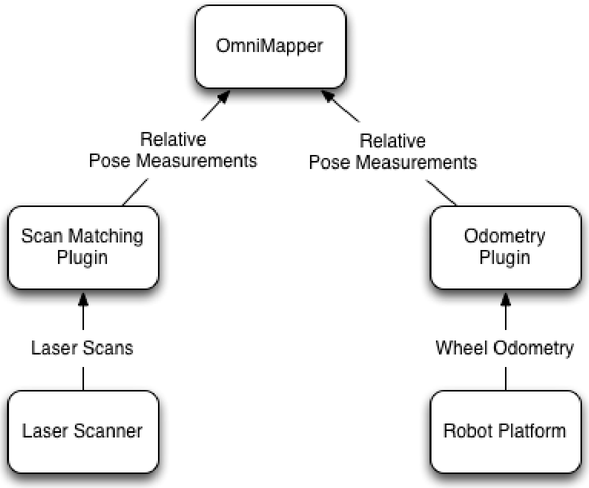


Figure 61: Left: A holonomic mobile robot equipped with laser scanners and wheel odometry. Center: A system diagram representing the configuration and plugins used. Right: An example map produced by our system with this platform and configuration.

the total runtime of our SLAM system is about three times faster than the reported runtime of [43], though we could not directly compare the total runtime because both SLAM systems were run on different computers. Fig. 62 shows two trajectory plots of “FR1 plant” and “FR1 room” sequences where our SLAM reported better results. According to the plots, it is clear that trajectories of the edge-based SLAM are smoother as well as more accurate. These results indicate that the occluding edges are promising yet efficient measurement for SLAM problems.

Table 10: Performance of RGB-D SLAM [43] and our edge-based SLAM over some of Freiburg 1 sequences

Sequence	**SIFT-based RGB-D SLAM [43]**			**Edge-based Pose-graph SLAM**		
	Transl. RMSE	**Rot. RMSE**	**Total Runtime**	**Transl. RMSE**	**Rot. RMSE**	**Total Runtime**
FR1 desk	**0.049** m	**2.43** deg	199 s	0.153 m	7.47 deg	**65** s
FR1 desk2	**0.102** m	**3.81** deg	176 s	0.115 m	5.87 deg	**92** s
FR1 plant	0.142 m	6.34 deg	424 s	**0.078** m	**5.01** deg	**187** s
FR1 room	0.219 m	9.04 deg	423 s	**0.198** m	**6.55** deg	**172** s
FR1 rpy	**0.042** m	**2.50** deg	243 s	0.059 m	8.79 deg	**95** s
FR1 xyz	0.021 m	**0.90** deg	365 s	0.021 m	1.62 deg	**111** s

5.5.2 3D and 2D Measurements of Planar Landmarks Results

The mapper has also been used to evalute the planar mapping approach described in Section 4.2. These results were originally published in [149].

5.5.2.1 Robot Platform

The robot platform used in this evaluation consists of a Segway RMP-200 mobile base, which has been modified to be statically stable. It is equipped with a SICK LMS-291 for obstacle avoidance and navigation, although this is not used for this work. 3D point cloud information is collected with an Asus Xtion Pro depth camera. Laser line information is collected with a Hokuyo UTM-30 laser scanner. Computation is performed on an onboard laptop; however, our experiments are run on desktop machines using log data gathered from the robot. The robot is shown in figure 63.

Although it was not used in this work, the robot is also equipped with a Schunk PG-70 gripper with custom fingers attached to a vertical linear actuator, enabling the robot to grasp objects on tabletop surfaces.

5.5.2.2 Experimental Results

We collected data from office and home environments to test the performance of our SLAM system. The first two test runs were collected in the the Georgia Tech Robotics and Intelligent Machines center, and another test run was collected at a house near the Georgia Tech campus.

In the first experiment, the robot is teleoperated in the robotics student cubicle area in two large overlapping loops. The robot moves through the atrium twice in this data set. The atrium is large enough that the plane extraction filters will not find any planes close enough to the robot to make measurements during that part of the trajectory. The trajectory is relatively free of clutter so the laser line extractor has a

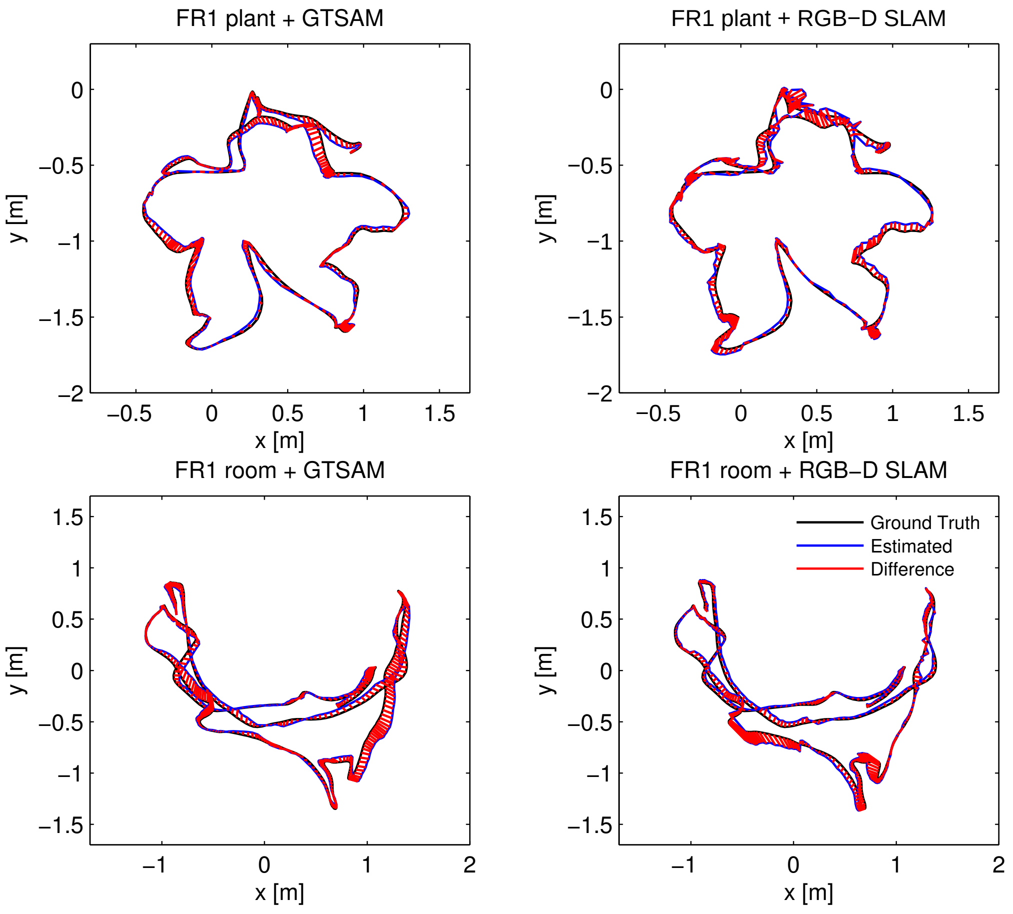


Figure 62: Plots of trajectories from two SLAM approaches.

clear view of large planar structures at all times. A photo of one of the corridors in this area is shown in Figure 66, and a floor plan is shown in Figure 70. A top-down orthographic view of the map and robot trajectory produced by our system on this dataset is shown in Figure 69.

The second experiment is a loop that passes through the hallways and a very cluttered laboratory. The hallways are narrow enough that both plane and laser measurements will be possible; however, the laboratory is both large and cluttered. The size of the laboratory will prevent finding large planar structures within the range of the depth camera, so the robot will not be able to make planar measurements

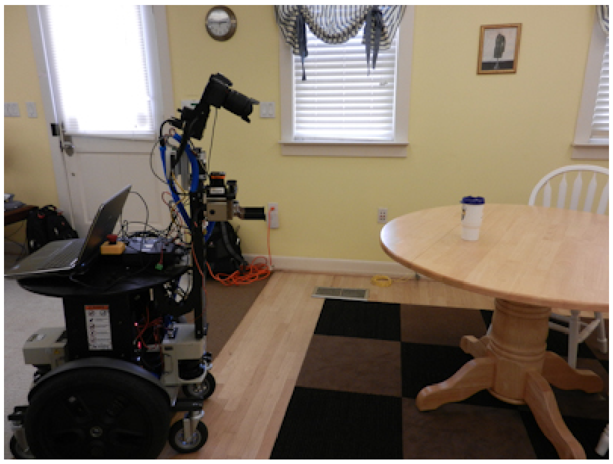


Figure 63: The robot platform used in this work. Point cloud data is collected with the Asus Xtion Pro camera, and laser scan data is collected with the Hokuyo UTM-30. Both of these sensors are placed on the gripper near the middle of the robot. The Asus camera is mounted upside-down on top of the gripper for more rigid and repeatable attachment.

here. The laboratory is also very cluttered, so the laser line extractor will not be able to make many long linear measurements along planar structures. A photo of the cluttered lab area is shown in Figure 67.

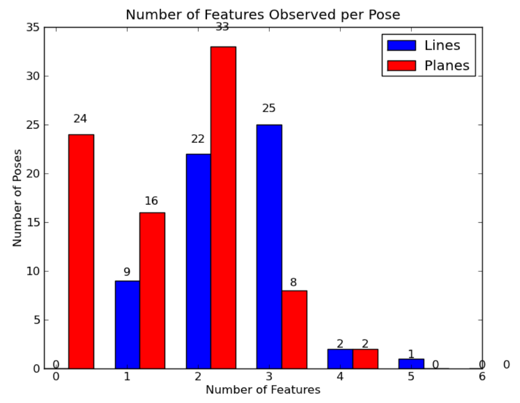
The final experiment is a trajectory in a house where the robot is teleoperated to examine each room. The house includes a living room, kitchen area, bedroom, hallway, and bathroom. The size of rooms and clutter of this test environment falls in between the size and clutter of the two previous test environments. The robot can be seen in this environment in Figure 63.

In order to evaluate the contributions of each sensor type, we performed a series of analyses on each data set. First, we analyzed the number of measurements of each type per pose. For all mapping runs, our robot was configured to add a pose to the trajectory on the next sensor measurement from each sensor type after traveling 0.5 meters or more according to the odometry, even if no features were detected. The results are shown in Figure 64. It can be seen that all of the datasets include several poses that have no plane measurements. This is not surprising, due to the limited field of view and range of the range camera, which was especially problematic in the open areas present in the first and second data sets, but was less of an issue in the smaller spaces of the home environment. We should also note that there were even some poses on the second dataset which also had no detected linear features, despite the wide field of view of the laser scanner. These poses occurred primarily in the lab portion of the trajectory, which includes high clutter occluding the views of the walls.

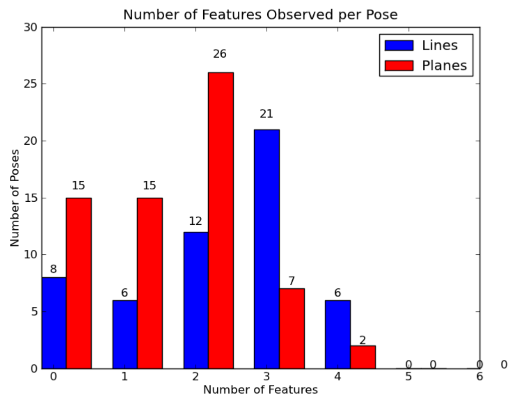
Then, in order to demonstrate that both of the sensors are contributing in different ways, we plotted the number of landmarks measured only by laser measurements, the number of landmarks measured only by 3D planar measurements, and the number of landmarks measured by both. The results are displayed in Figure 65. It is clear that both types of sensors are making contributions to the overall map, as for all datasets, there are many features measured only by the laser scanner, only by the range camera, and many features measured by both. As one would expect, there are more features measured only by the laser scanner in our second data set, which included larger spaces, which posed a challenge for the range camera's limited field of view and range. In general, the landmarks that were observed only by the laser scanner tended to be either out of the field of view of the range camera, or beyond its maximum range. The landmarks viewed only by the range camera often included horizontal surfaces such as tables or shelves, or planar patches that did not intersect the 2D laser scanner's measuring area. Many large planar surfaces such as nearby walls were measured by both.

5.5.3 Object Landmarks: Door Signs Results

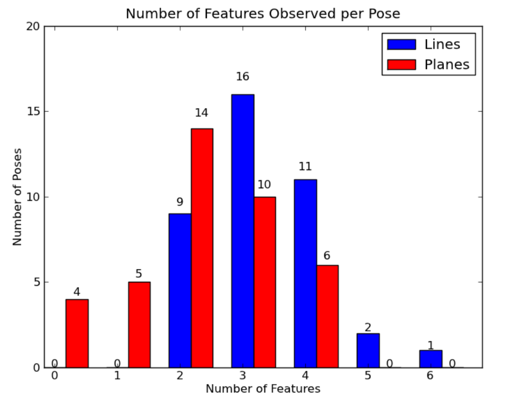
We performed a series of experiments to demonstrate the use of a learned classifier to generate landmark measurements in a SLAM context. We used the door sign



(a)

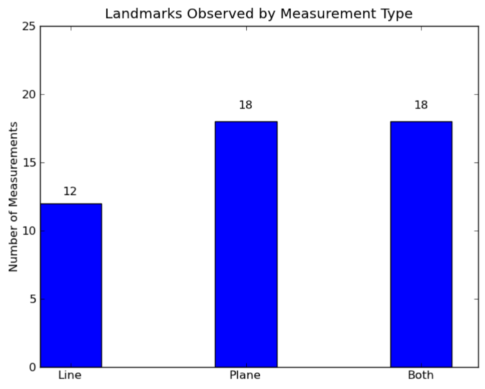


(b)

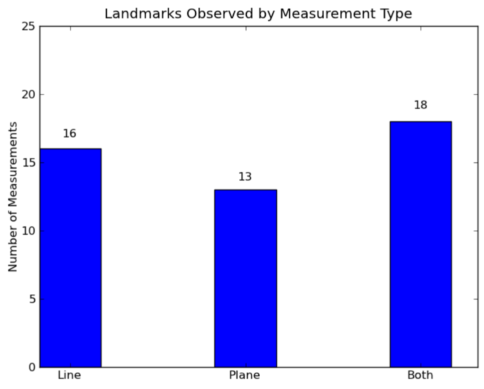


(c)

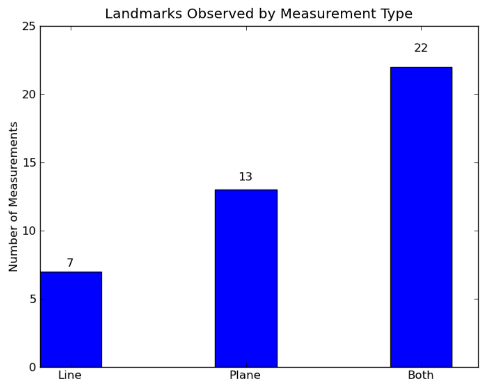
Figure 64: Shown are the number of features observed per pose from each of our three data sets. From left to right, shown are results for our low-clutter office dataset, high clutter dataset, and home environment dataset.



(a)



(b)



(c)

Figure 65: Shown are the number of landmarks observed by line measurements only, plane measurements only, and measured by both. From left to right, shown are results for our low-clutter office dataset, high clutter dataset, and home environment dataset.

Figure 66: Another view of the Robotics & Intelligent Machines lab at the Georgia Institute of Technology, where mapping was performed.

Figure 67: A photo of a relatively high-clutter lab in the Robotics & Intelligent Machines lab at the Georgia Institute of Technology.

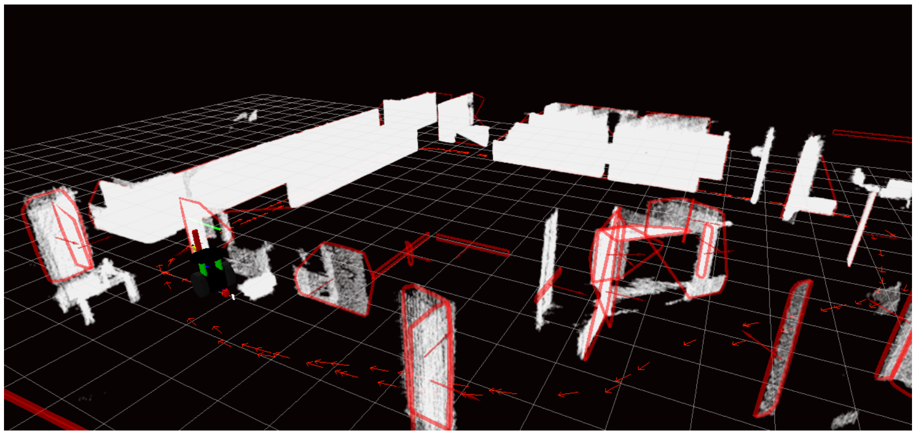


Figure 68: A visualization of a map produced from our low-clutter office environment. Many planar features are visible, including some that have been measured only by the 2D laser scanner. Planar features are visible by the red convex hulls, and red normal vectors. The small red arrows on the ground plane show the robot's trajectory. The point clouds used to extract these measurements are shown in white, and have been rendered in the map coordinate frame by making use of the optimized poses from which they were taken.

classifier described in Section 3.5.3 to generate measurements from door signs in our office, using the mapping approach described in Section 4.3. The classifier used in this experiment was trained on images taken from a hand-held camera from a variety of different door signs on the second floor of our building. Multiple test runs consisting of different size loops were collected. Additionally, the training set was made from hand labeled and selected regions while the test run was made using the automatic saliency analysis and blob extraction and fixed sampling as explained in Section 3.5.3. We also collected wall measurements from the laser scanner and used both feature types to generate maps.

The robot is shown in Figure 72. It is a Segway RMP-200 modified with external caster wheels. This modification allows us to operate without using the balancing mode, which offers additional stability and safety. The robot makes use of a SICK LMS-291 laser scanner to collect measurements of walls and a Prosilica 650c camera

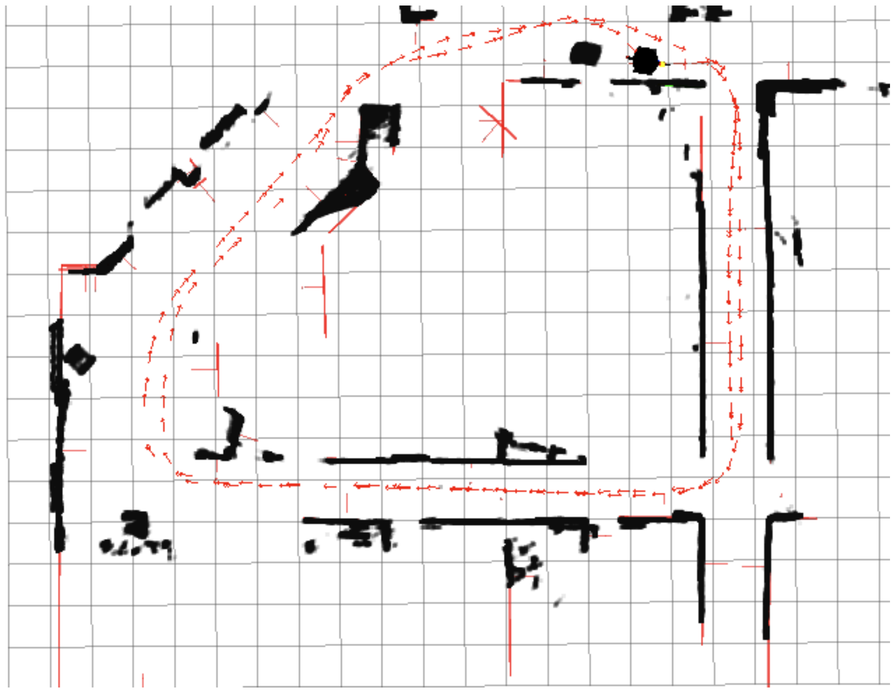


Figure 69: A top-down visualization of a map produced from our low-clutter office environment. Many planar features are visible, including some that have been measured only by the 2D laser scanner. Planar features are visible by the red convex hulls, and red normal vectors. The small red arrows on the ground plane show the robot's trajectory. The point clouds used to extract these measurements are shown in white, and have been rendered in the map coordinate frame by making use of the optimized poses from which they were taken. A 1 meter square grid is displayed for scale.

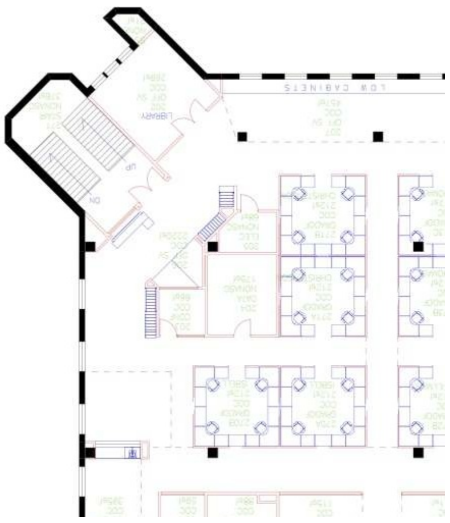


Figure 70: A floorplan of the low-clutter area that was mapped.

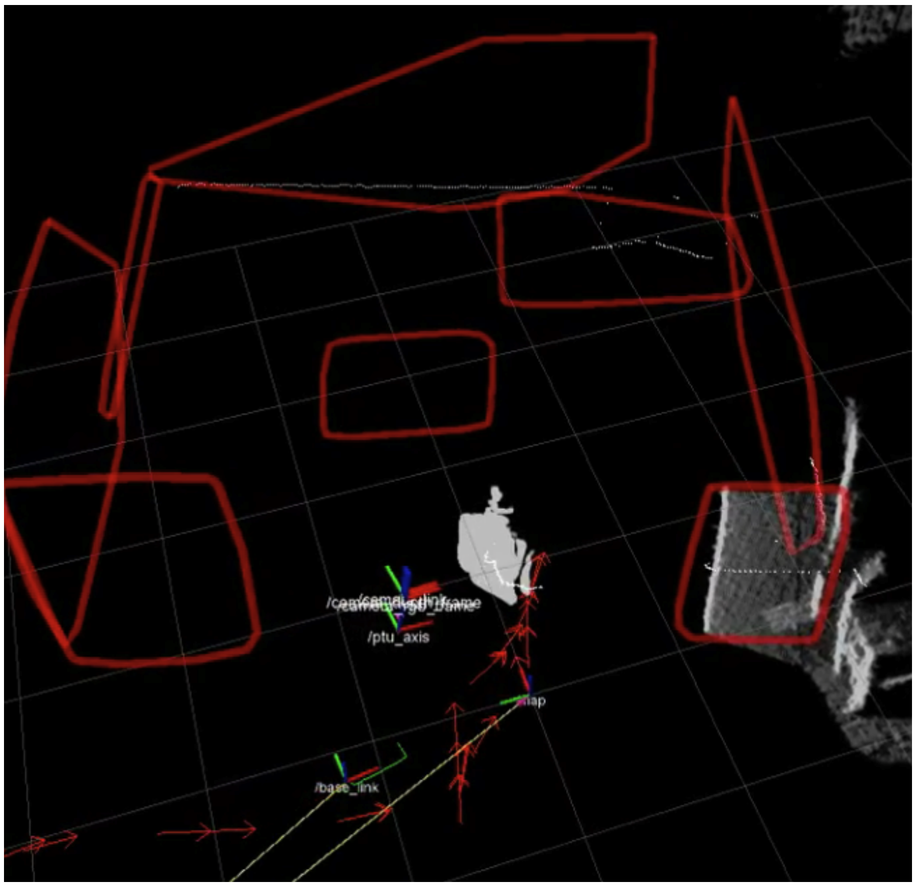


Figure 71: A visualization of a map that includes a low coffee table, a desk, and several walls.

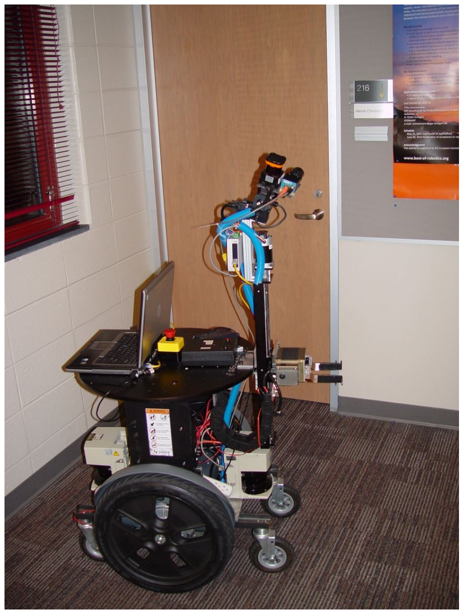


Figure 72: The Segway RMP 200 with LMS 291 laser scanner for wall measurements and a Prosilica 650c camera with a Hokuyo UTM30 laser scanner on a PTU-46-70 pan-tilt unit. Four caster wheels were added for stability. The robot is shown in a position typical of reading a door sign.

with a Hokuyo UTM30 laser scanner mounted on a pan-tilt unit to collect images of door signs. The pan-tilt unit was controlled by the robot operator to point at the door signs during the test data collection. Camera images are collected automatically at a regular interval, not just when the camera is aimed at a door sign.

Currently, the robot is tele-operated in the environment while its sensor and odometry data are logged by ROS. The data file is processed offline by our system to construct the final map. This is done for convenience and repeatability, as well as to support our development. Our algorithms run in better than 2x real time with up to 353 poses and over 800 total measurements in the longest run. The transaction through GoogleGoggles server however does require about 4 seconds per frame, but this is only performed on image regions which are determined to be door signs. The mapper has been designed to operate asynchronously with measurement sources, so it can process messages arriving out of order from the recent past.

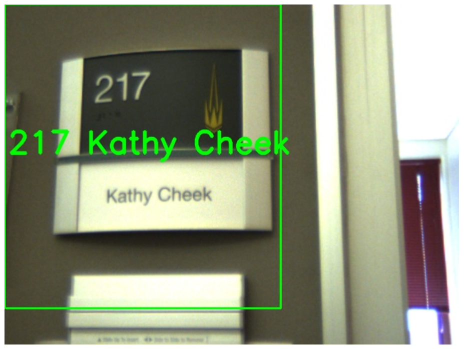


Figure 73: This sign is recognized and a measurement is made in the mapper. Google-Goggles has read both the room number and the text, so this sign can be used for data association.

A total of five test runs were performed, two runs at each of short and middle loop sizes, and one run at the large loop size. An example image is show in Figure 74 which illustrates the types of features which are mapped and how they are displayed in the maps. Some of the larger loops are big enough that it is hard to see the details in these images; please consult the accompanying video for additional details.

The short loop size is about 30 meters. In the runs at this loop size, the robot starts by proceeding down the west hallway and it is carefully driven and the camera is aimed at the door signs. The robot then drives through a cluttered laboratory space where few measurements can be made of the walls, resulting in significant pose error when the robot exits the lab. The robot is then driven back into the west hallway, but it doubles the hallway because of significant pose error, see Figure 75. In each of the test runs, the robot makes three successful sign matches and realigns the map, as shown in Figure 76.

The middle size loop run takes the robot first down the west hallway, and then onward into the half of the floor which is still under construction. Significant portions of this area contain clutter and are difficult to find wall segments large enough for mapping, resulting in some significant pose error in this portion. The robot then proceeds through the back hallways and around the loop for about 200 meters, where it re-enters the west hallway. The robot is now lost because the walls are not successfully matched due to significant pose error, see Figure 77. After door signs are matched, the robot is relocalized and the map is corrected in Figure 78.

The long run starts out the same as the middle length run but instead of proceeding back to the west hallway after exiting the back hallways and the construction area, the robot proceeds around the largest loop possible on our floor by driving down the east hallways and through the kitchen and atrium before returning to the west hallway. As with the other runs, the robot has become lost by the time it re-enters the west hallway and is unable to close the loop using only the wall features, see Figure 79. Once again, door signs are re-observed and the loop is closed, resulting in a useable map and a localized robot, see Figure 80.

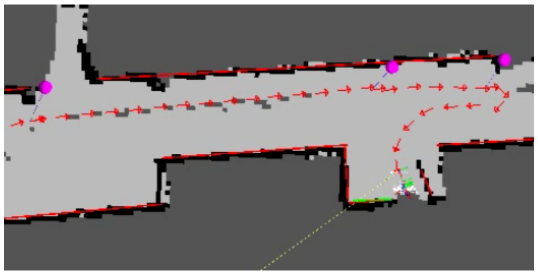


Figure 74: A close-up of the west hallway. Robot poses are shown as red arrows. Wall features are shown as red lines. Door signs are shown as pink spheres. The occupancy grid is displayed only for clarity. This figure is best viewed in color.

5.5.4 Object Landmarks: Online Object Discovery

The object discovery, modeling, and mapping approach described in Section 4.4 has been evaluated using the OmniMapper framework. These results were originally published in [24].

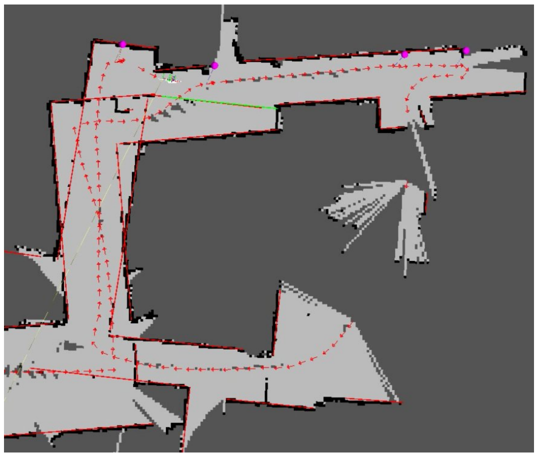


Figure 75: A map through a cluttered lab space which has few line features for measurement, resulting in significant pose error and a failure to close the loop before re-measuring a door sign.

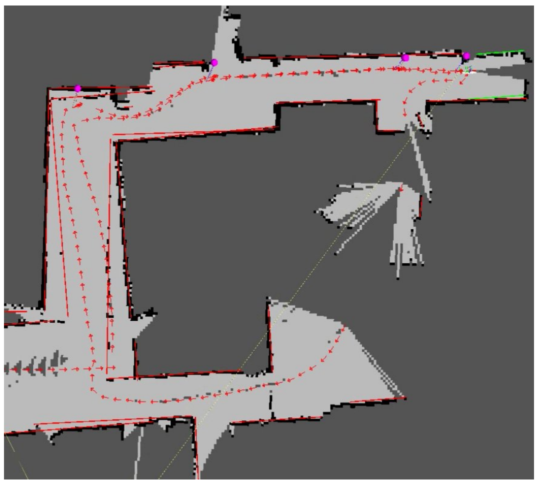


Figure 76: A door sign is re-observed and the text is matched, resulting in a loop closure.

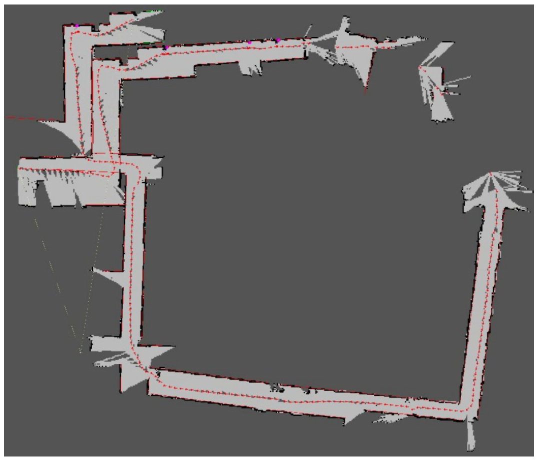


Figure 77: A longer mapping run through the hallways in our building. The mapping run goes through a cluttered area under construction and gets lost, resulting in a several meter trajectory error, before a door sign is re-observed.

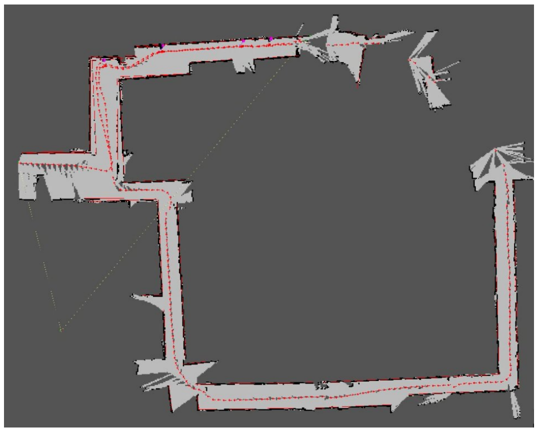


Figure 78: Once a door sign is re-observed, the loop is closed and the map is corrected.

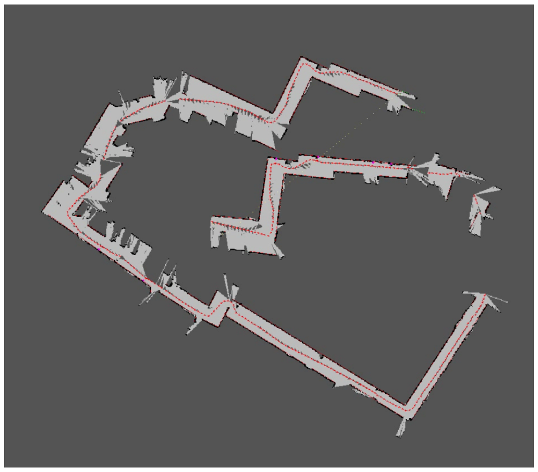


Figure 79: The robot has completed the long loop around the building, but has not yet found a sign match to perform a loop closure. Note that there is significant error in this map when the door signs have not been re-observed.

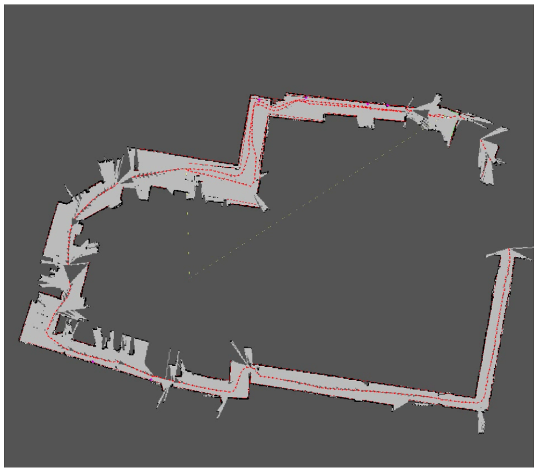


Figure 80: The robot has proceeded further along and it has just re-observed a door sign from the first time around this loop. The map is corrected and the robot now knows where it is.

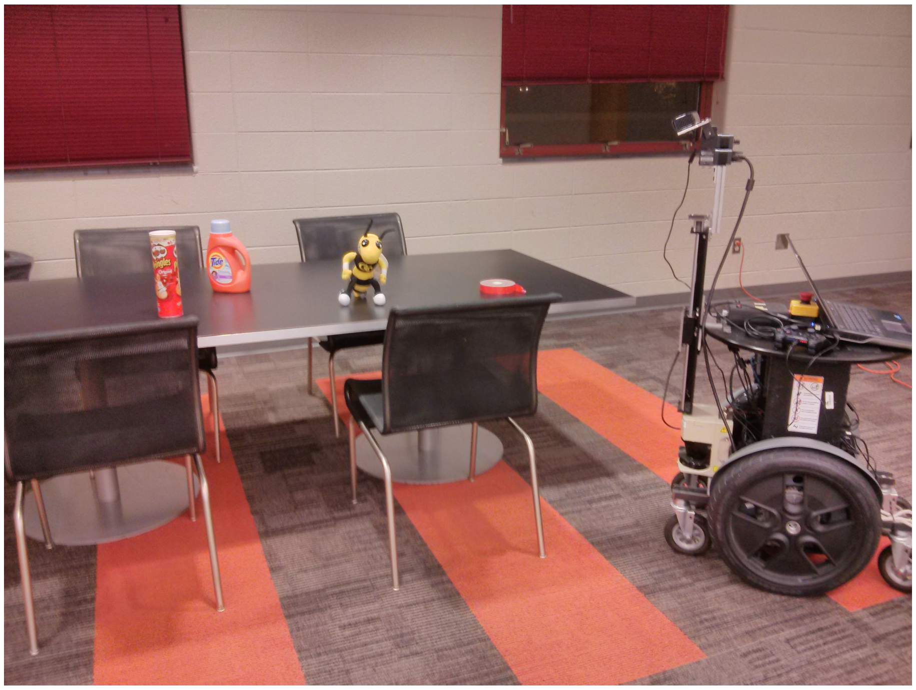


Figure 81: The robot and a snapshot of the scene where the experiment was conducted.

5.5.4.1 Robot Platform

The robot platform used in this work consists of a Segway RMP-200 mobile base, which has been modified to be statically stable. It is equipped with a SICK LMS-291 for obstacle avoidance and navigation, although not used for this work. Kinect depth camera is used to collect point cloud information. Computation is performed on an on board laptop; however, our experiments are run on desktop machines using log data gathered from the robot. To evaluate our system, we collected data from the Georgia Tech Institute for Robotics and Intelligent Machines. The floor plan of the institute is shown in Figure 87(c) and 85(c). The robot and a snapshot of the scene is shown in Figure 81.

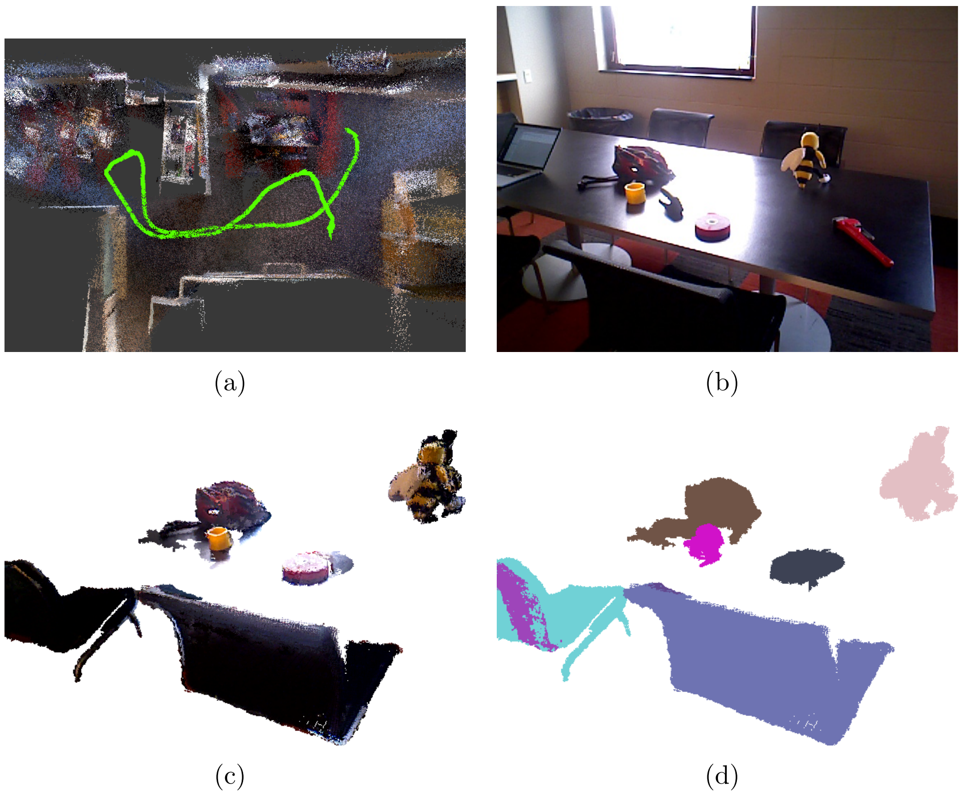
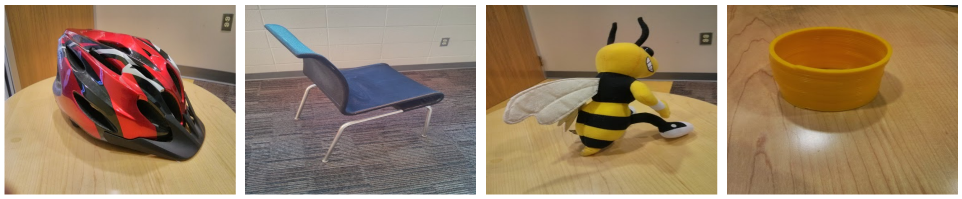


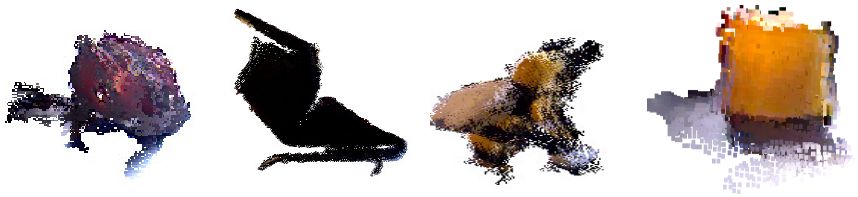
Figure 82: Various stages of the object discovery pipeline. (a) The map of the scene and the trajectory along which the robot is tele-operated. (b) One particular frame from the trajectory showing the objects kept on the table. (c) The corresponding object segments estimated by segmentation propagation (Section 4.4.1.2). (d) The same frame with labels given to discovered objects after object refinement. Each color represents a different object.

Table 11: Number of true positive objects discovered and the effect of object refinement for merging under-segmented objects.

	#Objects	**#True Positive Objects Discovered**	**Percentage Reduction in #Objects Discovered after Object Refinement**
1	12	9	30%
2	5	4	27.28%
3	5	4	24.56%

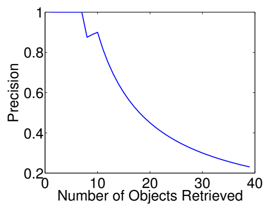


(a)

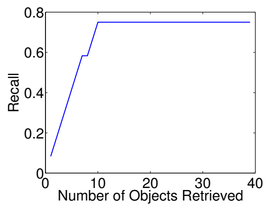


(b)

Figure 83: The top four objects discovered sorted in an descending order of the number of frames it is seen in. (a) Snapshot of the discovered objects. (b) The corresponding reconstructed object models.



(a) Precision



(b) Recall

Figure 84: Precision and Recall curves of Object Discovery

5.5.4.2 Object Discovery

In an experiment, the robot is tele-operated along a trajectory shown in Figure 82(a) to test the object modeling capabilities. Twelve objects were used in this experiment. Figure 82(b) shows one particular frame from the trajectory which shows the objects kept on the table. Figure 82(c) shows the corresponding object segments estimated by segmentation propagation (Section 4.4.1.2). Figure 82(d) shows the same frame with object labels given to discovered objects after object refinement. Each color represents a different object. The objects labels are well defined with each label representing a different object. Labeling error in the back of the left chair is due to per-frame segmentation error which gives it a new label. However the segmentation errors are not propagated across many frames.

Figure 83 shows the top four objects discovered sorted by number of frames it is seen in. Figure 83(a) shows the image of the discovered objects and figure 83(b) shows the corresponding reconstructed model. We found that the noisy objects detected due to segmentation failures are not propagated across many frame and the quality of the object discovered is directly proportional to the number of frames it is seen in. This is analogous to the 3D point based SLAM problem where a 3D point visible from many images has a well defined location and that point is more likely to be seen in a new image. Similarly an object seen across many frames is better modeled and it is more likely to recognize this object in a new frame. Assuming 12 objects, Figure 84 shows the precision and recall curves representing the quality of the retrieved objects when retrieved according to the number of frames they are seen in. As we can see most of the true positive objects are seen across many frames. In general, storing top 15 objects includes all the true positive discovered objects.

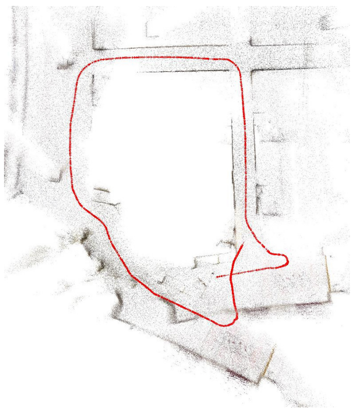
Table 11 shows the results of object discovery and the effect of object refinement for merging under-segmented objects. Row one represents the object discovery experiment where the robot is tele-operated along a trajectory shown in Figure 82(a). The remaining rows correspond to the loop closure experiments described in Section 5.5.4.4. As we can see the from the results, the object discovery pipeline is able to discover most of the objects used in the experiments. The number of discovered objects on an average is reduced by 25% on using object refinement which merges the under-segmented object parts.

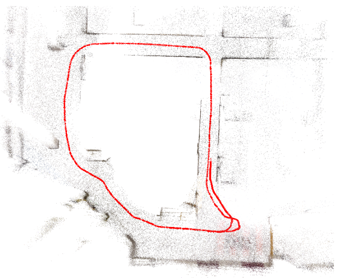
5.5.4.3 Object Recognition

Luis A. Alexandre evaluated different 3D descriptors for object and category recognition and showed that CSHOT (or SHOTCOLOR) worked the best considering its recognition accuracy and time complexity [3, 141]. CSHOT has an object recognition accuracy of 75.53% [3]. While doing object recognition we only consider the objects which lie within twice the uncertainty radius of an object (Equation 6). It helps in reducing the false positive matches. Other than this we assume that the objects do not move during the experiment and are non-repetitive (except chairs). All these increase the recognition performance. We do not explicitly remove the false positive data association and assume it to be unlikely considering the uncertainty constraint and static, non-repetitive assumption.

5.5.4.4 Loop closure detection

To test the loop closing capabilities, the robot is tele-operated in the cubicle area forming a loop as shown in Figure 85(c) and 87(c). In the first experiment, the robot moves through the atrium twice with objects kept on the table. Objects kept on the table are used as landmarks when closing the loop. Figure 85(c) shows the approximate trajectory on which the robot is run.


(a)


(b)

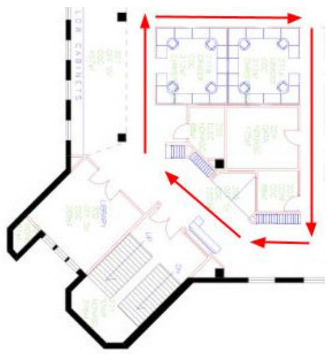

(c)

Figure 85: Trajectories estimated with and without using objects as landmarks during loop closure. (a) Robot trajectory estimated when using no object landmarks. There is an error in the bottom right section when the robot returns to the same location. (b) Robot trajectory estimated when using object landmarks for loop closure. The object-object constraints joins the bottom right location when loop closure is detected. (c) Shows the approximate ground truth trajectory w.r.t the floor plan.

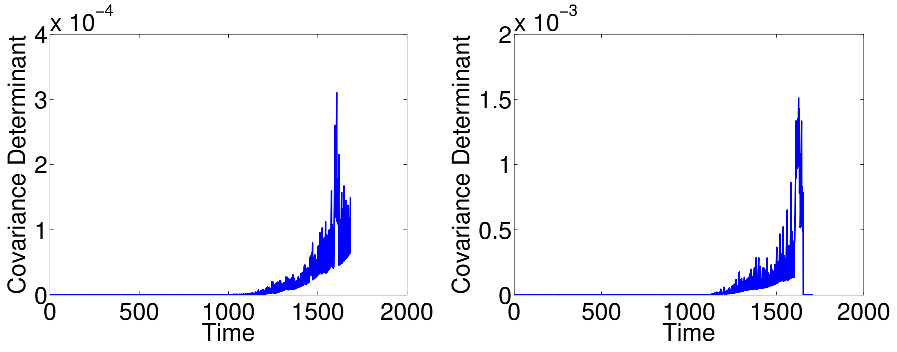


Figure 86: Covariance determinant of the latest pose w.r.t time when not using objects (left, corresponds to Figure 85(a)) as compared to when using objects for loop closure (right, corresponds to Figure 85(b)). There is a sudden fall in the value of covariance determinant due to a loop closure event.

In Figure 85(a) we see the robot trajectory estimated when using no object landmarks. There is an error in the bottom right section when the robot returns to the same location. Figure 85(b) shows the robot trajectory estimated when using object

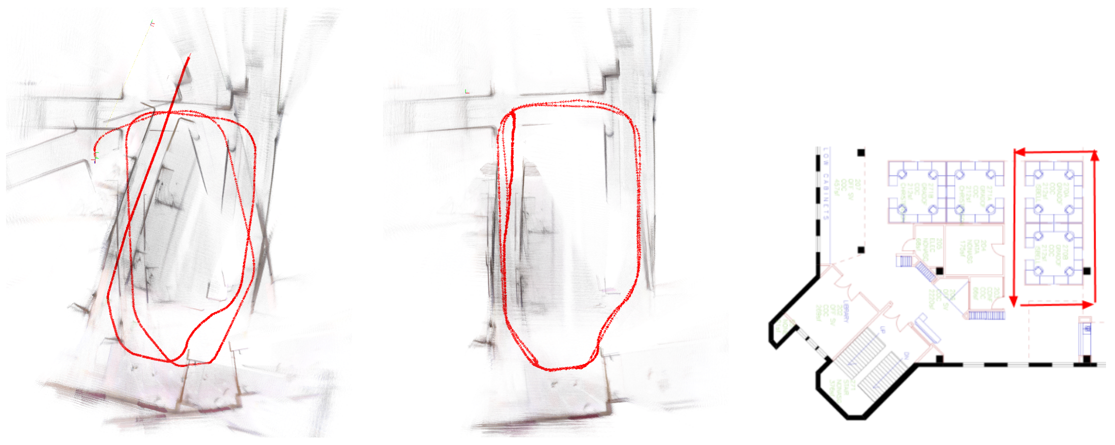

(a) (b) (c)

Figure 87: Trajectories estimated with and without using objects as landmarks during loop closure. (a) Robot trajectory estimated when using no object landmarks. (b) Robot trajectory estimated when using object landmarks for loop closure. (c) Shows the approximate ground truth trajectory w.r.t the floor plan.

landmarks for loop closure. The object-object constraints joins the bottom right location when loop closure is detected. Figure 86 shows the covariance determinant of the latest pose when using objects for loop closure as compared to not using objects. In the left plot, we see that the covariance determinant keeps on rising where as the in the right plot, we see a sudden fall in the value of covariance determinant representing a loop closure. The actual values of the determinant depends on the noise model used.

In the second experiment, the robot is tele-operated for a longer time period to form longer length trajectory. Figure 87(c) shows the approximate trajectory of this run.

In Figure 87(a) we see the robot trajectory estimated when using no object landmarks. There is an error in the estimated trajectory due to the error in odometry and ICP estimates propagated across the full trajectory. Figure 87(b) shows the robot trajectory estimated when using object landmarks for loop closure. Object-object loop closure constraint is able to reduce the error in the trajectory estimate.

5.6 Discussion

Chapter 4 focused on our contributions to SLAM in new feature types, while Chapter 5 discussed our mapping framework. The work on the OmniMapper framework was developed in parallel with the complex features we developed (planes and objects), because while informative, these features are not always suitable for use as sole measurement sources in SLAM, as sufficient measurements may not be present from all locations. To speed up our development cycles and ease software re-use, we developed the OmniMapper framework, aiming to make it easy to support any types of features described in Chapter 4.

We believe the framework is suitable for this purpose new plugins and feature types can now be developed with no modifications to the core mapper, and combined with existing plugins. Applying the system to new applications is also much easier we were able to adapt our handheld RGB-D mapping system for an augmented reality system in just a few days, because only the visualizer needed to be modified. By releasing the framework as open source, we hope that others will be able to benefit from it as well.

CHAPTER VI

HUMAN-ROBOT INTERACTION FOR MAP ANNOTATION

A service robot system is not complete without an interface – some means for a user to interact with and communicate with the robot. This chapter will describe our contributions in this area, focusing on interfaces to enable map annotation and basic service robotic tasks. In particular, we developed an interface that combines deictic gesture recognition and a tablet or smartphone UI for interactively annotating structures and objects in maps, such as the maps described in Chapter 5. Section 6.1 describes our approach to detecting and modeling uncertainty for deictic gestures. We then show how such gestures, combined with a tablet / smartphone UI can be used to interactively annotate maps in Section 6.2. In Section 6.3, we then show how a robot can store observations of humans in a map, and leverage that information to stay out of their way during a cooperative task.

6.1 Pointing Gestures for Human-Robot Interaction

A natural way to reference structures and objects when interacting with a robot is through deictic gestures. RGB-D sensors and skeleton tracking such as [130] have enabled real-time recognition of human poses relative to an RGB-D sensor. In this chapter, we describe how this can be used to recognize such deictic gestures and the structures and objects they refer to, including uncertainty. This was joint work with Akansel Cosgun and Henrik Christensen, and was originally published in [28].

People often use deictic gestures in everyday activities when they are referring to an object, human or a space in their visual field. For example, when humans give directions, they point in the direction of the target they are discussing. If discussing a set of nearby objects, it is natural to refer to an object by pointing to disambiguate it from others. Robot perception systems that are able to correctly interpret pointing gestures can benefit human-robot interaction. When there are several potential pointing targets in the scene, such as multiple objects, the intended target may be ambiguous. Robots that are able to detect such ambiguity can act more intelligently by requesting clarification from a user.

Many approaches to robot learning involve humans teaching robots to perform tasks [139]. Pointing gestures can be helpful to teach new objects, places and people to service robots. We are particularly interested in a home tour scenario [147], in which a user guides a service robot through a space, and annotates rooms, structures and objects with labels. This process establishes common ground [16] between the human and the robot, enabling referencing of labeled locations and objects in future interactions. In our previous work, we have employed pointing gestures in service robot interactions for annotating planar structures in semantic maps [152], and object models [153]. Labels are input by the user to a tablet of smartphone interface, and the target of the pointing is then annotated with this label. While the detection and usage of pointing gestures has been addressed previously, these efforts do not detect or reason about ambiguous references.

In this work, we model the uncertainty of pointing gestures in a spherical coordinate system, use this model to determine the correct pointing target, and detect when there is ambiguity. Two pointing methods are evaluated using two skeleton tracking algorithms: elbow-hand and head-hand rays, using both OpenNI NITE and Microsoft Kinect SDK [129]. A data collection with 6 users and 7 pointing targets was

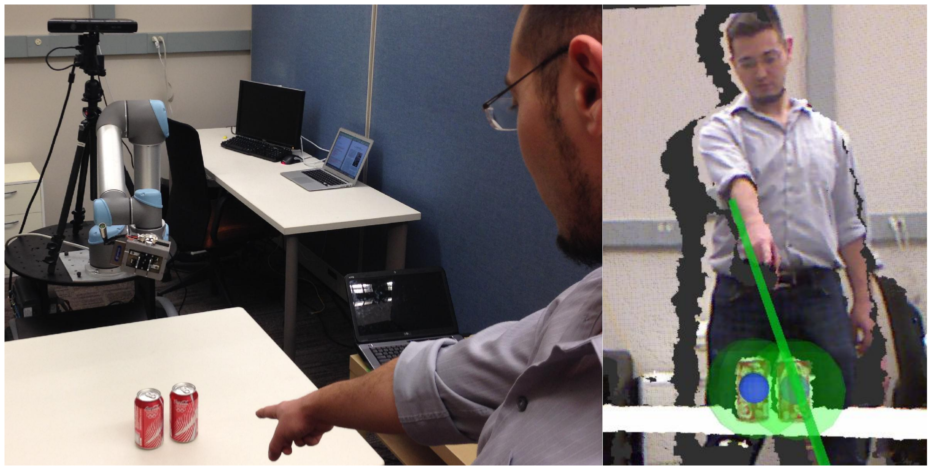


Figure 88: (Left) Our approach allows a robot to detect when there is ambiguity on the pointing gesture targets. (Right) The point cloud view from robot's perspective is shown. Both objects are identified as potential intended targets, therefore the robot decides that there is ambiguity.

performed, and the data was used to model users pointing behavior. The resulting model was evaluated for its ability to distinguish between potential pointing targets, and to detect when the target is ambiguous. An example scenario is shown in Figure 88.

6.1.1 Related Works

Pointing gestures are widely used in Human-Robot Interaction applications. Examples include interpretation of spoken requests [167], pick and place tasks [14], joint attention [39], referring to places [56] and objects [126], instructing [94] and providing navigation goals [117] to a robot.

Early works on recognizing pointing gestures in stereo vision utilized background subtraction [25, 69, 73]. Other popular methods include body silhouettes [75], hand poses [66], motion analysis [95] and Hidden Markov Models [161, 10, 90, 103, 40, 4].

After deciding if a pointing gesture occurred or not, an algorithm must estimate the direction of pointing. This is typically done by extending a ray from one body part to another. Several body part pairs are used in the literature, such as eye-fingertip [75] and shoulder-hand [64]; with the two of most commonly used methods being elbow-hand [117, 16, 14] and head-hand [10, 126] rays. Some studies found that head-hand approach is a more accurate way for estimating pointing direction than elbow-hand [116, 39]. Recent works made use of skeleton tracking data from in depth images [116, 14, 117]. Other approaches, such as measuring head orientation with a magnetic sensor [103] and pointing with a laser pointer [22, 76] is reported to have a better estimation accuracy than the body parts method, but require additional hardware. We prefer not to use additional devices in order to the interaction as natural as possible.

Given a pointing direction, several methods have been proposed to determine which target or object is referred by the gesture, including euclidean distance on a planar surface [22], ray-to-cluster intersection in point clouds [14, 116] and searching a region in interest around the intersection point [126]. Droeschel [40] trains a function using head-hand, elbow-hand and shoulder-hand features with Gaussian Process Regression and reports a significant improvement on pointing accuracy. Some efforts fuse speech with pointing gestures for multi-modal Human-Robot Interaction [4, 83]. Aly [4] focuses on relation between non-verbal arm gestures and para-verbal communication based on a HMM approach. Matuszek *et. al* presented a method for detecting deictic gestures given a set of detected tabletop objects, by first segmenting the users hands and computing the most distal point form the user, then applying a Hierarchical Matching Pursuit (HMP) approach on these points over time [96].

To our knowledge, only Zukerman [168] and Kowadlo [83] considered a probabilistic model for determining the referred object for a pointing gesture. The probability that the user intended an object is calculated using the 2D distance to the object and the occlusion factor. Objects that reside in a Gaussian cone emanating from the user's hand are considered as candidates in the model. The approach is implemented in [168], where it is reported that due to high variance of the gesture recognition system, the Gaussian cone typically encompassed about five objects in cluttered settings. Our work addresses the confusion in such settings. In contrast to their work, we use a 3D elliptical Gaussian cone where its shape is extracted using a prior error analysis, and use point cloud data to account for the size of the objects.

6.1.2 Pointing Gesture Recognition

Our approach to pointing gesture recognition is to use a skeleton tracking algorithm as implemented by OpenNI NITE 1.5 (OpenNI) or Microsoft Kinect SDK 1.5 (MS-SDK). Skeleton tracking software produces 3D positions for several important points on the user's body, including hands, elbows, shoulders and head. Our points of interests are user's hands, elbows, and head for the recognition of pointing gestures.

6.1.2.1 Types of Pointing Gestures

We are primarily interested in deictic gestures generated by pointing with one's arm. We consider two rays for determining the pointing direction: elbow-hand and head-hand. Both of these methods were evaluated with the two skeleton tracking implementations. For each depth frame, this yields two rays for each of the OpenNI and MS-SDK trackers:

- $\overrightarrow{v_{eh}} := p_{elbow} p_{hand}$
- $\overrightarrow{v_{hh}} := p_{head} p_{hand}$

6.1.2.2 Pointing Gesture Recognition

When a pointing gesture recognition request is received from a higher level process, the gesture is searched in a time window of T seconds. Two conditions must be met to trigger a pointing gesture:

- Forearm makes an angle more than ϕ_g with the vertical axis
- Elbow and hand points stays near-constant for duration Δt_g

The first condition requires the arm of the person to be extended, while the second ensures that the gesture is consistent for some time period. The parameters are empirically determined as: $T = 30s$, $\phi_g = 45°$ and $\Delta t_g = 0.5s$.

6.1.3 Representing Pointing Directions

We represent a pointing ray in two angles: a “horizontal” / “azimuth” sense we denote as θ and a “vertical” / “altitude” sense we denote as ψ. We first attach a coordinate frame to the hand point, with its z-axis oriented in either Elbow-hand $\vec{v}_{eh}$ or Head-Hand $\vec{v}_{hh}$ directions. The hand was chosen as the origin for this coordinate system because both of head-hand and elbow-hand pointing methods include the user’s hand. The transformation between the sensor frame and the hand frame ${}^{sensor}T_{hand}$ is calculated by using an angle-axis rotation. An illustration of the hand coordinate frame for Elbow-Hand method and corresponding angles are shown graphically in Figure 89.

Given this coordinate frame and a potential target point P, we first transform it to the hand frame by:

$$^{hand}p_{target} = T_{hand} * p_{target}$$

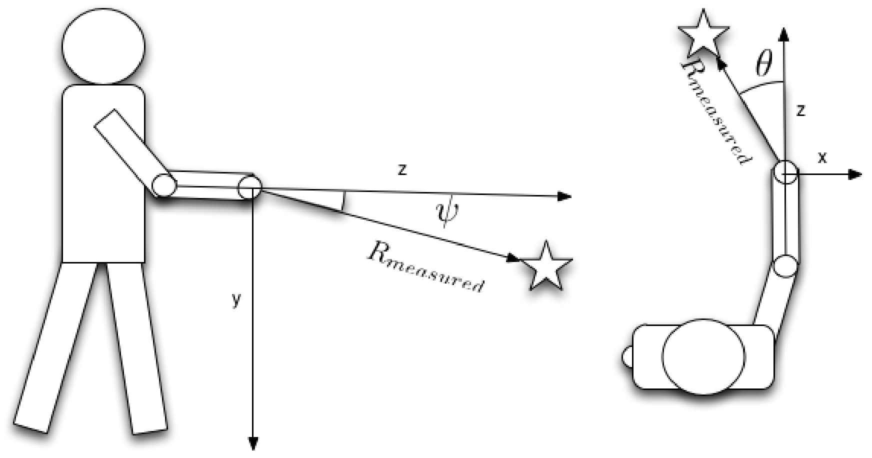


Figure 89: Vertical (ψ) and horizontal (θ) angles in spherical coordinates are illustrated. A potential intended target is shown as a star. The z-axis of the hand coordinate frame is defined by either the Elbow-Hand (this example) or Head-Hand ray.

We calculate the horizontal and vertical angles for a target point as ${}^{hand}p_{target} = (x_{targ}, y_{targ}, z_{targ})$ follows:

$$[\theta_{target}\ \psi_{target}] = [atan2(x_{targ}, z_{targ})\ atan2(y_{targ}, z_{targ})]$$

Where $atan2(y, x)$ is a function returns the value of the angle $\arctan(\frac{y}{x})$ with the correct sign. This representation allows us to calculate the angular errors in our error analysis experiments in Section 6.1.6. The angles for each object is then used to find the intended target, as explained in the following section.

6.1.4 Determining Intended Target

We propose a probabilistic approach to determine the referred target by using statistical data from previous pointing gesture observations. We observed that head-hand and elbow-hand methods, implemented using two skeleton trackers, returned different angle errors in spherical coordinates. Our approach relies on learning statistics of each of these approaches, and compensating the error when the target object is searched for. First, given a set of prior pointing observations, we calculate the mean and variance of the vertical and horizontal angle errors for each pointing method. This analysis will be presented in Section 6.1.6. Given an input gesture, we apply correction to the pointing direction and find the Mahalanobis distance to each object in the scene.

When a pointing gesture is recognized, and the angle pair $[\theta_{target}\ \psi_{target}]$ is found as described in the previous section, we first apply a correction by subtracting the mean terms from measured angles:

$$[\theta_{cor}\ \ \psi_{cor}] = [\theta_{target} - \mu_\theta \quad \psi_{target} - \mu_\psi]$$

We also compute a covariance matrix for angle errors in this spherical coordinate system:

$$S_{type} = \begin{bmatrix} \sigma_\theta & 0 \\ 0 & \sigma_\psi \end{bmatrix}$$

We get the values for $\mu_\theta, \mu_\psi, \sigma_\theta, \sigma_\psi$ from Tables 12 and 13 for the corresponding gesture type and target. We then compute the mahalanobis distance to the target by:

$$D_{mah}(target, method) = \sqrt{[\theta_{cor}\ \psi_{cor}]^T S_{method}^{-1} [\theta_{cor}\ \psi_{cor}]}$$

We use D_{mah} to estimate which target or object is intended. We consider two use cases: the objects are represented as a 3D point or a point cloud. For point targets, we first filter out targets that have a Mahalanobis distance larger than a threshold $D_{mah} > D_{thresh}$. If none of the targets has a D_{mah} lower than the threshold, then we consider that the user did not point to any targets. If there are multiple targets that has $D_{mah} <= D_{thresh}$, then we determine ambiguity by employing a ratio test. The ratio of the least D_{mah} and the second-least D_{mah} among all targets is compared with a threshold to determine if there is ambiguity. If the ratio is higher than a threshold, then the robot can ask the person to solve the ambiguity.

If the target objects are represented as point clouds, we then compute the horizontal and vertical angles for every point in the point cloud and find the minimum mahalanobis distance among all. The distance to an object is then represented by this minimum value. Usage of the point cloud instead of the centroid for determining the intended object has several advantages. First, it yields better estimations due to the coverage of the full point cloud. Second, it takes into account the size of the object. For example, if a person is pointing to a chair or door, it is very unlikely that he/she will target the centroid of it. If the point cloud is used, then we can easily tell that the object is targeted.

6.1.5 Data Collection

To evaluate the accuracy of pointing gestures, we created a test environment with 7 targets placed on planar surfaces in view of a Kinect sensor (Figure 90). Depth data was collected from six users, who pointed at each of the seven targets with their right arm while standing at 2 meters away from the sensor. Targets 1 through 4 were on tables positioned around the user, while targets 5 through 7 were located on a wall to the user's right. Our use case is on a mobile robot platform capable of positioning itself relative to the user. For this reason, we can assume that the user is always centered in the image, as the robot can easily rotate to face the user and can position itself at a desired distance from the user.

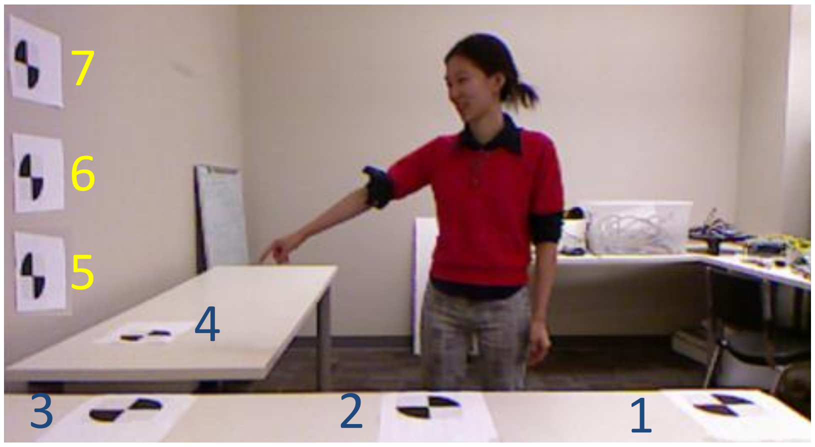


Figure 90: Our study involved 6 users that pointed to 7 targets while being recorded using 30 frames per target.

6.1.5.1 Ground Truth Target Positions

We make use of plane extraction technique in a point cloud to have an accurate ground truth measurement. First, the two table and wall planes are extracted from the point cloud data using the planar segmentation technique described in Section 3.4. We then find the pixel coordinates of corners on targets in RGB images, using Harris corner detection [55], which produces calculated corners in image coordinates with sub-pixel accuracy. The pixel coordinate corresponding to each target defines a ray in 3D space relative to the Kinect's RGB camera. These rays are then intersected with the planes detected from the depth data, yielding the 3D coordinates of the targets.

6.1.5.2 Skeleton Data Capture

In order to be able to do a fair comparison between MS-SDK and OpenNI skeleton trackers, we used the same dataset for both. MS-SDK and OpenNI use different device drivers, therefore can not be directly used on the live depth stream at the same time. Because of this, we extract the skeleton data offline in multiple stages. The pipeline for the data capture procedure is illustrated in Figure 91. We first save the depth streams as .xed files using Microsoft Kinect Studio program. The acquired .xed file is converted to .oni in a OpenNI recorder application by streaming the depth stream to through Kinect Studio. The .oni is then played back in skeleton tracking application in OpenNI, which writes the OpenNI skeleton data to a .txt file. MS-SDK skeleton data is obtained by playing back the original .xed in the skeleton tracking application.

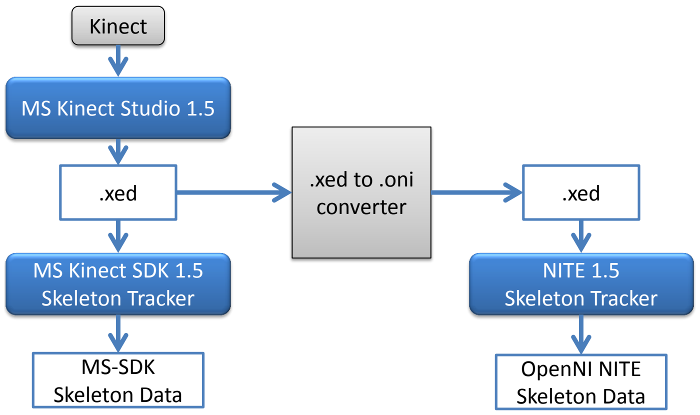

Kinect
MS Kinect Studio 1.5
.xed
.xed to .oni converter
.xed
MS Kinect SDK 1.5 Skeleton Tracker
NITE 1.5 Skeleton Tracker
MS-SDK Skeleton Data
OpenNI NITE Skeleton Data

Figure 91: Data capturing pipeline for error analysis.

6.1.5.3 Pointing Gesture Annotation

To factor out the effectiveness of our pointing gesture recognition method described in Section 6.1.2.2, we manually labeled when each pointing gesture began for data collection. Starting from the onset of the pointing gesture as annotated by the experimenter, the following 30 sensor frames were used as the pointing gesture. This corresponds to a recording of 1 second in the Kinect sensor stream. For each frame, we extracted skeleton data using both the MS-SDK and the OpenNI.

6.1.6 Error Analysis

The four rays corresponding to the four different pointing approaches described in Section 6.1.2.1 were used for our error analysis. As described in Section 6.1.5.1, the ground truth target positions are available. We computed two types of errors for each gesture and target:

1. Euclidean error of ray intersections with target planes (Figure 92)

2. Angular error in spherical coordinates (Tables 12,13 and Figure 93)

We elaborate our error analysis subsequent sections:

6.1.6.1 Euclidean error

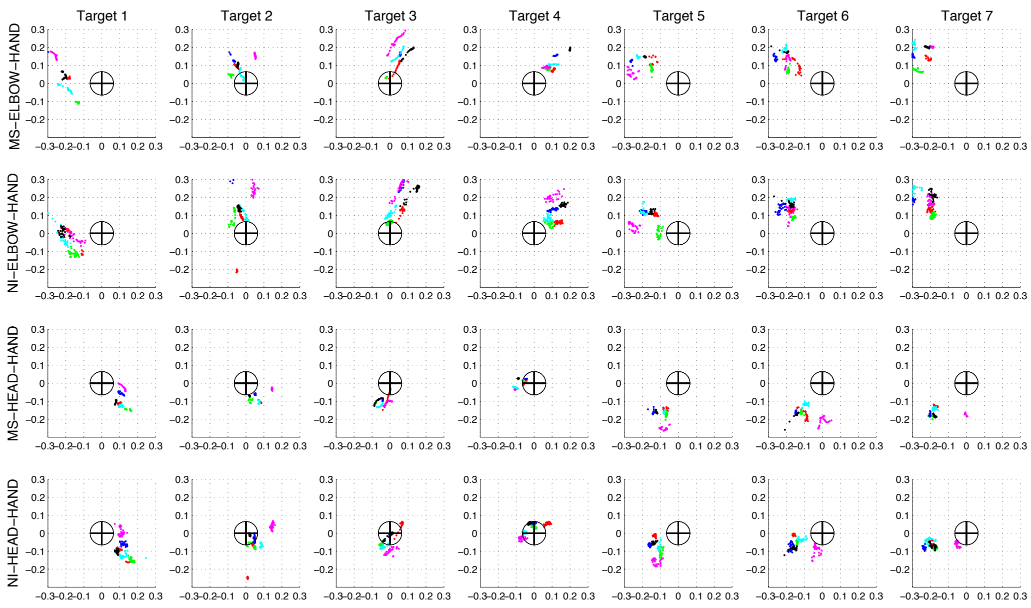


Figure 92: Euclidean distance error in cartesian coordinates for each method and target. The gesture ray intersection points in centimeters with the target plane, are shown here for each target (T1-T7) as the columns. Each subject's points are shown in separate colors. There are 30 points from each subject, corresponding to the 30 frames recorded for the pointing gesture at each target. Axes are shown in centimeters. The circle drawn in the center of each plot has the same diameter (13 cm) as the physical target objects used.

Table 12: μ and σ angular errors in degrees for each of Targets 1-4 (Figure 90), for each pointing method.

	Target 1				Target 2				Target 3				Target 4			
	θ		ψ		θ		ψ		θ		ψ		θ		ψ	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
MS-Elbow-Hand	-15.7	2.9	5.5	6.1	-4.3	6.6	7.8	3.1	-3.7	2.4	7.4	3.1	-3.6	1.9	11.5	2.9
NI-Elbow-Hand	-16.4	2.9	4.3	7.0	-3.8	6.6	11.3	10.9	-4.8	2.6	9.7	3.4	-4.0	4.7	12.4	2.5
MS-Head-Hand	7.7	2.6	-12.0	5.3	10.8	6.4	-9.1	3.2	2.2	2.0	-8.3	3.2	-4.0	1.7	-4.3	3.2
NI-Head-Hand	8.5	2.6	-11.7	6.1	10.2	6.7	-5.7	8.0	2.0	2.3	-2.9	4.7	-3.2	2.5	1.45	4.8

Table 13: μ and σ of angular error in degrees for Targets 5-7 (Figure 90), for each pointing method. The aggregate μ and σ is also shown.

	Target 5				Target 6				Target 7				ALL TARGETS			
	θ		ψ		θ		ψ		θ		ψ		θ		ψ	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
MS-Elbow-Hand	-14.5	4.1	11.6	3.7	-16.6	3.3	9.9	3.7	-20.8	3.7	5.7	3.4	-11.3	7.7	8.5	4.5
NI-Elbow-Hand	-12.9	4.2	9.7	5.1	-16.2	2.6	11.7	3.7	-20.2	4.5	8.1	-3.0	-11.2	7.6	9.6	6.3
MS-Head-Hand	-9.4	1.9	-11.6	3.0	-7.1	4.7	-13.8	1.8	-8.0	5.5	-15.6	-2.9	-1.1	8.5	-10.7	4.8
NI-Head-Hand	-11.5	1.5	-4.7	4.8	-11.2	4.8	-4.9	2.9	-12.3	5.2	-8.9	-2.5	-2.4	9.6	-5.3	6.4

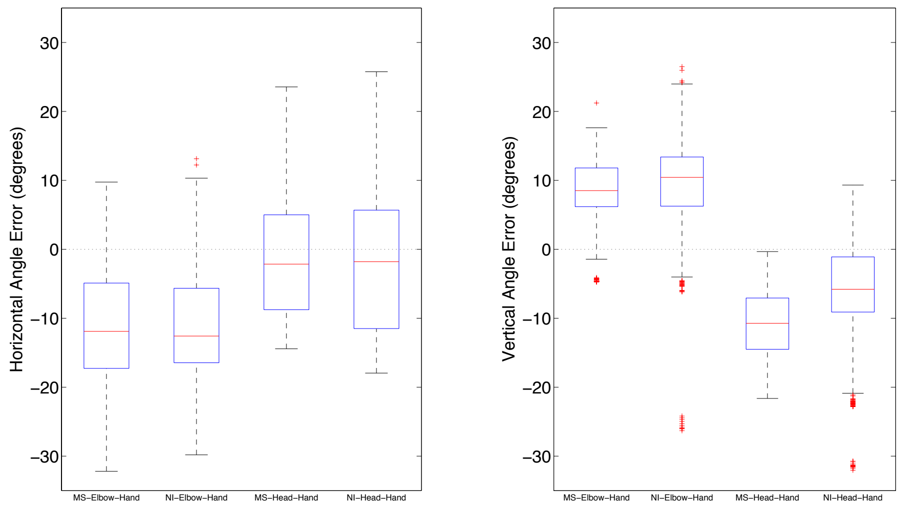


Figure 93: Box plots of the errors in spherical coordinates θ and ψ for each pointing method.

Given a ray $\vec{v}$ in the sensors frame from one of the pointing gesture approaches, and a ground truth target point p_{target} lying on a target planar surface, the ray-plane intersection between $\vec{v}$ and plane was computed for each ray, resulting in a 3D point lying on the plane. Figure 92 shows the 2D projections for all 30 measurements from each subject (shown in different colors) and each target. For ease of display, the 3D intersection coordinates with the target planes are displayed in a 2D coordinate system attached to the target plane, with the ground truth target point as the origin.

As can be seen in Figure 92, the pointing gesture intersection points were fairly consistent across all users, but varied per target location. The elbow-hand method produced similar results for MS-SDK and OpenNI. The same is true for the head-hand method. It is also noteworthy that the data for each target tends to be quite clustered for all methods, and typically not centered on the target location.

6.1.6.2 Angular Error

We computed the mean and standard deviations of the angular errors in the spherical coordinate system for each pointing gesture method and target. Section 6.1.3

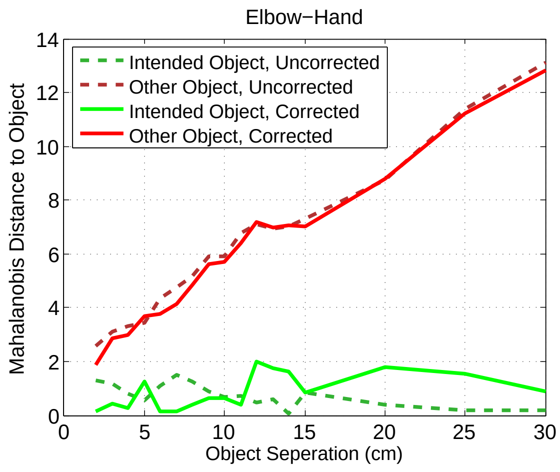
describes in detail how the angles (θ, ψ) are found. The mean and standard deviation analyses are given in Tables 12 and 13. The aggregate errors are also displayed as a box plot in Figure 93.

Several observations can be made from these results. The data from the elbow-hand pointing method reports that users typically point about 11 degrees to the left of the intended target direction, and about 9 above the target direction. Similarly, the data from the head-hand pointing method reports that users typically point about 2 degrees to the left of the intended pointing direction, but with a higher standard deviation than the elbow-hand method. On average, the vertical angle ψ was about 5 degrees below the intended direction for the OpenNI tracker and 10 degrees below for the MS-SDK tracker, with a higher standard deviation than the elbow-hand methods. The disparity between the two skeleton trackers for this pointing method is because they report different points for the head position, with the MS-SDK head position typically being reported higher than the OpenNI head position. The overall performance of the OpenNI and MS-SDK skeleton trackers, however, is fairly similar, with the MS-SDK having slightly less variation for our dataset.

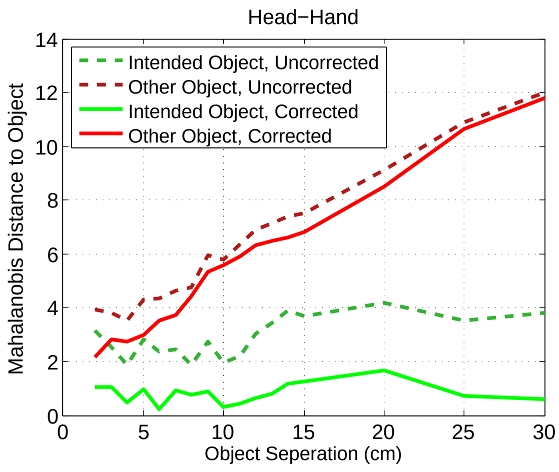
As can be seen in the aggregate box plot in Figure 93, the horizontal angle θ has a higher variation than the vertical angle ψ. Examining the errors for individual target locations shows that this error changes significantly with the target location. As future work, it would be interesting to collect data for a higher density of target locations to attempt to parameterize any angular correction that might be applied.

6.1.7 Pointing Evaluation

Using the error analysis and pointing gesture model we presented in previous sections, we conducted an experiment to determine how our approach distinguished two potentially ambiguous pointing target objects. We use the results of the angular error analysis results and not the euclidean error analysis for the remainder of the paper because of our angular representation of pointing gestures.



(a)



(b)

Figure 94: Resulting Mahalanabis distances of pointing targets from the Object Separation Test is shown for a) Elbow-Hand and b) Head-Hand pointing methods. Intended object are shown in green and other object is in red. Solid lines show distances after correction is applied. Less Mahalanobis distance for intended object is better for reducing ambiguity.

6.1.7.1 Object Separation Test

The setup consisted of a table between the robot and the person and two coke cans on the table (Figure 95) where the separation between objects was varied. To detect the table plane and segment out the objects on top of it, we used the segmentation approach presented in Section 3.4. The objects centroid positions, along with their point clouds were calculated in real-time using our segmentation algorithm. The separation between objects were varied with 1 cm increments from 2-15 cm and with 5 cm increments between 15-30 cm. We could not conduct the experiment below 2 cm separation because of the limitations of our perception system. We used the OpenNI skeleton tracker because rest of our system is based in Linux, and we already found that performance difference with MS-SDK for pointing angle errors is insignificant. The experiment was conducted with one user, who was not in the training dataset. For each separation, the user performed 5 pointing gestures to the object on the right and 5 to the object on the left. The person pointed to one of the objects and the Mahalanobis distance D_{mah} to the intended object and the other object is calculated using the approach in Section 6.1.4. We used the mean and standard deviation values of Target 2 (Figure 90) for this experiment because the objects were located between the robot and the person.

6.1.7.2 Results and Discussion

The results of the object separation experiment is given for Elbow-Hand (Figure 94(a)) and Head-Hand (Figure 94(b)) methods. The graphs plot object separation versus the Mahalanobis distance for the pointed object and the other object for corrected and uncorrected pointing direction. There are several observations we make by looking at these results.

First, the Mahalanobis distance D_{mah} for the intended object was always lower than the other object. The corrected D_{mah} for both Elbow-Hand and Head-Hand methods for the intended object was always below 2, therefore selecting the threshold $D_{thres} = 2$ is a reasonable choice. We notice that some distances for the unintended object at 2cm separation is also below $D_{mah} < 2$. Therefore, when the objects are 2 cm apart, then the pointing target becomes ambiguous for this setup. For separations of 3cm or more, D_{mah} of the unintended object is always over the threshold, therefore there is no ambiguity.

Second, correction significantly improved Head-Hand accuracy at all separations, slightly improved Elbow-Hand between 2-12cm but slightly worsened Elbow-Hand

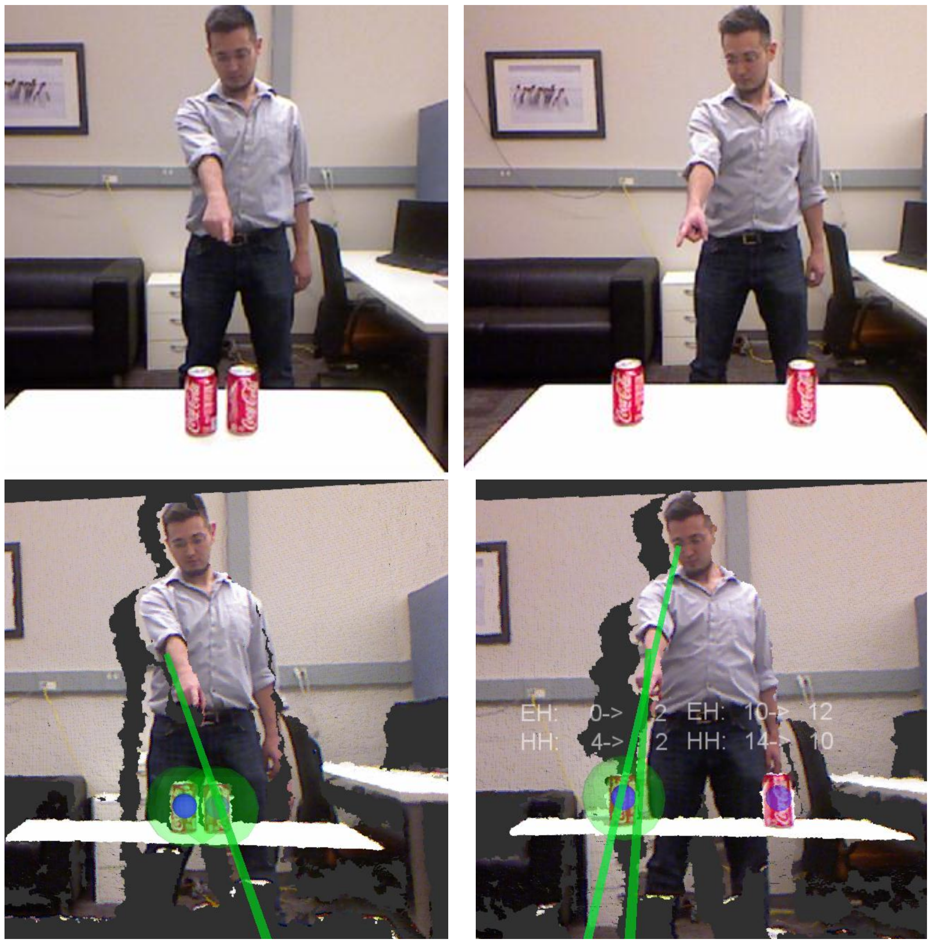


Figure 95: Example scenarios from the object separation test is shown. Our experiments covered separations between 2cm (left images) and 30cm (right images). The object is comfortably distinguished for the 30cm case, whereas the intended target is ambiguous when the targets are 2cm apart. Second row shows the point cloud from Kinect's view. Green lines show the Elbow-Hand and Head-Hand directions whereas green circles show the objects that are within the threshold $D_{mah} < 2$.

after 12cm. We attribute this to the fact that the angles we receive is heavily user-dependent and can have a significant variance across methods as showed in Figure 93. Moreover, the user was not in the training set.

Third, the Mahalanobis distance stayed generally constant for the intended object, which was expected. It linearly increased with separation distance for the other object.

Fourth, patterns for both methods are fairly similar to each other, other than

Head-Hand uncorrected distances being higher than Elbow-Hand.

6.1.8 Conclusions

Robot perception systems that are able to correctly interpret pointing gestures can greatly benefit human-robot interaction. A pointing target detection approach that returns a single hypothesis can lead to failures due to perception error. On the other hand, estimation of a pointing likelihood of nearby objects can give valuable insight to a robot. For example, a robot can ask the user to disambiguate the objects if it perceives more than one object that has a significant likelihood of being referred to.

In this work, we model the uncertainty of pointing gestures in a spherical coordinate system, use this model to determine the correct pointing target, and detect when there is ambiguity. Two pointing methods are evaluated using two skeleton tracking algorithms: Elbow-Hand and Head-Hand rays, using both OpenNI NITE and Microsoft Kinect SDK. A data collection with 6 users and 7 pointing targets was performed, and the data was used to model users pointing behavior. The resulting model was evaluated for its ability to distinguish between potential pointing targets, and to detect when the target is ambiguous. Our evaluation showed that in a scenario where the separation between two objects were varied, our system was able to identify that there is ambiguity for 2 cm separation and comfortably distinguished the intended object for 3 cm or more separation.

6.2 Interactive Map Annotation & Object Modeling

Section 6.1 described our approach to detection of pointing gestures. We now describe how these gestures, combined with a tablet UI can be used to interactively annotate structures and objects in a map, such as the ones produced by OmniMapper (Chapter 5). This work was originally published in [152] and [153].

Maps that include semantic information such as labels for structures, objects, or landmarks can provide context for navigation and planning of tasks, and facilitate interaction with humans. For example, we might want our service robot to be able to accept commands such as “fetch the mug from the kitchen table”. To perform such a task, the robot should know the location and extent of the structure referred to by “kitchen table”. As another example, we might want the robot to “go down the hallway”. To do this, it is useful to have a representation of the location and extent of the hallway. In this way, we must establish “common ground” [26] for communication, by grounding these labels to landmarks in the robot’s map.

A simple approach to adding labels to robot maps would be to attach labels to specific coordinates in the map. For example, we might attach a label “kitchen table” to coordinate $(2.4, 7.12, 0.0)$. Such an approach fails to capture the shape and extent of the structure designated by the label, which might be important to tasks such as object search. In addition, point based references may be ambiguous for a region or volume in a map, such as a hallway or room. For this reason, we will take an approach that allows us to represent both location and extent of landmarks and spaces corresponding to these labels.

We propose a map representation that includes planar patches represented by a plane equation in hessian normal form, a polygon representing the boundary of the surface, and an optional label. Surfaces can be detected from point cloud data, and represented in a consistent map coordinate frame. Maps composed of such features can represent the locations and extents of landmarks such as walls, tables, shelves, and counters. This need not be the only information in the map – occupancy information or other landmark types could also be mapped and used for other purposes such as localization or obstacle avoidance. We employ a SLAM system described in previous

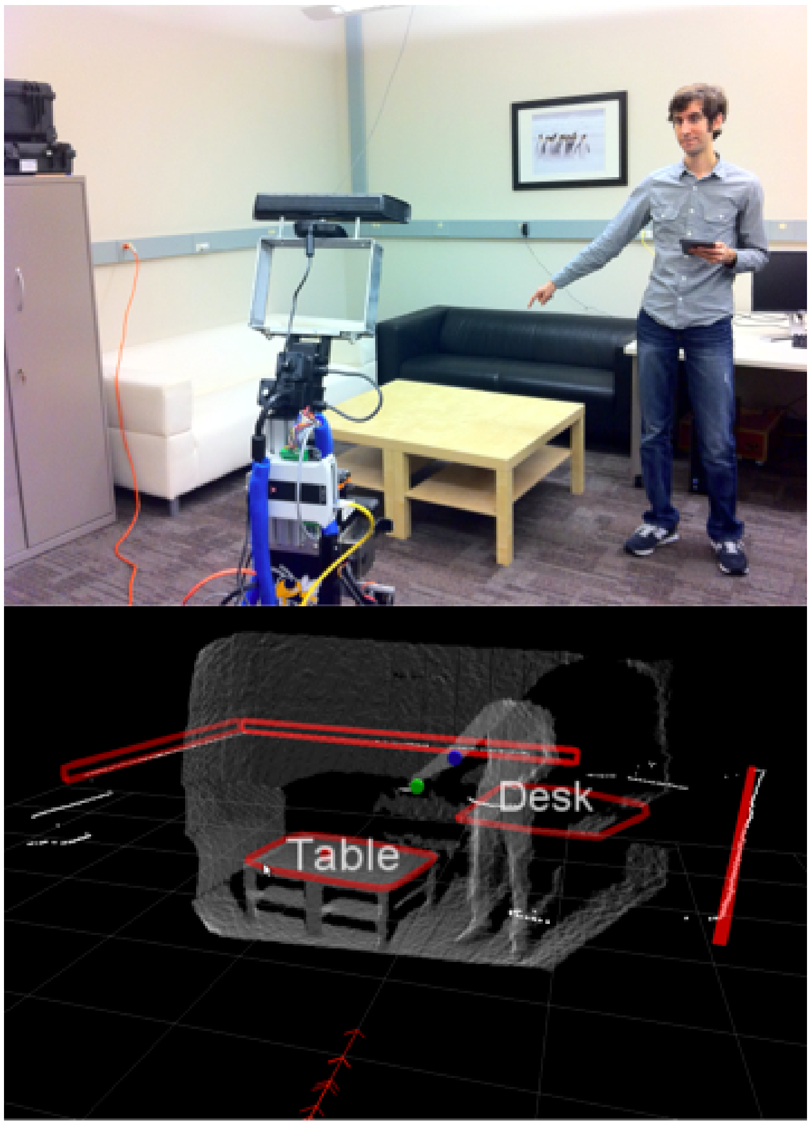


Figure 96: Top: A photo of a user pointing at a nearby table, after entering the desired label on a handheld tablet UI. Bottom: A visualization of the robot's map and sensor data. Red outlines represent the convex hulls of mapped planar features. The detected gesture can be seen via the green sphere at the user's hand, the blue sphere at the user's elbow, and the red sphere (near the "a" in Table) indicating where the pointing gesture intersects a mapped plane.

work [149] that uses such surfaces as landmarks in a feature-based SLAM system.

Many task-relevant landmarks may be represented as single planar structures that have unique labels, such as “coffee table” or “front door”. Other labels might correspond to regions of a map bounded by multiple planar landmarks, such as the walls of a room or hallway. If a user labels multiple planar landmarks with the same label, such as the four walls of a room, or two walls of a hallway, it specifies a region of space corresponding to this label by finding the convex hull of the extent of all planar features with that label. In this way, we can label both specific landmarks such as tables or shelves, as well as regions such as rooms, hallways, or cubicles.

To label the map, we have developed an interactive system that uses a combination of a tablet based user interface and pointing gestures. Using this tablet interface, users can command the robot to follow them through a mapped environment, as well as enter labels for nearby features. Labels are entered using an on-screen keyboard. Pointing gestures are recognized using data from a Microsoft Kinect, and the referenced landmark is annotated with the entered label. An example of the labeling process, as well as a map generated by our system is shown in Figure 96.

6.2.1 Related Work

There are several areas of related research. The most closely related approach to our own is that of Human Augmented Mapping (HAM), introduced by Topp and Christensen in [146] and [145]. The Human Augmented Mapping approach is to have a human assist the robot in the mapping process, and add semantic information to the map. The proposed scenario is to have a human guide a robot on a tour of an indoor environment, adding relevant labels to the map throughout the tour. The HAM approach involves labeling two types of entities: regions, and locations. Regions are meant to represent areas such as rooms or hallways, and serve as containers for locations. Locations are meant to represent specific important places in the environment, such as a position at which a robot should perform a task. This approach was applied to the Cosy Explorer system, described in [166], which includes a semantic mapping system that multi-layered maps, including a metric feature based map, a topological map, as well as detected objects. While the goal of our approach is similar, we use a significantly different map representation, method of labeling, and interaction.

The problem of following a person with a mobile robot has found a lot of interest in the literature. One of the methods to track people is to detect legs in laser scans [5, 144]. One other common method is to use face or blob detection and fuse it with laser-based methods [80]. More recently, RGB-D cameras have been used to detect and follow people. Loper [91] presents a system that follows a person and recognizes speech commands as wells as non-verbal gestures. In that work, specifically designed arm gestures were used to start/stop following and give lower level motion commands to the robot. There also has been work to recognize natural gestures, such as the gesture of pointing to a direction. Nickel [103] estimated the pointing direction by detecting the head and hand in a stereo image. Steifelhagen [135] detected pointing gestures and used it in the context of Human-Robot Interaction. Fong [49] describes a robot that can be controlled by both arm gestures and a handheld device. Dias [38] have used a multi-modal interface combining speech and gesture in a tablet computer for a multi robot - multi user setting. Using touch screen devices as the main user interface is also common for robots [78].

6.2.2 Approach

Our map annotation approach requires three main components: a mapping system, a person following system, and a label annotation user interface. The system diagram shown in Figure 97 demonstrates the relationship between these components. For mapping, we use a SLAM system from the authors' previous work described in [149]. For the map annotation, we utilize a combination of a tablet user interface and gesture recognition. The expected usage scenario is similar to that of the Human Augmented Mapping [145] approach, where a user can command the robot to follow them on a tour of an environment, and stop to annotate important landmarks and regions.

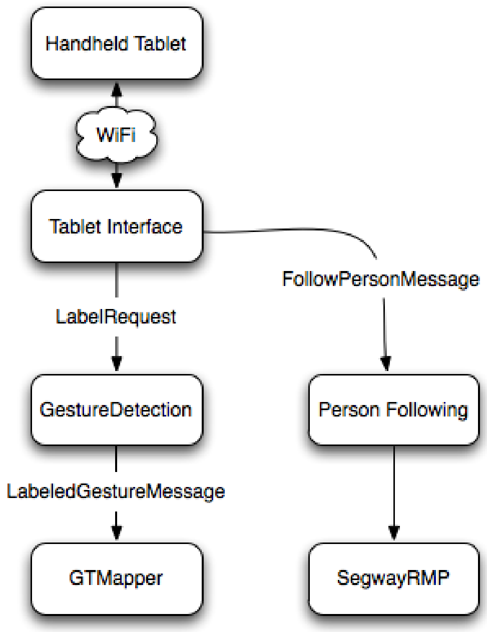


Figure 97: A system diagram of our approach.

6.2.2.1 Map Representation

Many robotic mapping systems employ occupancy-based information. While this can be helpful for localization and path planning purposes, maps may also need to contain other types of spatial information for other applications. Our approach relies on keeping track of a set of observed planar surface as part of the map representation. Planes are represented in the hessian normal form by the well known equation: $ax +$

$by + cz + d = 0$. Since the observed planes do not have infinite extent, we bound the observed area with a polygon – in our case, the convex hull of all observed points on the plane. We allow these planar landmarks to have an optional label. This results in a map that contains a list of planes of the form $p = \left[n, boundary, label\right]$, where $n = \left[a, b, c, d\right]$ and *boundary* is a polygon that bounds the observed planar surface.

Our mapping system has been described in detail in Chapter 4 and Chapter 5. In the context of this system, we use planar landmarks as in Section 4.2.

6.2.3 Gesture Recognition

We use pointing gestures as a natural interface between the robot and the user. The pointing direction is used to determine which planar surface the user is pointing to. To detect people in the environment, we used a Microsoft Kinect RGB-D sensor with OpenNI natural interaction framework for skeleton tracking. OpenNI necessitates the person to make a calibration pose to start skeleton tracking. This limitation actually serves to our purposes since we allow only one user to be in control and the person who does the calibration pose is chosen to be the user. Pointing gesture detection is initiated when a LabelRequest message is received, which contains the label entered by the user. The gesture is then searched for T seconds. Let's denote ${}_se$ and ${}_sh$ as the elbow and hand position of one arm in the sensor's coordinate frame. Let ϕ_g denote a threshold angle above which the user's arm must be raised to be considered a pointing gesture. Two conditions have to be satisfied to trigger a pointing gesture:

The first condition requires the forearm not to be on the side of the body, and the second ensures that the gesture is consistent for some time period. We have used $T = 30s$, $\phi_g = -45\,^{\circ}$ and $\Delta t_g = 0.5s$. Whenever the pointing gesture is triggered, a LabeledGestureMessage is sent to the mapper. The mapper then applies the coordinate transformations ${}_me = ({}_mT_s){}_se$ and ${}_mh = ({}_mT_s){}_sh$, which allows us to find all the planes that intersects the ray emanating from ${}_m e$ towards ${}_m h$ in the map frame. When testing if the ray intersects with a given plane, we also enforce that the intersection point lies within the convex hull of that landmark. If the ray intersects with multiple planar landmarks, the closest one to the user's hand is selected.

6.2.4 Person Following

For a segment, the leg features are calculated and the weighted mahalanobis distance to the mean leg features is calculated. If the mahalanobis distance is below a threshold, the segment is identified as a leg. While the leg is being tracked; a fourth parameter, the proximity of the segment center to the estimated position of the leg, is also considered. The leg is tracked with a Kalman Filter in the odometry frame using a constant velocity model. When no segment is below the leg distance threshold, the tracker expects the leg to reappear in the same location for some time, before declaring the person lost. This allows handling of no-detection frames and temporary occlusions(i.e. another person passes between the tracked person and robot). The mean and variances of leg features are adaptively updated using the last 20 frames, so that the tracker adjusts to the leg shape/clothing of the person after the initial detection. After the command to follow a user is received, the person to follow is selected by detecting a significant movement by a leg segment in the region in front of the robot.

The leg position is considered as the position of the person and the robot's objective is set to be 0.8m away from the person and oriented towards him/her. The goal position is re-calculated at every laser scan and provided to a local navigation planner. We have used the Robot Operating System (ROS) navigation stack to implement the person following behavior.

6.2.5 Tablet UI

A tablet-based interface was developed for interaction between the user and the robot. Similar functionality could be achieved using a speech dialog system, but such interfaces tend to have a high error rate [37], and ultimately we decided to use a the tablet as a more robust way of communicating with the robot. The interface has been implemented on a Samsung Galaxy Tablet running the Android OS. The tablet is equipped with 802.11 WiFi, and can communicate with a ROS node running on our robot via a TCP socket.

The interface contains a series of buttons which can be used via the tablet's touch screen. A screenshot of the tablet's UI is shown in Figure 98. This interface can be used to command the robot to follow the user, stop following the user, annotate landmarks with labels, and navigate to labeled landmarks. Labels are entered via the tablet's on-screen keyboard, and sent to the robot.

6.2.6 Navigating to labeled structures

After labeling various landmarks in the map, we want the robot to be able to navigate back to labeled landmarks. We have implemented a function to calculate a goal location corresponding to a label.

We consider two cases when calculating a goal point for a given label. The requested label may correspond to one landmark, or multiple landmarks. In the case that it is one landmark, it corresponds to a plane somewhere in the environment, but if many landmarks share the label, it corresponds to a volume.

Let us first consider the case where there is only one plane with a given label. In this case, we assume that the robot should navigate to the closest edge of the plane, so we select the closest vertex on the landmark's boundary to the robot's current

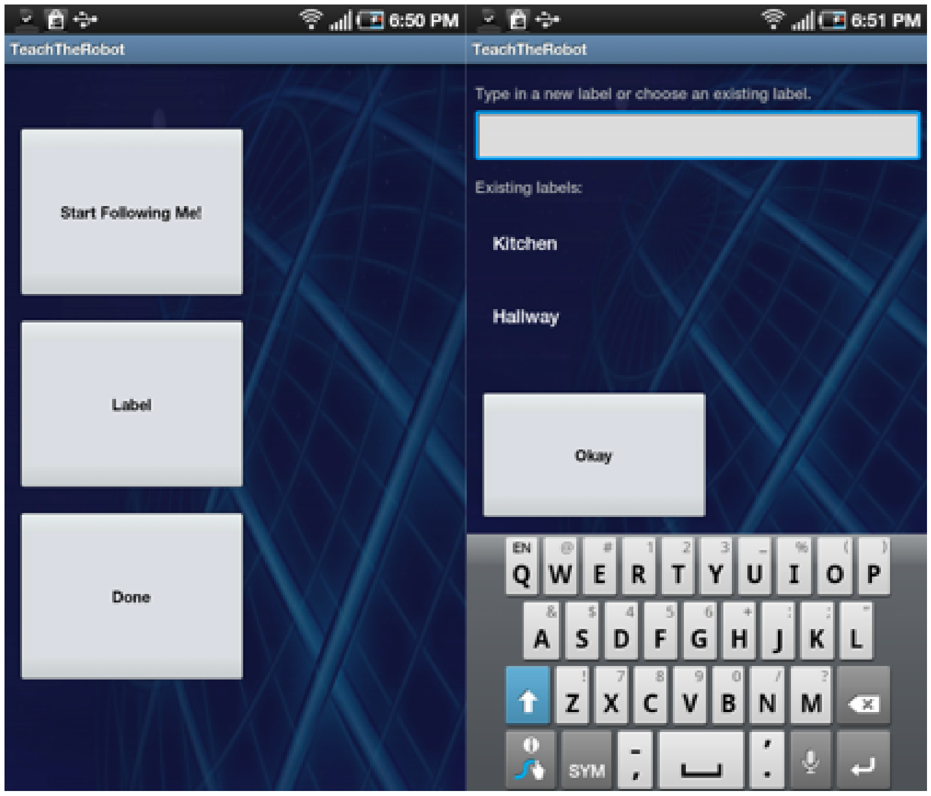


Figure 98: Screenshots of our tablet user interface. Left: A screenshot of the interface that allows the user to command the robot to follow, or stop following them, or enter a label. Right: A screenshot of the interface for labeling landmarks. Previous labels are available for the user's convenience when labeling multiple surfaces with the same label (e.g. the four walls of a room). New labels can be entered via the onscreen keyboard. Upon pressing the "Okay" button, the user points at the landmark that should be annotated with the desired label.

position. This point is projected down to the ground plane, as our robot navigates on the floor. We calculate a line between this point and the robot's current pose, and navigate to a point on this line 1 meter away from the point, and facing this point. This results in the robot navigating to near the desired landmark, and facing it. This method is suitable for both horizontal planes such as tables, or vertical planes such as doors.

In the case that we have multiple planar landmarks annotated with the same label, this label corresponds to a region of space such as a room or corridor. In this case, we project the points of all planes with this label to the ground plane, and compute the convex hull. For the purposes of navigating to this label, we simply navigate to the centroid of the hull. While navigating to a labeled region is a simple task, this labeled region could also be helpful in the context of more complex tasks, such as specifying a finite region of the map for an object search task.

6.2.7 Annotation Results

Preliminary validation of the system has been performed on a robot platform in our lab and office environment. The environment was mapped *a priori* by teleoperating the robot through the environment. The interactive labeling system was then demonstrated on the robot using this map. A formal user study of the system with non-expert users is future work.

6.2.7.1 Robot Platform

The robot platform used in this work is the *Jeeves* robot from the Georgia Institute of Technology's Cognitive Robotics Lab, shown in Figure 100 and 102. Jeeves is comprised of a Segway RMP-200 mobile base, and is equipped with a variety of sensors. A SICK LMS-291 laser scanner is used for navigation and obstacle avoidance.

A Microsoft Kinect RGB-D sensor is mounted on a Directed Perception DP-47-70 pan-tilt unit, allowing us to capture point cloud data, which we require for mapping. The platform is also equipped with a parallel jaw gripper mounted on a 1-DOF linear actuator, which allows basic manipulation tasks when combined with the Segway's additional degrees of freedom.

6.2.7.2 Labeling System

We used our system in an office environment for two types of scenarios. First, labeling single important landmarks, such as the coffee table and desk , seen in Figure 96. Once a metric map was collected, the user used the tablet-based UI to command the robot to follow them through the environment to important landmarks. The tablet UI was used to type the relevant labels for the landmarks, and gestures were used to indicate which landmark should be annotated with this label.

We also tested labeling of regions by labeling multiple surfaces with the same label. For example, a hallway can be seen in Figure 101.

6.2.7.3 Navigating to labeled landmarks and regions

Navigation to labeled landmarks has also been tested by using the tablet-based user interface. An example goal location corresponding to the labeled table is shown in Figure 99. As was described in Section 6.2.6, the goal location is calculated by first selecting the nearest point on the landmark's convex hull to the robot, as is shown as the white sphere in Figure 99. A pose 1m from this point in the direction of the robot, and facing the labeled landmark has been selected as the goal location. The robot then navigates to this point, as can be seen in Figure 100.

Navigation to a labeled region has also been tested. An example of this is shown in Figure 101, which includes a goal point that has been calculated for the label

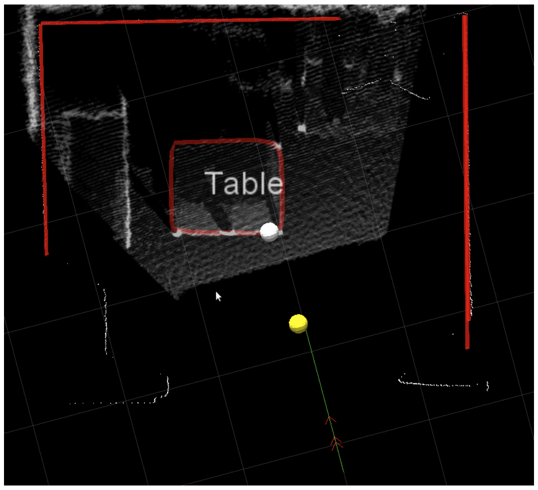


Figure 99: A goal point corresponding to a labeled table is shown in yellow.

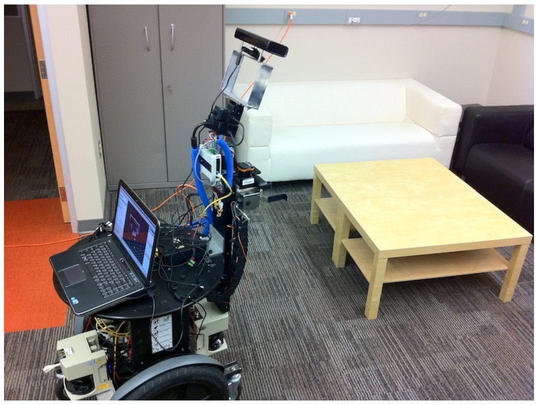


Figure 100: A photo of the robot after navigating to the goal point corresponding to the table.

"hallway". A photo of the robot after navigating to this point is shown in Figure 102.

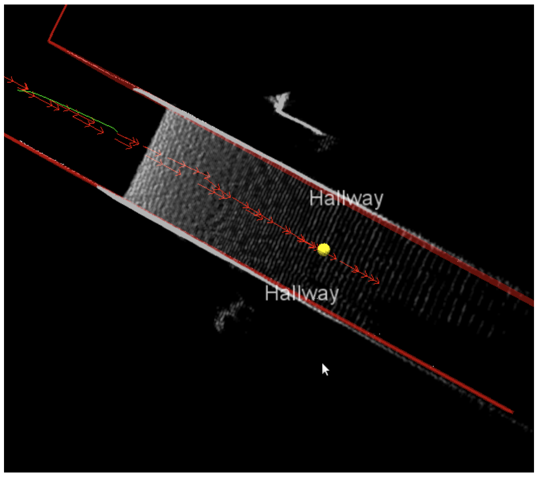


Figure 101: A goal point corresponding to a labeled hallway is shown in yellow.

6.2.8 Labeling Object Models

This approach has also been extended to labeling of object models, such as the objects described in Section 3.5.2. This extension to object labeling was originally published in [153]. Section 6.1 demonstrated how objects can be referenced via pointing gesture. The mobile device UI was updated to include two labeling buttons "label object" and "label map" as shown in Figure 103, so that the intended target type (mapped landmark or object) is provided by the user. The referenced object is then annotated with the label entered by the user (Figure 104), and can then be recognized later (Figure 105). Service robotic tasks that use such a map can then reference the object by label, rather than generating a more complex referring expression (e.g. "the large object" or "the object on the left").

Note that this approach was developed prior to the improved pointing gesture recognition in Section 6.1, and indeed inspired that work. Object labeling with an uncertainty model of pointing can more effectively handle objects that are not well

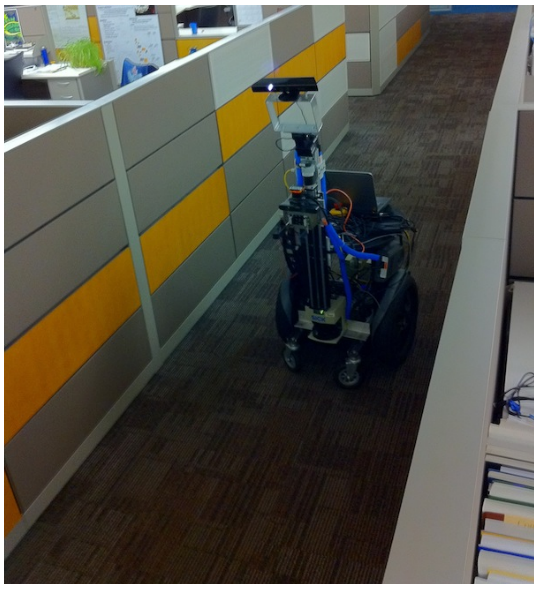


Figure 102: A photo of the robot after navigating to the goal point corresponding to the hallway.

spaced, by detecting that there is ambiguity, and requesting clarification from the user.

6.2.9 Discussion and Conclusion

We have proposed an approach for labeling feature based maps, and demonstrated a system implementing this approach. We described a map representation that supports labeled landmarks, such as tables, counters or shelves, as well as regions, such as rooms or hallways. We also described a user interface that allows users to annotate such maps, by entering labels on a tablet based UI, and gesturing at the landmark that should be annotated with the entered label. Landmarks throughout the environment can be labeled in a "home tour" scenario by making use of our person following system. This approach was validated by implementing and using it on our robot in our laboratory and office environment. Annotated maps were produced, and used to command the robot to go back to labeled landmarks and regions.

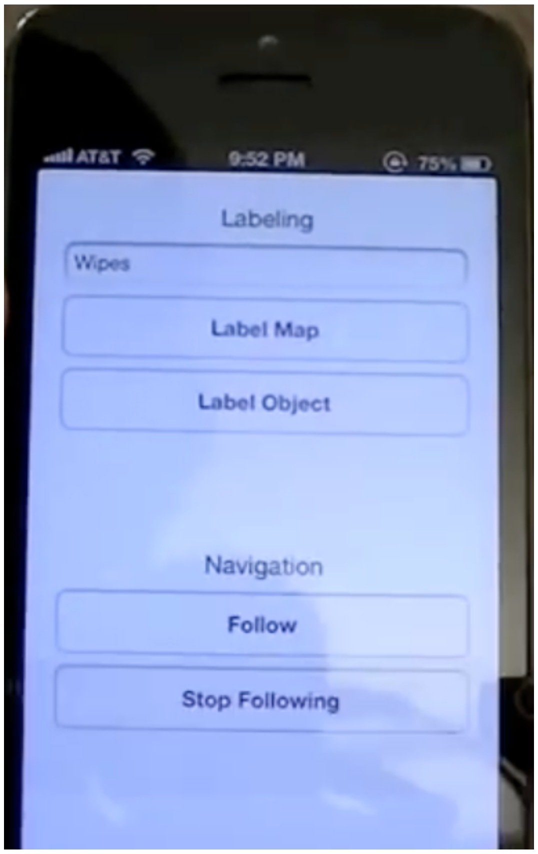
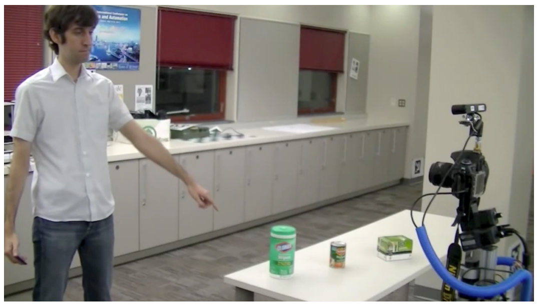
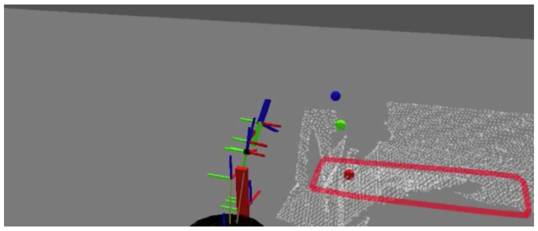


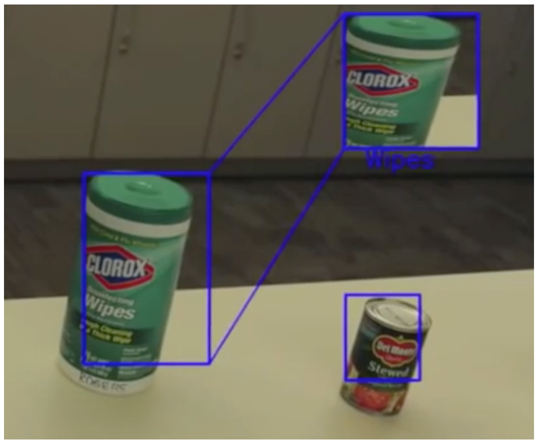
Figure 103: Our smartphone UI for labeling objects and landmarks.



(a)



(b)



(c)

Figure 104: Top: A user pointing at an object. Center: A detected pointing gesture (blue and green spheres), and referenced object (red sphere). Bottom: Features are extracted from an image patch corresponding to the cluster, and annotated with the provided label.

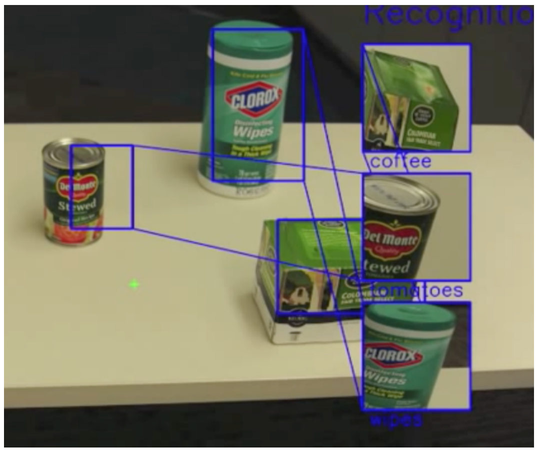


Figure 105: Objects that have been previously annotated with labels can later be recognized, and referenced by the appropriate label.

6.3 Mapping Human Usage of Space

As we move towards having service robots operating in human environments such as homes and offices, it could be useful for these robots to have an understanding of how humans use these spaces. It might be desirable for service robots to understand what areas are heavily used, which are less used, and what areas people tend to spend time in. Towards this goal, we aim to augment a service robot equipped to generate maps using a Simultaneous Localization and Mapping (SLAM) technique with information about humans usage of the different portions of these maps. This work is unpublished joint work with Jayasree Kumar and Andy Echinique.

Maps that include information about humans usage patterns could be useful for a variety of purposes, such as segmentation of the map into different functional regions, informing object search, or informing path planning. If the maps were augmented with information about specific peoples usage patterns, they could be used to find a person for a delivery task, for example, delivering medicine to a specific user at a specified time. In this work, we will investigate a simpler mapping technique, in which we will only map humans usage in aggregate that is, we will only map the total level of usage for each area of the map, and not recording temporal information, or identity of specific users. Such a map could be useful for selecting a standby location for the service robot a location that it can park while not actively performing a task. We aim to show that maps of human usage can be used to better choose standby locations that are out of the way of most tasks of the user, leading to better shared usage of the space. We will introduce a method for augmenting maps with information about human usage, which will be evaluated in the context of a service robot task via a user study. Users will interact with the robot to perform a collaborative task.

6.3.1 Mapping Approach

Our system allows maps used for robot localization to be augmented with information about humans usage. In this work, we used maps generated by the OmniMapper system, as described in Chapter 5. The system described here does not depend on any specific features of this mapping software any map which can allow the robot to localize itself in a global map coordinate frame could be used. In order to begin mapping human usage of space, we assume that the area of interest has already been mapped by a system such as OmniMapper or gmapping, and the robot is well localized in the map. Jeeves is equipped with a Microsoft Kinect, which allowed us to use the OpenNI tools for recognition and pose estimation of humans within the robots environment. During mapping, we use the OpenNI tracker package available in ROS Diamondback in order to track the relative position of a person with respect to the sensor. When this is used on the robot, which is well localized with respect to the map coordinate frame, it is straightforward to determine the persons location in the map frame. During mapping, we record the positions of the persons torso, as output by OpenNIs tracking, and transform this into the map frame. Data is collected by manually teleoperating the robot through the environment to be mapped, and allowing the robot to observe people performing everyday tasks in the environment. In this work, positions of people are recorded at 1 Hz. The result is a set of measurements of person locations in the map frame, as shown in Figure 2.

Once sufficient measurements of people in using the environment have been recorded, we then put these into an occupancy grid, in which each grid cell will represent the amount of measurements of people that were recorded for that cell. This allows us to discretize the space, and reason about usage of discrete areas of the map. This grid will be used for selecting standby locations for the service robot, which will be presented in the following section.

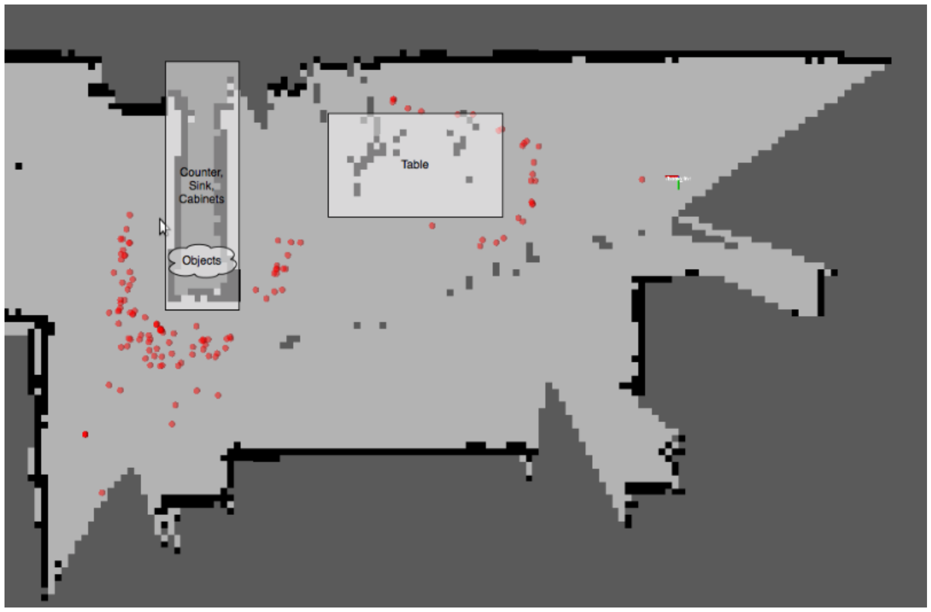


Figure 106: The area mapped by our system with relevant locations labeled, and person measurements shown in red.

6.3.2 Standby Location Selection

In the context of a service robot, we call a standby location a suitable location for the robot to park when not currently working on a task. Ideally, the service robot should be readily available to handle users requests without being in their way. To do this, we make use of the occupancy grid described in the previous section, and select points that represent the center of a selected occupancy grid cell. In our experiment, we will make use of two types of standby locations: low-traffic and high-traffic standby locations. Both types of standby locations are selected by using the occupancy grid, where low-traffic standby locations are relatively unused locations, and have few person measurements in their grid cells, and high-traffic standby locations are highly used locations, and have a large number of person measurements in their corresponding grid cells. We will select n standby locations from our occupancy grid, while requiring that these locations be a specified distance away from each other. In this work, we selected 3 points, and required that they be at least 3 meters away from each other. We additionally require that the points lie within a manually specified bounding rectangle, which constrains the task location. The algorithm used to select these points is relatively simple. The grid cells are sorted based on the number of person measurements recorded in them, starting with either the highest, or lowest cell, depending on whether we are selecting a high-traffic location, or a low-traffic location. Points are selected in order of occupancy, while maintaining our constraint that the points must be at least y meters apart. We always accept the first point, and then iterate along the list of grid cells until we have selected n points that meet these constraints. Points that lie within obstacles such as walls or tables were manually discarded by the experimenters for this work. As future work, we will make use of our occupancy grid map to automatically discard such points.

6.3.3 Experiment & User Study

6.3.3.1 Research Question & Hypothesis

Our research question is: Does the use of maps of human space usage affect human-robot interaction in a collaborative task? . More specifically, we aimed to determine

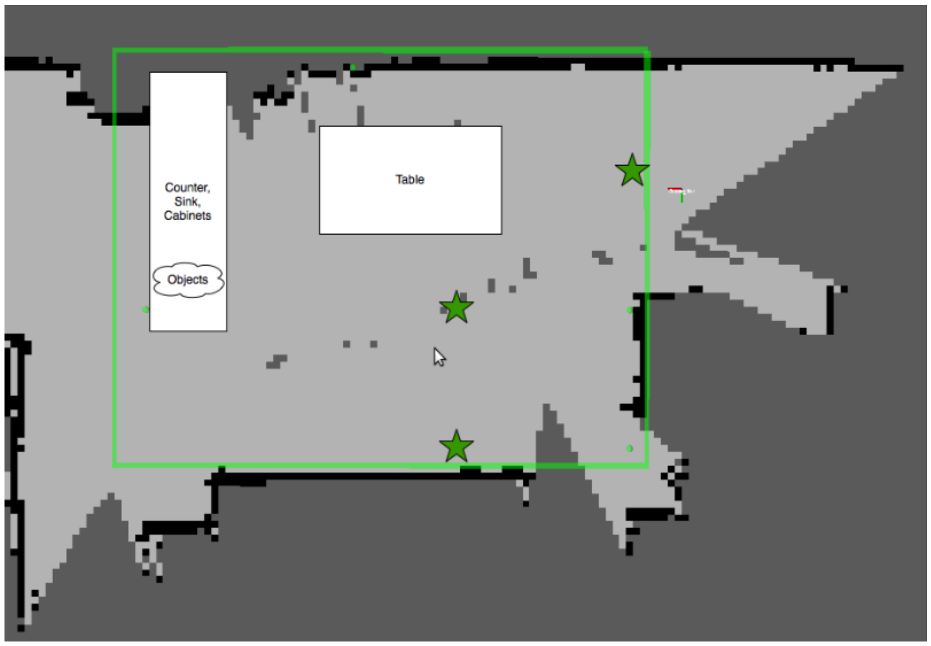


Figure 107: Low traffic locations selected by our system.

whether or not the information stored by our mapping system to select a good standby location for a home service robot is preferred by human users. Our hypothesis is that using maps of human space usage to select low-traffic standby locations will be preferred by users, will lead to less requests for the robot to move out of the way when compared to selecting high-traffic standby locations. The goal of this research was to determine if a robots standby location affected its interaction with participants in a collaborative task. We believed that the robots standby location chosen based on spatial mapping of the participants movement in the restricted task area, does affect the interaction between the human participant and the robot. We argue that the choice of robot standby location is an important contributor to the comfort and joy associated with the execution of a collaborative task between a human and a robot.

The independent variable in our experiment was the the type of standby location selected, which will be either a high-traffic location, or a low-traffic location. The dependent variables we recorded include: the number of robot, move out of my way

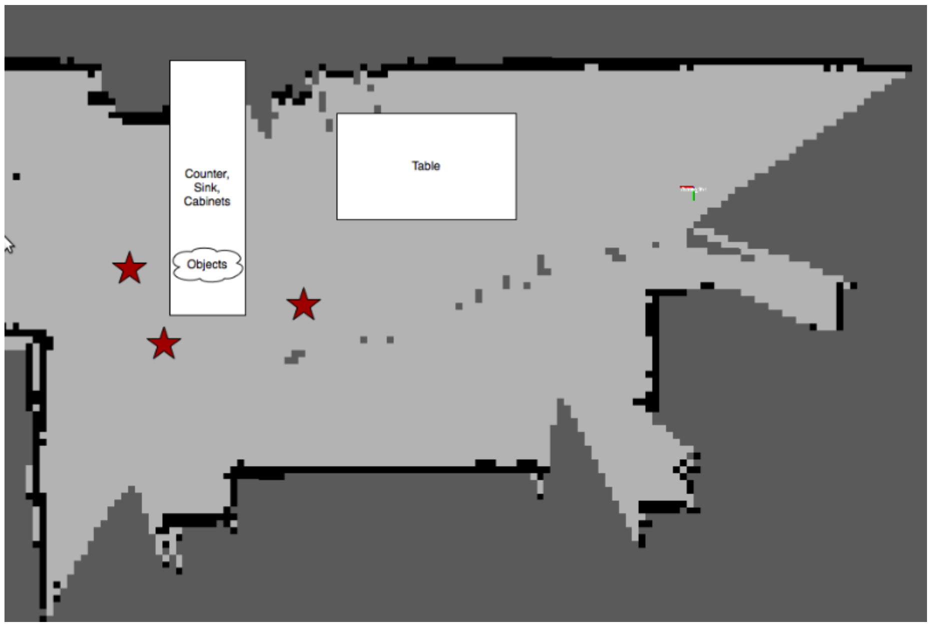


Figure 108: High traffic locations selected by our system.

requests, and user preference as measured by a survey. We performed the study as a between groups study. The survey was an important instrument for measuring our results, so we wanted to ensure that participants will not have to remember two subtly different conditions when responding to survey questions.

6.3.3.2 Scenario

A user study was conducted using the Jeeves robot in the coffee area of Georgia Techs College of Computing building. Recruited participants were asked to set the kitchen table with the robot acting as an assistant by bringing two required items to the user. Participants interacted with the robot through a tablet-based interface, which allowed the user to request objects as well as ask the robot to move out of the way. The exact specification of the setting the table task given to participants is as follows. The task consists of a sequence of sub-tasks, which include:

- Placing a table cloth

- Placing plates on the table
- Placing silverware on the table
- Placing glassware on the table

The tablecloth and silverware are delivered by the robot, and the user is required to get the plates and glassware from the kitchen cabinets. Specific locations are labeled in Figure 106.

Being a between subjects design, participants only run one session with the robot, which was situating itself either in a high or low traffic standby location, both being determined by the mapping system described above. The portions of the study that we controlled include the environment, due to participants performing the study in a controlled lab space with the same environment each time. Another item that was controlled was the participants interaction with the robot, as all participants were constrained to performing the same task, in the same sequence, with the same robot. The amount of time taken by the robot to get from the standby location to the location where it picks up the item was also controlled. To make sure that all participants were waiting the same amount of time for the robot to return with their items, the time for it to return to the person will be standardized across situations, as items are picked up from a fixed location, and also delivered to a fixed location near the table. During the study, participants interaction with the robot was recorded by logging the timestamp of each user action. Video of the interactions was also recorded. Upon completion of the study, participants were asked to fill out a survey asking them their opinions towards their interaction with the robot. They were then interviewed to obtain better clarity on their survey replies and thanked for their time. Two conditions were tested selection of high-traffic standby location by the robot vs. selection of low-traffic standby location by the robot. The robots choice of standby location was varied between the two scenarios and the number of times the human requested the robot to move out of his way was counted. This counting was facilitated through a move button on the tablet interface, touching which caused the robot to choose an alternative standby location and navigate to it. In addition to the subjective measure of the number of times move button was pressed, the human participants feeling about the entire interaction was captured through a survey and a brief interview as a follow up on the survey.

6.3.3.3 Tablet Interface

A tablet-based interface was developed to enable interaction between the human and the robot. A Samsung Galaxy Tab running the Android. Touch and speech were the means of interaction, with touch for the user to make requests for items the robot needs to fetch, and speech for the tablet to communicate the robots intentions. A couple of buttons made up the user-interface, with each button touch transmitting an XML request to the robot via a socket connection. The robot in turn parses the request, navigates to the predetermined fetch location, performs the fetch action and returns with the requested item to a predetermined user location. The following images are screenshots of the tablet user-interface, with the first image depicting the Welcome Activity with an introduction to the task to be performed by the participant in collaboration with the robot, and the second image depicting the Activity where the participant can issue requests to the robot, via the touch interface and receive associated voice feedback.

6.3.4 Results

A total of 8 participants participated in the pilot study. 4 participants were randomly chosen to participate in the low-traffic scenario and 4 others were allotted the high-traffic scenario. The study was carried out as a between-groups study as described above. From the count of the number of times the robot was asked to move out of a human participants way, a significant difference ($p < 0.05$) between was observed between the low- traffic and high-traffic scenarios. As seen in the table, the high-traffic standby location scenario yielded a mean number of move button press of 1.25 with a standard deviation of 0.96. Whereas, the low-traffic standby location scenario produced a mean number of move button presses of 0 with a standard deviation of 0, thus providing proof of choice of standby locations. The observation also revealed that the robot was not asked to move out of the way at all when it chose low-traffic standby locations. An obvious reason that can be attributed to this is that, the robot was never in the way of the human participant and caused no hindrance/intrusion to his task. This is a likely indication that having the robot to choose to remain stationary in a busy standby location does affect human-robot interaction. The set of questions posed to participants as part of the survey after the task is as follows:

- How enjoyable was your interaction with the robot?
- How helpful was the robot?
- How efficient was the robot at retrieving the object requested?
- How would you rate your interaction with the robot via the tablet?
- How intrusive was the robot when stationary and why?

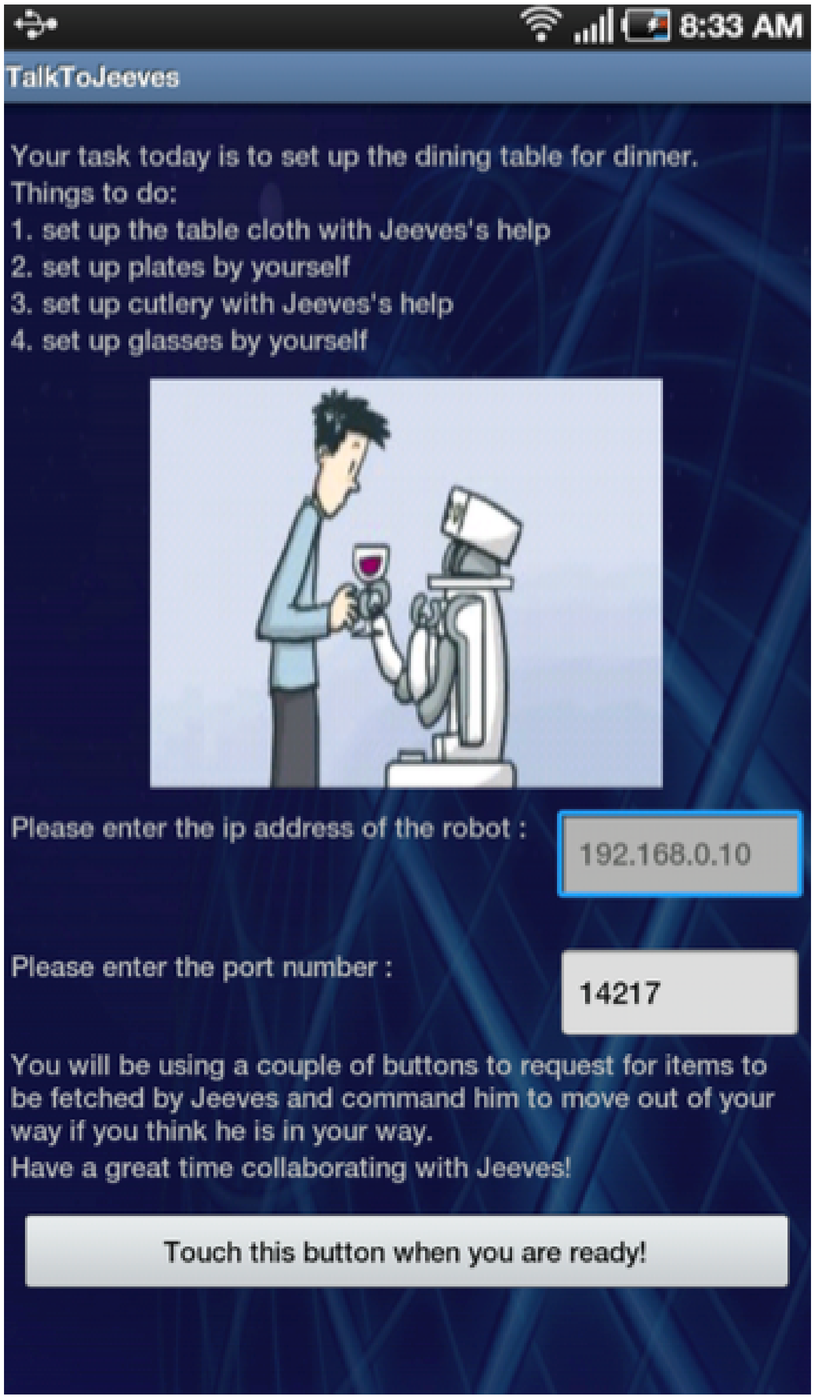
Condition	1	2	3	4	5	Move Button Count
High Traffic	4.5	4	4	4.25	4	1.25
Low Traffic	3.75	4.5	3.75	3.75	3.75	0

The table illustrates the mean and standard deviation pertaining to the survey answers provided by the participants.

Surprisingly, the high-traffic location was considered to be more enjoyable than the low-traffic condition in survey results. Our interview results explain this result as, the participants enjoyed the interaction with the robot in both scenarios, but since the high-traffic scenario demanded more interaction with the robot for asking it to move out of their way, they saw the interaction itself to be more enjoyable. The helpfulness factor was more pronounced in the low-traffic scenario as compared to the high-traffic scenario. From the interview results, it was observed that the participants were satisfied with the reaction speed of the robot on issuing fetch requests using the tablet, but they expressed some dissatisfaction towards the robot not delivering items to their location. We chose a predetermined drop-off location for the purposes of the study, and this is unarguably an avenue for future refinement. The interaction with the tablet did not differ between the two scenarios, so the survey answers to the question pertaining to this were considered as a whole. The mean interaction was rated a mean score of 4, with a standard deviation of 0.926. Multiple participants cited that they preferred interacting with the robot using the tablet to an alternate voice command system (not implemented). Intrusiveness was rated by the participants with mean 2.667 and standard deviation 1.154 for the high traffic standby location scenario, whereas a mean 5 and standard deviation an absolute 0 for the low traffic standby location scenario. Here, 5 denoted a least intrusiveness and 1 maximum intrusiveness. As expected, the participants did feel the robot in high-traffic scenario to be more intrusive to the collaborative task. A summed up result from the survey, the interview and the objective study indicated that when the robot choses a high-traffic standby location participants were less happy about the robot being intrusive. Participants in the complimentary low-traffic scenario, on the other hand, expressed happiness about the interaction as a whole.

6.3.5 Discussion

Participants' main concern was the lack of feedback from the robot. Particularly when the robot was handing off the item, participants were unsure when the robot was going to stop or if it would run into them. If the robot were to provide some feedback on its status, we expect the interaction would be more satisfying. Some users also reported that the robot moved too slowly, so a faster speed may be preferable (while maintaining safety). Another observation was that in this study, participants generally did not carry the tablet with them due to it being difficult to use while performing a task that requires using one's hands. Smartphones are generally easier to pocket than tablets, so this may have been a better choice for such a task. Speech commands would be more convenient, but are still a challenge in real world conditions on mobile robots operating in human environments.


8:33 AM

TalkToJeeves

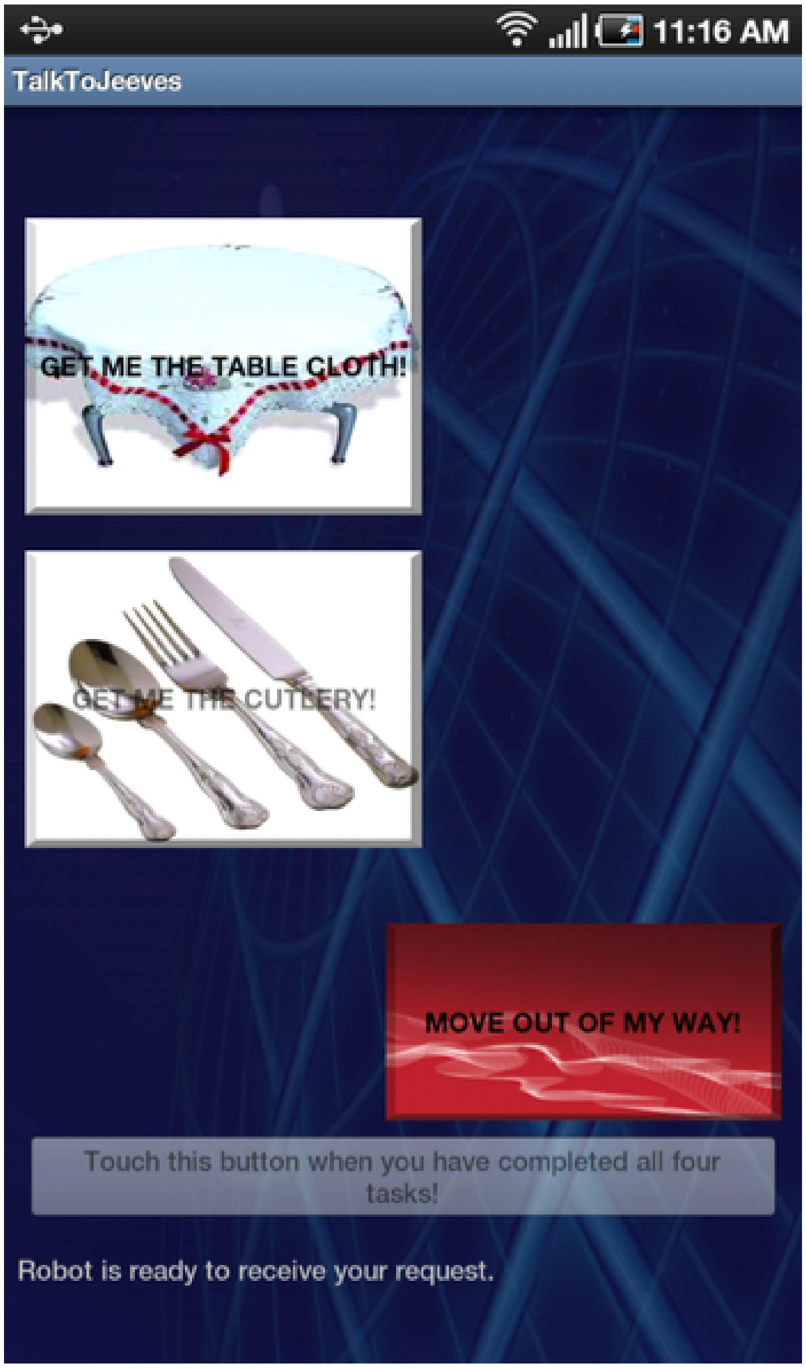
Your task today is to set up the dining table for dinner.
Things to do:
1. set up the table cloth with Jeeves's help
2. set up plates by yourself
3. set up cutlery with Jeeves's help
4. set up glasses by yourself

Please enter the ip address of the robot : 192.168.0.10

Please enter the port number : 14217

You will be using a couple of buttons to request for items to be fetched by Jeeves and command him to move out of your way if you think he is in your way.
Have a great time collaborating with Jeeves!

Figure 109: A screenshot of the tablet UI.



Robot is ready to receive your request.

Figure 110: An additional screenshot of the tablet UI.

CHAPTER VII

CONCLUSIONS & FUTURE WORK

This thesis addressed several challenges: efficient semantic segmentation using RGB-D sensors, map representations that include complex features, flexible multi-modal mapping, and interfaces for interactive annotation of maps to establish common ground. This chapter discusses how and to what degree these were addressed, and outlines future work for next steps on each topic.

To conclude, we restate the thesis statement: Utilizing prior knowledge of semi-structured environments enables semantic feature representations for SLAM and improves data association and loop closure. Such features are meaningful to humans to establish common ground and simplify tasking.

Semantic feature representations presented include planar landmarks (Section 4.2), door sign landmarks (Section 4.3), discovered object landmarks (Section 4.4), and virtual measurements (Section 4.5). Improvements to data association were shown for 3D and 2D measurements on planar landmarks (Section 5.5.2), door sign landmarks (Section 5.5.3), discovered object landmarks (Section 5.5.4), and through detection of moveable object landmarks 4.6.3. Loop closure improvements were shown for both types of object landmarks (Section 5.5.3 and Section 5.5.4). Annotation of semantic features was demonstrated (Section 6.2.7), enabling users to establish common ground with the robot, and task it to navigate to a region or location by name.

7.1 Efficient Semantic Segmentation

Chapter 3 presented our work on per-frame perception techniques applicable to SLAM and semantic mapping. The main contributions of the chapter are the organized connected component algorithm, and the 3D edge detection algorithm. Both are suitable for near-real time operation with VGA point clouds. The organized connected component approach of Section 3.4 has been used with our mapping approach, and has been shown to be suitable for this purpose both in terms of efficiency and quality of the resulting segmentation.

Implementations of both algorithms are available as open source under the BSD license as part of the Point Cloud Library (PCL), and have been used by other groups.

While the work presented has made progress in this area, there are still several challenging areas. As with many computer vision and 3D perception algorithms, out approach breaks down in cluttered environments. Detection of planar surfaces using our approach requires that they be sufficiently clutter-free, so that a large enough portion of the surface can be observed directly. One possibility for addressing this limitation would be to infer the location and extent of the planar surface by the objects resting on (and therefore occluding) it. Our approach to segmenting objects is also affected by clutter, and will under-segment piles of objects, or objects that are placed very close together. As we are interested in service robots, perhaps the most effective means of detecting if a detected cluster corresponds to one object or multiple objects is through physical interaction. Such approaches have been successfully applied to singulation of objects from a pile in [21].

7.2 Semantic Features for SLAM

Chapter 4 presented several high-level feature types for SLAM: large planar surfaces such as walls and tables, and also objects, either modeled a-prior as in Section 4.3, or detected and modeled online during SLAM as in Section 4.4. We demonstrated that these can be used to improve data association and loop closure, as in Section 5.5. Since these features types are discrete entities that have meaning to humans, in addition to aiding the mapping process, maps that include such features can aid service robots in achieving tasks.

While these contributions are a start, there are many more opportunities for improvement and future work in this area. As object recognition was not a focus of this thesis, we used only off-the shelf recognition techniques. Advances in object recognition and pose estimation will improve these techniques. In particular, our approach to object landmarks uses position information, but not orientation information, as we found the object orientation estimates generated by our approach to be quite noisy. Techniques that provide improved pose information could be applied, enabling objects to be represented as 6DOF poses (SE(3)) rather than 3D points, leading to richer landmarks.

Another opportunity for future work is increasing the number of object types used for recognition. As with the door sign landmarks of Section 4.3, with relatively little effort, a human can select common objects that tend to be static and would serve as good landmarks, and train a recognizer. Some examples include light switches, wall sockets, exit signs, and light fixtures. Unlike door signs, most of these are not unique, so the data association approach would need to reason accordingly (as is done with our other landmarks, such as planes).

Section 4.3 discussed how using text read from door signs can inform mapping.

This text is easy to use because we know that it corresponds to a room number and/or person's name. However, there is an opportunity to use text discovered in other (non-door sign) locations for mapping also. Posner *et. al* presented work in this direction in [111]. Combining such a textspotting approach with landmarks such as planes would be of interest, for finding place or building names

7.3 Multi-modal Mapping

The OmniMapper framework as described in Chapter 5 enables multiple feature types and sensors to be used in various combinations. In the work presented here, we have shown that using multiple sensor types together can lead to more features being detected, and more poses where landmarks are sensed (Section 5.5.2). We've also shown that modeling relationships between landmarks can lead to more accurate results (Section 4.5). However, the framework lends itself to usage for more detailed quantitative benchmarking of different feature types, and combinations thereof. Performing such a benchmark is of interest as future work.

Most perception in this thesis has focused on 3D data, such as RGB-D cameras and laser scanners. While mapping using visual features has been addressed widely in the literature, we have not yet implemented visual-only features in OmniMapper, and could improve results. Inertial sensors are also quite powerful, and could benefit the mapping approach, as in [68]. Appearance based loop closure techniques such as FAB-MAP [30] have become popular, and would also be a natural addition to the framework.

In addition to adding new feature and sensor types, another potentially fruitful area of future work is using additional prior knowledge about the structure of the environment. For example, Rogers showed that knowledge of what type of room one is in can inform what objects to search for, and vice versa [118], and that room adjacency models can be learned from readily available data such as books of room plans. Other types of information like this could be added, such as priors on room sizes or layouts, expected furniture, and more.

7.4 Semantic Labeling

Chapter 6 presented a deictic gesture recognition system that can recognize pointing gestures with uncertainty, and an interface for using these gestures with a tablet UI to annotate structures and object models in maps. Such annotated objects can then be referenced by these labels in interactions with the robot. We demonstrated navigation to labeled regions and structures in Section 6.2.7.

While we've used these maps for some basic tasks, as future work it would be of interest to perform more complex tasks as part of a user study. For example, a user's ability to create an annotate maps could be performed, to demonstrate that the system is usable by non-experts. Such a map can then be used for retrieving annotated objects by selecting the desired object by name.

Studying the usage of annotated feature-based maps in other contexts would also be of interest. Directions giving tasks have been studied previously as in [81]. Such tasks could perhaps benefit from richer map representations that include labeled extents and volumes to represent places. Labeled maps could also be of interest for active visual search [7], as labeled surfaces or rooms could be searched first, or named in the query to bound the search space.

REFERENCES

[1] "Willow garage mobile manipulation challenge icra 2013." [Online] Available: `http://www.willowgarage.com/blog/2013/01/08/cfp-icra-2013-mobile-manipulation-challenge`, 2013.

[2] AGARWAL, S., MIERLE, K., and OTHERS, "Ceres solver." [Online] Available: `https://code.google.com/p/ceres-solver/`.

[3] ALEXANDRE, L. A., "3D descriptors for object and category recognition: a comparative evaluation," in *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Vilamoura, Portugal), October 2012.

[4] ALY, A. and TAPUS, A., "An integrated model of speech to arm gestures mapping in human-robot interaction," in *Information Control Problems in Manufacturing*, vol. 14, pp. 817–822, 2012.

[5] ARRAS, K. O., ÓSCAR MARTÍNEZ MOZOS, and BURGARD, W., "Using boosted features for the detection of people in 2d range data," in *Proc. IEEE International Conference on Robotics and Automation (ICRA'07)*, (Rome, Italy), 2007.

[6] AYDEMIR, A., SJÖÖ, K., FOLKESSON, J., PRONOBIS, A., and JENSFELT, P., "Search in the real world: Active visual object search based on spatial relations," in *Proc. Int. Conf. Robotics and Automation (ICRA)*, 2011.

[7] AYDEMIR, A., PRONOBIS, A., GOBELBECKER, M., and JENSFELT, P., "Active visual object search in unknown environments using uncertain semantics," *Robotics, IEEE Transactions on*, vol. 29, no. 4, pp. 986–1002, 2013.

[8] BAILEY, T. and DURRANT-WHYTE, H., "Simultaneous localisation and mapping (SLAM): Part II state of the art," *Robotics and Automation Magazine*, September 2006.

[9] BEESON, P., MACMAHON, M., MODAYIL, J., MURARKA, A., KUIPERS, B., and STANKIEWICZ, B., "Integrating multiple representations of spatial knowledge for mapping, navigation, and communication," in *Symposium on Interaction Challenges for Intelligent Assistants*, AAAI Spring Symposium Series, (Stanford, CA), March 2007. AAAI Technical Report SS-07-04.

[10] Bennewitz, M., Axenbeck, T., Behnke, S., and Burgard, W., "Robust recognition of complex gestures for natural human-robot interaction," in *Proc. of the Workshop on Interactive Robot Learning at Robotics: Science and Systems Conference (RSS)*, 2008.

[11] Besl, P. and McKay, N., "A method for registration of 3-D shapes," *IEEE Transactions on pattern analysis and machine intelligence*, pp. 239–256, 1992.

[12] Bibby, C. and Reid, I., "Simultaneous localisation and mapping in dynamic environments (SLAMIDE) with reversible data association," in *Robotics: Science and Systems*, 2007.

[13] Biswas, J. and Veloso, M., "Depth camera based indoor mobile robot localization and navigation," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1697–1702, IEEE, 2012.

[14] Blodow, N., Marton, Z.-C., Pangercic, D., Rühr, T., Tenorth, M., and Beetz, M., "Inferring generalized pick-and-place tasks from pointing gestures," in *IEEE International Conference on Robotics and Automation (ICRA), Workshop on Semantic Perception, Mapping and Exploration*, 2011.

[15] Bradski, G. and Kaehler, A., *Learning OpenCV: Computer vision with the OpenCV library.* O'Reilly Media, Inc., 2008.

[16] Brooks, A. G. and Breazeal, C., "Working with robots and objects: Revisiting deictic reference for achieving spatial common ground," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pp. 297–304, ACM, 2006.

[17] Canny, J., "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 679–698, 1986.

[18] Castellanos, J., Montiel, J., Neira, J., and Tardós, J., "The SPmap: A probabilistic framework for simultaneous localization and map building," *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 5, pp. 948–952, 1999.

[19] Castle, R. O., Gawley, D. J., Klein, G., and Murray, D., "Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras," *International Conference on Robotics and Automation*, 2007.

[20] Censi, A., "An ICP variant using a point-to-line metric," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, (Pasadena, CA), May 2008.

[21] Chang, L., Smith, J. R., and Fox, D., “Interactive singulation of objects from a pile,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 3875–3882, IEEE, 2012.

[22] Cheng, K. and Takatsuka, M., “Hand pointing accuracy for vision-based interactive systems,” *Human-Computer Interaction–INTERACT 2009*, pp. 13–16, 2009.

[23] Choi, C., Trevor, A. J., and Christensen, H. I., “Rgb-d edge detection and edge-based registration,” in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 1568–1575, IEEE, 2013.

[24] Choudhary, S., Trevor, A. J., Christensen, H. I., and Dellaert, F., “Slam with object discovery, modeling and mapping,” in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 1018–1025, IEEE, 2014.

[25] Cipolla, R. and Hollinghurst, N. J., “Human-robot interface by pointing with uncalibrated stereo vision,” *Image and Vision Computing*, vol. 14, no. 3, pp. 171–178, 1996.

[26] Clark, H. and Brennan, S., “Grounding in communication,” *Perspectives on socially shared cognition*, vol. 13, no. 1991, pp. 127–149, 1991.

[27] Collet Romea, A., Xiong, B., Gurau, C., Hebert, M., and Srinivasa, S., “Exploiting domain knowledge for object discovery,” in *IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, ed.), May 2013.

[28] Cosgun, A., Trevor, A. J. B., and Christensen, H. I., “Uncertainty analysis and modeling of pointing gestures for human-robot interaction,” in *HRI 2015 Workshop “Towards a Framework for Joint Action”*, 2015.

[29] Couprie, C., Farabet, C., LeCun, Y., and Najman, L., “Causal graph-based video segmentation,” in *IEEE Internatinal Conference on Image Processing (ICIP)*, 2013.

[30] Cummins, M. and Newman, P., “Appearance-only slam at large scale with fab-map 2.0,” *The International Journal of Robotics Research*, vol. 30, no. 9, pp. 1100–1123, 2011.

[31] Curless, B. and Levoy, M., “A volumetric method for building complex models from range images,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 303–312, ACM, 1996.

[32] Dalal, N. and Triggs, B., "Histograms of oriented gradients for human detection," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, p. 866, 2005.

[33] Dellaert, F., "Square root SAM: Simultaneous localization and mapping via square root information smoothing," in *Robotics: Science and Systems*, 2005.

[34] Dellaert, F. and Kaess, M., "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1204, 2006.

[35] Dellaert, F., "Factor Graphs and GTSAM: A Hands-on Introduction," 2012.

[36] Dellaert, F. and Kaess, M., "Square Root SAM: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.

[37] Deng, L. and Huang, X., "Challenges in adopting speech recognition," *Communications of the ACM*, vol. 47, no. 1, pp. 69–75, 2004.

[38] Dias, M., Harris, T., Browning, B., Jones, E., Argall, B., Veloso, M., Stentz, A., and Rudnicky, A., "Dynamically formed human-robot teams performing coordinated tasks," in *AAAI Spring Symposium ìTo Boldly Go Where No Human-Robot Team Has Gone Before*, 2006.

[39] Droeschel, D., Stuckler, J., Holz, D., and Behnke, S., "Towards joint attention for a domestic service robot-person awareness and gesture recognition using time-of-flight cameras," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1205–1210, IEEE, 2011.

[40] Droeschel, D., Stuckler, J., and Behnke, S., "Learning to interpret pointing gestures with a time-of-flight camera," in *Human-Robot Interaction (HRI), 2011 6th ACM/IEEE International Conference on*, pp. 481–488, IEEE, 2011.

[41] Durrant-Whyte, H. and Bailey, T., "Simultaneous localisation and mapping (SLAM): Part I the essential algorithms," *Robotics and Automation Magazine*, June 2006.

[42] Ekvall, S., Kragic, D., and Jensfelt, P., "Object detection and mapping for service robot tasks," *Robotica: International Journal of Information, Education and Research in Robotics and Artificial Intelligence*, vol. 25, pp. 175–187, March/April 2007.

[43] Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., and Burgard, W., "An evaluation of the RGB-D SLAM system," pp. 1691–1696, May 2012.

[44] Engelhard, N., Endres, F., Hess, J., Sturm, J., and Burgard, W., "Real-time 3d visual slam with a hand-held rgb-d camera," in *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, vol. 2011, 2011.

[45] Felzenszwalb, P. F. and Huttenlocher, D. P., "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[46] Fischler, M. and Bolles, R., "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Comm. ACM*, vol. 24, pp. 381–395, 1981.

[47] Folkesson, J. and Christensen, H., "Graphical SLAM - a self-correcting map," *International Conference on Robotics and Automation*, 2004.

[48] Folkesson, J., Jensfelt, P., and Christensen, H. I., "The M-Space Feature Representation for SLAM," *IEEE Transactions on Robotics*, vol. 23, no. 5, pp. 1024–1035, 2007.

[49] Fong, T., Conti, F., Grange, S., and Baur, C., "Novel interfaces for remote driving: gesture, haptic and pda," *SPIE Telemanipulator and Telepresence Technologies VII, Boston, MA*, 2000.

[50] Fowler, M., "Inversion of Control Containers and the Dependency Injection pattern." [Online] Available: `http://www.martinfowler.com/articles/injection.html`.

[51] Grisetti, G., Stachniss, C., and Burgard, W., "Non-linear constraint network optimization for efficient map learning," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, pp. 428–439, 2009.

[52] Gumhold, S., Wang, X., and MacLeod, R., "Feature extraction from point clouds," in *Proc. 10th Int. Meshing Roundtable*, pp. 293–305, 2001.

[53] Hähnel, D., Schulz, D., and Burgard, W., "Map building with mobile robots in populated environments," *International Conference on Intelligent Robots and Systems*, pp. 496–501, 2002.

[54] Hähnel, D., Triebel, R., Burgard, W., and Thrun, S., "Map building with mobile robots in dynamic environments," *International Conference on Robotics and Automation*, pp. 1557–1563, 2003.

[55] Harris, C. and Stephens, M., "A combined corner and edge detector," *Alvey Vision Conference*, pp. 141–151, 1988.

[56] Hato, Y., Satake, S., Kanda, T., Imai, M., and Hagita, N., "Pointing to space: modeling of deictic interaction referring to regions," in *Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction*, pp. 301–308, IEEE Press, 2010.

[57] Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D., "RGB-D Mapping: Using depth cameras for dense 3D modeling of indoor environments," in *the 12th International Symposium on Experimental Robotics (ISER)*, 2010.

[58] Herbst, E., Henry, P., Ren, X., and Fox, D., "Toward object discovery and modeling via 3-D scene comparison," in *International Conference on Robotics and Automation*, 2011.

[59] Hoiem, D. and Savarese, S., *Representations and Techniques for 3D Object Recognition and Scene Interpretation.* Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2011.

[60] Holz, D. and Behnke, S., "Fast range image segmentation and smoothing using approximate surface reconstruction and region growing," in *Intelligent Autonomous Systems 12*, pp. 61–73, Springer, 2013.

[61] Holz, D., Holzer, S., Rusu, R. B., and Behnke, S., "Real-time plane segmentation using rgb-d cameras," in *RoboCup 2011: Robot Soccer World Cup XV*, pp. 306–317, Springer, 2012.

[62] Holzer, S., Rusu, R., Dixon, M., Gedikli, S., and Navab, N., "Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 2684–2689, IEEE, 2012.

[63] Horn, B. K. and others, "Closed-form solution of absolute orientation using unit quaternions," *Journal of the Optical Society of America A*, vol. 4, no. 4, pp. 629–642, 1987.

[64] Hosoya, E., Sato, H., Kitabata, M., Harada, I., Nojima, H., and Onozawa, A., "Arm-pointer: 3d pointing interface for real-world interaction," in *Computer Vision in Human-Computer Interaction*, pp. 72–82, Springer, 2004.

[65] Hou, X. and Zhang, L., "Saliency detection: A spectral residual approach," *CVPR*, 2007.

[66] Hu, K., Canavan, S., and Yin, L., “Hand pointing estimation for human computer interaction based on two orthogonal-views,” in *Proc. Intl. Conf. on Pattern Recognition (ICPR)*, pp. 3760–3763, 2010.

[67] Hulik, R., Beran, V., Spanel, M., Krsek, P., and Smrz, P., “Fast and accurate plane segmentation in depth maps for indoor scenes,” in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 1665–1670, IEEE, 2012.

[68] Indelman, V., Williams, S., Kaess, M., and Dellaert, F., “Factor graph based incremental smoothing in inertial navigation systems,” in *Information Fusion (FUSION), 2012 15th International Conference on*, pp. 2154–2161, IEEE, 2012.

[69] Jojic, N., Brumitt, B., Meyers, B., Harris, S., and Huang, T., “Detection and estimation of pointing gestures in dense disparity maps,” in *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pp. 468–475, IEEE, 2000.

[70] Kaess, M., Ranganathan, A., and Dellaert, F., “Fast incremental square root information smoothing,” in *Internation Joint Conference on Artificial Intelligence*, 2007.

[71] Kaess, M., Ranganathan, A., and Dellaert, F., “iSAM: Incremental smoothing and mapping,” *IEEE Transactions on Robotics*, 2008.

[72] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. J., and Dellaert, F., “iSAM2: Incremental smoothing and mapping using the Bayes tree,” *The International Journal of Robotics Research*, 2011.

[73] Kahn, R. E. and Swain, M. J., “Understanding people pointing: The perseus system,” in *Computer Vision, 1995. Proceedings., International Symposium on*, pp. 569–574, IEEE, 1995.

[74] Karpathy, A., Miller, S., and Fei-Fei, L., “Object discovery in 3d scenes via shape analysis,” in *IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, ed.), May 2013.

[75] Kehl, R. and Van Gool, L., “Real-time pointing gesture recognition for an immersive environment,” in *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pp. 577–582, IEEE, 2004.

[76] Kemp, C. C., Anderson, C. D., Nguyen, H., Trevor, A. J., and Xu, Z., “A point-and-click interface for the real world: laser designation of objects for mobile manipulation,” in *Human-Robot Interaction (HRI), 2008 3rd ACM/IEEE International Conference on*, pp. 241–248, IEEE, 2008.

[77] Kerl, C., Sturm, J., and Cremers, D., "Robust odometry estimation for rgb-d cameras," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2013.

[78] Keskinpala, H., Adams, J., and Kawamura, K., "Pda-based human-robotic interface," in *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, vol. 4, pp. 3931–3936, IEEE, 2003.

[79] Kim, D., , and Nevatia, R., "A method for recognition and localization of generic objects for indoor navigation," *Image and Vision Computing*, vol. 16, pp. 729–743, 1994.

[80] Kleinehagenbrock, M., Lang, S., Fritsch, J., Lomker, F., Fink, G. A., and Sagerer, G., "Person tracking with a mobile robot based on multi-modal anchoring," 2002.

[81] Kollar, T., Tellex, S., Roy, D., and Roy, N., "Toward understanding natural language directions," in *Proceeding of the 5th ACM/IEEE international conference on Human-robot interaction*, pp. 259–266, ACM, 2010.

[82] Koppula, H. S., Anand, A., Joachims, T., and Saxena, A., "Semantic labeling of 3d point clouds for indoor scenes," in *NIPS*, pp. 244–252, 2011.

[83] Kowadlo, G., Ye, P., and Zukerman, I., "Influence of gestural salience on the interpretation of spoken requests.," in *INTERSPEECH*, pp. 2034–2037, 2010.

[84] Krainin, M., Curless, B., and Fox, D., "Autonomous Generation of Complete 3D Object Models Using Next Best View Manipulation Planning," in *Proc. International Conference on Robotics and Automation*, 2011.

[85] Kröse, B., "An efficient representation of the robot's environment," in *in: Proceedings Intelligent Autonomous Systems, IOS*, 2000.

[86] Kruijff, G., Zender, H., Jensfelt, P., and Christensen, H., "Clarification dialogues in human-augmented mapping," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pp. 282–289, ACM, 2006.

[87] Kuipers, B., "The spatial semantic hierarchy," *Artificial Intelligence*, vol. 119, pp. 191–233, May 2000.

[88] Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W., "g^2o: A General Framework for Graph Optimization: A General Framework for Graph Optimization," in *International Conference on Robotics and Automation*, 2011.

[89] Leutenegger, S., Furgale, P., Rabaud, V., Chli, M., Konolige, K., and Siegwart, R., "Keyframe-based visual-inertial slam using nonlinear optimization," *Robotics, Science and Systems*, 2013.

[90] Li, Z., Hofemann, N., Fritsch, J., and Sagerer, G., "Hierarchical modeling and recognition of manipulative gesture," in *Proc. of the Workshop on Modeling People and Human Interaction at the IEEE Int. Conf. on Computer Vision*, vol. 77, 2005.

[91] Loper, M., Koenig, N., Chernova, S., Jones, C., and Jenkins, O., "Mobile human-robot teaming with environmental tolerance," in *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, pp. 157–164, ACM, 2009.

[92] Lorensen, W. E. and Cline, H. E., "Marching cubes: A high resolution 3d surface construction algorithm," in *ACM Siggraph Computer Graphics*, vol. 21, pp. 163–169, ACM, 1987.

[93] Lowe, D., "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, 2004.

[94] Martin, C., Steege, F.-F., and Gross, H.-M., "Estimation of pointing poses for visually instructing mobile robots under real world conditions," *Robotics and Autonomous Systems*, vol. 58, no. 2, pp. 174–185, 2010.

[95] Matikainen, P., Pillai, P., Mummert, L., Sukthankar, R., and Hebert, M., "Prop-free pointing detection in dynamic cluttered environments," in *Automatic Face & Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on*, pp. 374–381, IEEE, 2011.

[96] Matuszek, C., Bo, L., Zettlemoyer, L., and Fox, D., "Learning from unscripted deictic gesture and language for human-robot interactions," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[97] Miller, S. D., "CPU TSDF Library." [Online] Available: `https://github.com/sdmiller/cpu_tsdf`.

[98] Mozos, O. M., Triebel, R., Jensfelt, P., Rottmann, A., and Burgard, W., "Supervised semantic labeling of places using information extracted from sensor data," *Robotics and Autonomous Systems (RAS)*, vol. 55, pp. 391–402, May 2007.

[99] Neira, J. and Tardós, J. D., "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 890–897, Dec 2001.

[100] Newcombe, R. A., Davison, A. J., Izadi, S., Kohli, P., Hilliges, O., Shotton, J., Molyneaux, D., Hodges, S., Kim, D., and Fitzgibbon, A., "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pp. 127–136, IEEE, 2011.

[101] Newman, P., Cole, D., and Ho, K., "Outdoor slam using visual appearance and laser ranging," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 1180–1187, IEEE, 2006.

[102] Nguyen, V., Martinelli, A., Tomatis, N., and Siegwart, R., "A comparison of line extraction algorithms using 2D laser rangefinder for indoor mobile robotics," *International Conference on Intelligent Robots and Systems*, 2005.

[103] Nickel, K. and Stiefelhagen, R., "Pointing gesture recognition based on 3d-tracking of face, hands and head orientation," in *Proceedings of the 5th international conference on Multimodal interfaces*, pp. 140–146, ACM, 2003.

[104] Nüchter, A. and Hertzberg, J., "Towards semantic maps for mobile robots," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 915–926, 2008.

[105] Nüchter, A., Wulf, O., Lingemann, K., Hertzberg, J., Wagner, B., and Surmann, H., "3D mapping with semantic knowledge," *RoboCup 2005: Robot Soccer World Cup IX*, pp. 335–346, 2006.

[106] Ohtake, Y., Belyaev, A., and Seidel, H. P., "Ridge-valley lines on meshes via implicit surface fitting," in *ACM Trans. Graphics*, vol. 23, pp. 609–612, 2004.

[107] Pathak, K., Birk, A., Vaskevicius, N., Pfingsthorn, M., Schwertfeger, S., and Poppinga, J., "Online three-dimensional SLAM by registration of large planar surface segments and closed-form pose-graph relaxation," *Journal of Field Robotics*, vol. 27, no. 1, pp. 52–84, 2010.

[108] Pathak, K., Vaskevicius, N., Poppinga, J., Pfingsthorn, M., Schwertfeger, S., and Birk, A., "Fast 3D mapping by matching planes extracted from range sensor point-clouds," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 1150–1155, IEEE, 2009.

[109] Pauly, M., Keiser, R., and Gross, M., "Multi-scale feature extraction on point-sampled surfaces," in *Computer Graphics Forum*, vol. 22, pp. 281–289, 2003.

[110] Poppinga, J., Vaskevicius, N., Birk, A., and Pathak, K., "Fast plane detection and polygonalization in noisy 3d range images," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 3378–3383, IEEE, 2008.

[111] Posner, I., Corke, P., and Newman, P. M., “Using text-spotting to query the world.,” in *IROS*, vol. 10, pp. 3181–3186, 2010.

[112] Pronobis, A., “Semantic mapping with mobile robots,” *PhD Thesis*, 2011.

[113] Pronobis, A., Martínez Mozos, O., Caputo, B., and Jensfelt, P., “Multi-modal semantic place classification,” *The International Journal of Robotics Research*, vol. 29, no. 2-3, p. 298, 2010.

[114] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y., “Ros: an open-source robot operating system,” in *ICRA workshop on open source software*, vol. 3, 2009.

[115] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T. B., Leibs, J., Wheeler, R., and Ng, A. Y., “ROS: an open-source robot operating system,” *International Conference on Robotics and Automation*, 2009.

[116] Quintero, C. P., Fomena, R. T., Shademan, A., Wolleb, N., Dick, T., and Jagersand, M., “Sepo: Selecting by pointing as an intuitive human-robot command interface,” in *IEEE Int. Conference of Robotics and Automation, Karslruhe, Germany*, 2013.

[117] Raza Abidi, S. S., Williams, M., and Johnston, B., “Human pointing as a robot directive,” in *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, pp. 67–68, IEEE Press, 2013.

[118] Rogers, J. and Christensen, H. I., “Robot planning with a semantic map,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 2239–2244, IEEE, 2013.

[119] Rogers, J., Trevor, A., Nieto-Granda, C., and Christensen, H., “SLAM with Expectation Maximization for moveable object tracking,” in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 2077–2082, IEEE, 2010.

[120] Rogers III, J. G., Trevor, A. J. B., Nieto-Granda, C., and Christensen, H., “Simultaneous localization and mapping with learned object recognition and semantic data association,” in *IEEE Conference on Intelligent Robots and Systems*, 2011.

[121] Rottmann, A., Mozos, Ó. M., Stachniss, C., and Burgard, W., “Semantic place classification of indoor environments with mobile robots using boosting,” in *AAAI’05: Proceedings of the 20th national conference on Artificial intelligence*, pp. 1306–1311, AAAI Press, 2005.

[122] Rusu, R. B. and Cousins, S., “3D is here: Point Cloud Library (PCL),” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[123] Rusu, R. B. and Cousins, S., “3d is here: Point cloud library (pcl),” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1–4, IEEE, 2011.

[124] Rusu, R., Blodow, N., Marton, Z., and Beetz, M., “Close-range Scene Segmentation and Reconstruction of 3D Point Cloud Maps for Mobile Manipulation in Human Environments,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), St. Louis, MO, USA*, 2009.

[125] Rusu, R., Marton, Z., Blodow, N., Holzbach, A., and Beetz, M., “Model-based and learned semantic object labeling in 3D point cloud maps of kitchen environments,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), St. Louis, MO, USA*, 2009.

[126] Schmidt, J., Hofemann, N., Haasch, A., Fritsch, J., and Sagerer, G., “Interacting with a mobile robot: Evaluating gestural object references,” in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 3804–3809, IEEE, 2008.

[127] Segal, A., Haehnel, D., and Thrun, S., “Generalized-icp.,” in *Robotics: Science and Systems*, vol. 2, 2009.

[128] Shapiro, L. and Stockman, G. C., *Computer Vision*, ch. 3, pp. 69–75. Prentice Hall, 2001.

[129] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A., “Real-time human pose recognition in parts from single depth images,” in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 1297–1304, IEEE, 2011.

[130] Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., Cook, M., and Moore, R., “Real-time human pose recognition in parts from single depth images,” *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, 2013.

[131] Smith, R., “An overview of the tesseract OCR engine,” *Proceedings of the ICDAR*, 2007.

[132] Smith, R. and Cheeseman, P., “On the representation and estimation of spatial uncertainty,” *International Journal of Robotics Research*, vol. 5, pp. 56–68, Winter 1987.

[133] Stachniss, C. and Burgard, W., "Mobile robot mapping and localization in non-static environments," *Proceedings of the National Conference on Artificial Intelligence*, vol. 20, no. 3, 2005.

[134] Steder, B., Rusu, R. B., Konolige, K., and Burgard, W., "Point feature extraction on 3d range scans taking into account object boundaries," in *Robotics and Automation (ICRA)*, pp. 2601–2608, IEEE, 2011.

[135] Stiefelhagen, R., Fugen, C., Gieselmann, R., Holzapfel, H., Nickel, K., and Waibel, A., "Natural human-robot interaction using speech, head pose and gestures," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, pp. 2422–2427, IEEE, 2004.

[136] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D., "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[137] Sucan, I. A. and Chitta, S., "MoveIt!." [Online] Available: `http://moveit.ros.org`.

[138] Szeliski, R. and Torr, P. H., "Geometrically constrained structure from motion: Points on planes," in *3D Structure from Multiple Images of Large-Scale Environments*, pp. 171–186, Springer, 1998.

[139] Thomaz, A. L. and Cakmak, M., "Learning about objects with human teachers," in *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, pp. 15–22, ACM, 2009.

[140] Thrun, S., Burgard, W., and Fox, D., *Probabilistic robotics.* MIT Press, 2005.

[141] Tombari, F., Salti, S., and Di Stefano, L., "A combined texture-shape descriptor for enhanced 3d feature matching," in *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pp. 809–812, 2011.

[142] Tomono, M. and Yuta, S., "Mobile robot navigation in indoor environments using object and character recognition," in *International Conference on Robotics and Automation*, 2000.

[143] Tomono, M. and Yuta, S., "Object-based localization and mapping using loop constraints and geometric prior knowledge," in *International Conference on Robotics and Automation*, 2003.

[144] Topp, E. A. and Christensen, H. I., "Tracking for following and passing persons," in *Intl Conf. on Intelligent Robotics and Systems (IROS)*, (Edmundton, Canada), pp. 70–77, Aug. 2005.

[145] Topp, E. A. and Christensen, H. I., "Detecting region transitions for human-augmented mapping," *Robotics, IEEE Transactions on*, pp. 1–5, 2010.

[146] Topp, E. A. and Christensen, H. I., "Topological modelling for human augmented mapping," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 2257–2263, Oct. 2006.

[147] Topp, E. A., *Human-robot interaction and mapping with a service robot: Human augmented mapping.* PhD thesis, Swedish School of Sport and Health Sciences, GIH, 2008.

[148] Trevor, A., Gedikli, S., Rusu, R., and Christensen, H., "Efficient organized point cloud segmentation with connected components," *Semantic Perception Mapping and Exploration (SPME)*, 2013.

[149] Trevor, A. J. B., Rogers III, J. G., and Christensen, H. I., "Planar Surface SLAM with 3D and 2D Sensors," *International Conference on Robotics and Automation*, 2012.

[150] Trevor, A. J. B., Rogers III, J. G., Nieto-Granda, C., and Christensen, H., "Applying domain knowledge to SLAM using virtual measurements," *International Conference on Robotics and Automation*, 2010.

[151] Trevor, A. J. B., Rogers III, J. G., Nieto-Granda, C., and Christensen, H., "Tables, counters, and shelves: Semantic mapping of surfaces in 3d," in *IROS Workshop on Semantic Mapping and Autonomous Knowledge Acquisition*, 2010.

[152] Trevor, A. J. B., Cosgun, A., Kumar, J., and Christensen, H. I., "Interactive map labeling for service robots," in *IROS 2012 Workshop on Active Semantic Perception*, 2012.

[153] Trevor, A. J. B., Rogers III, J. G., Cosgun, A., and Christensen, H. I., "Interactive object modeling & labeling for service robots," in *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, pp. 421–422, IEEE Press, 2013.

[154] Trevor, A. J., Rogers, J., and Christensen, H. I., "Planar surface slam with 3d and 2d sensors," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 3041–3048, IEEE, 2012.

[155] Trevor, A. J., Rogers, J. G., and Christensen, H. I., "Omnimapper: A modular multimodal mapping framework," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 1983–1990, IEEE, 2014.

[156] Vosselman, G., Gorte, B. G., Sithole, G., and Rabbani, T., "Recognising structure in laser scanner point clouds," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 46, no. 8, pp. 33–38, 2004.

[157] Wagner, D. and Schmalstieg, D., "ARToolKitPlus for Pose Tracking on Mobile Devices," *Proceedings of the 12th Computer Vision Winter Workshop*, 2007.

[158] Wang, C. and Thorpe, C., "Simultaneous localization and mapping with detection and tracking of moving objects," *International Conference on Robotics and Automation*, 2002.

[159] Wang, C., Thorpe, C., and Thrun, S., "Online simultaneous localization and mapping with detection and tracking of moving objects: Theory and results from a ground vehicle in crowded urban areas," *International Conference on Robotics and Automation*, 2003.

[160] Weingarten, J. and Siegwart, R., "3D SLAM using planar segments," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 3062–3067, IEEE, 2006.

[161] Wilson, A. D. and Bobick, A. F., "Parametric hidden markov models for gesture recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, no. 9, pp. 884–900, 1999.

[162] Wolf, D. and Sukhatme, G. S., "Towards mapping dynamic environments," *International Conference on Advanced Robotics*, 2003.

[163] Wolf, D. and Sukhatme, G. S., "Online simultaneous localization and mapping in dynamic environments," *International Conference on Robotics and Automation*, 2004.

[164] Wolf, D. and Sukhatme, G. S., "Mobile robot simultaneous localization and mapping in dynamic environments," *2005*, Autonomous Robots.

[165] Wu, C., "SiftGPU: a GPU implementation of scale invariant feature transform (SIFT)." [Online] Available: `http://cs.unc.edu/~ccwu/siftgpu/`, 2007.

[166] Zender, H., Jensfelt, P., Mozos, Ó., Kruijff, G., and Burgard, W., "An integrated robotic system for spatial understanding and situated interaction in indoor environments," in *Proceedings of the National Conference on*

Artificial Intelligence, vol. 22, p. 1584, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[167] Zukerman, I., Kowadlo, G., and Ye, P., "Interpreting pointing gestures and spoken requests: a probabilistic, salience-based approach," in *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pp. 1558–1566, Association for Computational Linguistics, 2010.

[168] Zukerman, I., Mani, A., Li, Z., and Jarvi, R., "Speaking and pointing—from simulations to the laborator," *Knowledge and Reasoning in Practical Dialogue Systems*, p. 58, 2011.