Prioritized Text Spotting using SLAM

by

Yafim Landa

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2013

© Massachusetts Institute of Technology 2013. All rights reserved.

Author . . . .
Department of Electrical Engineering and Computer Science
August 23, 2013

Certified by . . . .
Seth Teller
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
Dennis M. Freeman
Professor of Electrical Engineering
Chairman, Masters of Engineering Thesis Committee

Prioritized Text Spotting using SLAM

by

Yafim Landa

Submitted to the Department of Electrical Engineering and Computer Science
on August 23, 2013, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We show how to exploit temporal and spatial coherence of image observations to achieve efficient and effective text detection and decoding for a sensor suite moving through an environment rich in text at a variety of scales and orientations with respect to the observer. We use simultaneous localization and mapping (SLAM) to isolate planar "tiles" representing scene surfaces and prioritize each tile according to its distance and obliquity with respect to the sensor, and how recently (if ever) and at what scale the tile has been inspected for text. We can also incorporate prior expectations about the spatial locus and scale at which text occurs in the world, for example more often on vertical surfaces than non-vertical surfaces, and more often at shoulder height than at knee height. Finally, we can use SLAM-produced information about scene surfaces (e.g. standoff, orientation) and egomotion (e.g. yaw rate) to focus the system's text extraction efforts where they are likely to produce usable text rather than garbage. The technique enables text detection and decoding to run effectively at frame rate on the sensor's full surround, even though the the CPU resources typically available on a mobile platform (robot, wearable or handheld device) are not sufficient to such methods on full images at sensor rates. Moreover, organizing detected text in a locally stable 3D frame enables combination of multiple noisy text observations into a single higher-confidence estimate of environmental text.

Thesis Supervisor: Seth Teller
Title: Professor of Computer Science and Engineering

Acknowledgments

Thank you, Professor Seth Teller, for giving me the chance to work on such an exciting project and for your encouragement, advice, and support throughout my M.Eng. Thanks to the Andrea Bocelli Foundation for their support. Thanks to Professor Rob Miller for letting me work on past projects and for introducing me to this project. Thanks to Nick Wang for being a great and fun person to work with and for your valuable contributions to the project. Thanks to Maurice Fallon, Javier Velez, and Jon Brookshire for getting the project going and for your continued support along the way. Thanks to Brian Wu and Pallavi Powale for listening to my ideas and providing feedback, and keeping me company in the lab. Thanks to William Souillard-Mandar and Lulu Yang for their friendship and moral support. Thanks to Victor Costan for being a great friend, for teaching me how to navigate the world, and for seemingly being able to solve any problem (like Mr. Wolfe from Pulp Fiction). Thanks to Vazrik Chiloyan for similar reasons. Thanks to my teachers from Bergen Tech, especially Mr. Goodman, for introducing me to science and engineering. Thanks to my parents, Slava and Rita Landa, for their unconditional love and support.

Portions of Chapters 1, 6, and 8 appeared in H. C. Wang, Y. Landa, M. Fallon, and S. Teller. Spatially prioritized and persistent text detection and decoding. *Camera-based Document Analysis and Recognition (CBDAR)*, 2013. [35].

Contents

1 Introduction **15**

1.1 Problem Description . . . 17

1.2 Motivation and Challenges . . . 18

1.3 Prior Work . . . 20

1.4 Thesis Outline . . . 21

2 Proposed Method **23**

2.1 Overview . . . 24

2.2 Major Stages . . . 26

2.2.1 Sensing and SLAM . . . 26

2.2.2 Tiling and Warping . . . 26

2.2.3 Prioritizing . . . 28

2.2.4 Text Spotting . . . 29

2.3 System Design and Implementation Notes . . . 29

3 Sensors and Hardware Prototypes **33**

3.1 Sensors . . . 33

3.1.1 LIDAR . . . 33

3.1.2 Cameras . . . 34

3.1.3 IMU . . . 35

3.2 Sensor Data Preprocessing . . . 35

3.2.1 Camera Image Preprocessing . . . 35

3.2.2 Sensor Fusion . . . 36

3.3 Hardware Prototypes 37
3.3.1 Hand-Carried Rig 37
3.3.2 Wearable Rig 38

4 Scan Matcher, Tiler, and Warper **41**
4.1 Scan Matcher 41
4.2 Wall Tiler 42
4.2.1 Extracting and Merging Line Segments 42
4.2.2 Tile Creation and Clustering 43
4.3 Warper 45
4.3.1 OpenCV Warper 46
4.3.2 OpenGL Warper 48

5 Prioritizer **51**
5.1 Overview 51
5.2 Scoring 51
5.2.1 Distance 52
5.2.2 Obliquity 52
5.2.3 Speeds 53
5.2.4 Spatial Prior 53
5.2.5 Inspection State 53
5.2.6 Time Decay 54

6 Text Spotter **55**
6.1 Ground Truth 55
6.2 Text Detector 57
6.3 Text Decoder 58
6.4 Character and Word Clustering 58
6.5 Finding Optimal Word Configuration 59

7 Output **61**
7.1 3D Viewer 61

7.2 2D Viewer 62
7.3 Data Output 63

8 Experiments and Results **65**
8.1 Experiments 66
8.1.1 Stata Third Floor: Initial Data Collection 66
8.1.2 Stata Second Floor: More Realistic Experiment 68
8.2 Evaluation Experiment and Results 69
8.2.1 Warping Accuracy with Distance and Obliquity 69
8.2.2 Alignment of Warped Observations 71
8.2.3 Performance with Multiple Observations 72

9 Future Work **75**

A Installation Instructions **79**
A.1 Requirements 79
A.1.1 Aptitude Packages 79
A.1.2 LCM 80
A.1.3 Enable Large UDP Packets 80
A.1.4 libbot 80
A.2 Check Out Bocelli Text-Loco 80
A.3 Running the System 81

B LCM Types **83**

C Coordinate Systems **89**

List of Figures

1-1 Example of environmental text that provides navigation information (left), lab directions (middle) and information about current research at MIT (right). 15

2-1 Example of a map and pose estimate pair generated by the scan-matcher. The black lines are walls, the red disk is the position, and the red lines show the boundaries of the field of view. 27

2-2 Left: Wall tiler running on walls that were detected by the SLAM component. The red dots represent tile origins and the green line segments represent the tile normal vectors. Right: The same tiles, overlaid on the scene image. 28

2-3 An example of tile rankings produced by the prioritizer. The numbers within the tiles show the prioritizer ranking (where 1 is the highest-priority tile). The prioritizer has chosen the three closest, un-inspected tiles as the highest-priority tiles and a tile that was viewed head-on (but from far away) as the next most-important tile. Please note: the tile border colors in the bottom-right corner only indicate distance, not priority. 29

2-4 Final text spotting results from a tile in one of our experiments. . . . 30

2-5 System diagram. Red ovals indicate sensors, blue rectangles represent system components that process data, and green rectangles represent system components that behave as outputs. 31

3-1 The Hokuyo planar LIDAR (left) and an example of the data it produces (right). The data shows the ranges detected by the laser as yellow points. 33

3-2 The Dragonfly2 monocular camera. 34

3-3 The Bumblebee2 stereo camera. 34

3-4 The MicroStrain IMU. 35

3-5 Image rectification for the left Bumblebee2 camera. Notice how the barrel distortion is removed by looking at the curvature in the pipes near the ceiling. 35

3-6 This screen shot was taken when the LIDAR and the camera data were paired at different times when the yaw rate of the sensor suite was high. Notice that the warping of the scene image to each tile is incorrect. . 37

3-7 The first, hand-carried, rig with mounted Hokuyo, Bumblebee2, and IMU sensors. The rig also carries batteries, and there's a laptop in the background for data collection. 38

3-8 The second, wearable, rig with mounted Hokuyo and Dragonfly2 sensors. An IMU was later added where the word 'IMU*' is written on the Figure. There would also be a small laptop to record data or run the system, perhaps in a backpack. 39

4-1 A sample output of the scan-matcher. The black lines represent the walls, the red disk represents the position of the sensor suite, and the red lines represent the field of view. 41

4-2 Line segments extracted from the scan-matcher output using the probabilistic Hough transform. The black lines are the walls from the scan-matcher and the blue lines are the output from the probabilistic Hough transform algorithm. 43

4-3 Tiles created from the extracted line segments. The red dots indicate the tile origins and the green lines indicate the normals. 44

4-4 Tiles similar to the ones from Figure 4-3, viewed in the 3D viewer from above. 44

4-5 Two layers of tiles, as seen in both the 3D viewer and as tile outlines overlaid on the scene image. The bottom layer of tiles is mostly outside of the camera field of view. 45

4-6 The homography that was computed to warp the scene image onto the wall. 46

4-7 The result of applying the computed homography. 47

4-8 The tiles that are in the FOV are created as OpenGL QUADs and projective texture mapping is used to project the scene image onto them. 48

4-9 A virtual camera is configured so that each tile fits exactly in the FOV from a meter away. This camera is then placed one meter in front of each tile, and a tile image is captured. 49

5-1 The score as a function of the distance from the camera to the tile, in meters. 52

6-1 A screen shot of the LabelMe interface that was used to generate ground truth labels. 56

6-2 Ground truth bounding boxes, persisted on their respective tiles. Only the bounding boxes, and not the associated values, are shown in this screen shot. 57

7-1 3D viewer (no data). 62

7-2 Two examples of the 2D viewer output. 62

8-1 The first experiment setting and a close-up of one of the signs. 66

8-2 The second experiment setting and as seen through the 3D viewer. . . 68

8-3 Experiment settings and accuracy comparison of original and warped observations. (a) The normal of the surface is roughly antiparallel to the viewing direction. (b) The normal of the surface is about 45 degrees away from the viewing direction. Plots (c) and (d) show the accuracy of baseline decoding of original (O) and warped (W) tiles with respect to viewing distance for observations (a) and (b). (e) An original tile observation from 0.71 meters. (f) The warped observation corresponding to (e). The accuracy scores of (e) and (f) are 0.67 and 0.96, respectively. 70
8-4 The distribution of decoded characters. (a) There were only slight vertical and horizontal shifts. (b) Comparison between data with and without IMU for the second dataset (hand-carried). There were longer vertical drifts without IMU, but use of the IMU reduces drift. 71
8-5 Accuracy comparison with respect to viewing distance for observations. 72

9-1 The MultiSense SL sensor. The image was taken from the DRC website. 75
9-2 An example of a point cloud generated by the rotating Multisense laser. 76
9-3 Planar tiler running on a plane segment found in the environment (in this case, a piece of a wall). We first see the plane segment in (a). The plane segment is tiled in (b) and later expands in (c) as more of it is visible. The tiles expand to cover the newly-discovered parts of the plane in (d). 77
9-4 Planar tiler running on a plane segment found in the environment through the use of the MultiSense SL laser. 78
9-5 An example of a point cloud generated by the Multisense stereo pair. 78

C-1 The laser (left) and camera (right) coordinate frames. The sensor suite is pointing at a tile. 90

Chapter 1

Introduction

The world around us is full of text that provides us with important information. Examples include street signs and house numbers, office labels and building directories, documents, road signs and warnings, posters, and many others. These texts tell us where we are and where we might possibly want to go, inform us about the people and objects around us, warn us about potential hazards, and in general, provide a lot of information about our environment.

Figure 1-1: Example of environmental text that provides navigation information (left), lab directions (middle) and information about current research at MIT (right).

While it is easy for sighted individuals to absorb this information from environmental text, it is impossible (or is at least much more difficult) for the visually-impaired to extract this information. One way to harvest this information is to ask another person for help, a method that although effective, reduces the user's independence from others and is not always possible or practical. Another way to harvest textual information is to use technology, but unfortunately, current text recognition technol-

ogy is not designed for use in the typical unconstrained environments that a user is likely to experience.

The goal of this thesis is to improve the latter approach. We wish to develop machine perception systems, for use by mobile robots or by people, that can efficiently and effectively detect and decode text from sensor observations of the surroundings.

In this thesis, I will describe an end-to-end system that localizes text within a scene image and decodes the visible characters. We wanted to create a wearable or hand-held sensor suite that will allow a blind or visually-impaired user to read environmental text in real-time. My colleague Nick Wang and I (under the guidance of Prof. Teller and with help from others in the RVSN group) designed and built several prototypes and produced some noteworthy results. I will describe the motivation for working on this problem and some existing approaches; the overall design of the system, the hardware used, and the software implementation; the experiments that we performed; and end with a summary of the results obtained thus far.

1.1 Problem Description

End-to-end text spotting aims to localize text within a scene image and decode the visible characters. A scene image can be captured from any angle and any position in the environment, and will typically come in the form of a stream of images that surveys the environment from a chest height. Because camera-captured images do not capture enough information to detect and decode text, we will also capture depth data from a laser sensor, and also orientation data from an IMU. The problems that we face are (a) how to capture this data and what other data do we need (b) how do we design and implement the infrastructure needed to process this data (c) what kinds of components do we need to achieve our goal of identifying text in the environment and how do we implement them. All of these problems are addressed in this thesis.

1.2 Motivation and Challenges

Our goal in designing this system is to make a portable device that will let a person or a robot read environmental text. As mentioned previously, this kind of system would expose its user to a wealth of information that is typically available to sighed users, but is hard to access for our target users. In addition, the same kind of information could be very useful for robots (e.g. autonomous cars that can read road signs to learn about road condition warnings). To achieve this goal, our device must meet a few requirements.

First, we wish to extract text quickly enough to support real-time uses such as navigation (e.g., the user seeks a numbered room in an office or hotel), shopping (e.g., the user seeks a particular aisle, product, or price label in a store), gallery visits (e.g. the user wants notification and decoding of labels positioned on the walls and floors, or overhead), or the autonomous car example given above. The system must also be accurate and reliable so that the extracted information is actually useful and not misleading.

Second, because we desire this system to be carried around or worn by a person, it must satisfy certain physical requirements. For example, it must not be too heavy to wear for extended periods of time every day. It must also not be too big and bulky, or must use a form-factor lends itself to being wearable. Lastly, it would be desirable if it didn't draw too much attention to itself or its wearer (see [10] for more information).

Third, it would be desirable if this system were not too expensive. We decided not to focus on cost too much at this point of development, however.

We argue that the particular demands of this task imply that processing should be performed on-board (i.e., by hardware local to the user), rather than in the cloud, and in a way that exploits spatiotemporal coherence (i.e. the similarity of the text available now to the text available in the recent past). Because the user often needs a response in real time, we can rule out the use of intermittent or high-latency network connections. In addition, the task involves large amounts of data arising from obser-

vations of the user's entire field of view at a resolution sufficient for text detection. This rules out the use of a low-bandwidth network connection. Moreover, in 2013 one cannot analyze a full field of view of high-resolution pixels in real-time using hardware that would be reasonable to carry on one's body (say, a quad- or eight-core laptop). We investigated what useful version of the problem could be solved with wearable hardware, and designed the system to inspect, and extract text from, only those portions of the surroundings that we deem as high priority.

Unlike scene text in images observed by a stationary camera, text observed by a moving camera will generally be subject to motion blur or limited depth of field (i.e. lack of focus). Blurry and/or low-contrast images make it challenging to detect and decode text. In addition, environmental text comes in a variety of font styles, sizes, orientations, and languages. Textures that look like text are also typically present in environmental text, which can generate false-positives. Neither increasing sensor resolution, nor increasing CPU bandwidth, are likely to enable text detection alone; instead, improved methods are required.

1.3 Prior Work

The problems of end-to-end word spotting has been previously explored. Typically, text detection is performed to localize where text may appear; detected regions are then processed to extract text using an Optical Character Recognition (OCR) engine. Chen and Yuille [4] trained a strong classifier using AdaBoost to determine text regions, and used commercial OCR software for text decoding.

Neumann and Matas [23, 24, 25] used Maximally Stable Extremal Region (MSER) [19] detection and trained a classifier to separate characters from non-characters using several shape-based features, including aspect ratio, compactness, and convex hull ratio. They reported an average run time of 0.3 s on an 800×600 image, and achieved recall of 64.7% in ICDAR 2011 dataset [18] and 32.9% in SVT dataset [37].

Wang and colleagues [37, 36] described a character detector using Histograms of Oriented Gradient (HOG) features or Random Ferns, which given a word lexicon can obtain an optimal word configuration. They reported computation times of 15 seconds on average to process an 800×1200 image. Their lexicon driven method — combining the ABBYY FineReader OCR engine and a state-of-the-art text detection algorithm (Stroke Width Transform (SWT) [6]) — outperformed the method using ABBYY alone.

The open-source OCR engine Tesseract [32, 33] has some appealing features, such as line finding, baseline fitting, joined character chopping, and broken character association. Although its accuracy was not as high as that of some other commercial OCR engines [37], it has been widely used in many studies.

1.4 Thesis Outline

The rest of this thesis is structured as follows. This chapter explained the problem, why it's important, and what others have already done to address it (or parts of it). Chapter 2 explains the design of the system and its high-level components. Chapter 3 gives an overview of the sensors, how the sensor data is collected and pre-processed, and presents the hardware prototypes that were used in the project. Chapter 4 delves deeper into the design and implementation of the scan-matcher, tiler, and warper components. Chapter 5 delves deeper into the design and implementation of the prioritizer. Chapter 6 explains how the text spotter component works, including text detection, text decoding, and the clustering and language model methods that produce the final text spotting results. Chapter 7 explains how the output is presented to the user and how data is exported for further research and development. Chapter 8 talks about the experiments that were performed during and after the development of the system and the results that were obtained. Finally, Chapter 9 discusses some options for the future development of the system, both long-term and short-term.

Chapter 2

Proposed Method

This chapter explains our proposed method of solving the problem described in Chapter 1. It begins with a general overview of the entire approach and then provides an overview of each of the main stages of the approach. Lastly, an overview of the system design is provided, as well as some general implementation notes.

2.1 Overview

As mentioned in Section 1.3, a number of challenges prevent us from scanning every area of every image captured by a moving camera. In order to make text spotting more practical, we take advantage of two observations.

First, we observe that not all pixels in a scene image are equally likely to belong to text. For example, in an office environment, text is unlikely to appear within the pixels that belong to the ceiling or to the floor. Therefore, we can analyze the environment and prioritize our text spotting process on the areas that are more likely to have text. The areas where text is likely to appear depend, of course, on the environment: text is likely to appear at shoulder-height in an office environment, but is more likely to appear overhead in an airport.

To be able to do this, however, we need to have more information about the geometry of the scene, as it is hard to segment a simple two-dimensional image into such regions. I will explain how we acquire this information shortly.

In addition, if an area of the image is far away, poorly lit, or is captured at a bad angle, we can wait to examine that area until we've gotten a better look. We need the aforementioned extra information to do this, as well.

Second, we observe that text is typically persistent. That is, unless the text is (for example) printed on a T-shirt or on a truck, it typically doesn't move and can be observed multiple times. This means that if we can match two regions to one another and determine that they're images of the same surface in the world, we can get multiple looks and refine our text detection and decoding results as we get more (and hopefully, slightly different) looks, with each providing slightly more information.

Using these two observations, we designed our system to use the following process. We first constructed a sensor suite that we hand-carry through the world. We then use a scan-matching Simultaneous Localization and Mapping (SLAM) component developed within the group to build a map and maintain a pose estimate for the sensor suite. We then break up the walls that we acquired from the SLAM component into 'tiles' that are persisted in our systems.

Next, with every new RGB camera frame received, we take the following steps. First, we project the new camera information onto all of tiles that are in the camera's field of view. This projection results in one new fronto-parallel image for each visible tile. This image, called the 'tile image' is added to the existing stack of tile images for the tile. We then order all of the tiles that are known to the system based on a priority score and select some of the highest-scoring tiles. The tiles are scored based on their closest observation distance and best observation angle, their location within the world, the number of times that we've observed them, how long ago we last observed them, and some other factors (the scoring function is discussed in detail in Section 5).

The tiles that were selected are sent on to the text spotter components. The text spotter's job is to determine what (if any) is written on the tile. First, a text detection algorithm based on Discrete Cosine Transform (DCT) and Maximally Stable Extremal Regions (MSER) is run on each observation. This algorithm generates a set of character bounding boxes for each tile. The Tesseract OCR engine [32, 33] is then executed on the areas of the tile produced by the text detector, and character candidates are collected, along with their confidences. We then group these candidates into characters (to account for the slight shifts introduced by the warping process), and finally, into words based on a language model. In the end, we get a set of words positioned on each processed tile.

To display the results, we use a 3D viewer created in the RVSN group to show all of the tiles and their text spotting results. The tiles are shown by drawing a composite image — an image that is created by combining all of the observations for the tile — in the tile's 3D location. The text spotting results are shown as rectangles drawn on top of the composites, with the decoded words written above the bounding box rectangle.

As of now, we don't do anything after settling on a text spotting result. Although the project is designed to convey environment text to a blind user, the method in which we convey the spotted text is not the focus of this thesis.

The next few sections will discuss each of the major stages of the approach in

more detail.

2.2 Major Stages

2.2.1 Sensing and SLAM

Our goal in sensing the environment is to get a description of the environment and to get our pose within it. As a simplifying assumption, we settled on having just a description of the *walls* around us and our (X, Y, Yaw) coordinates. We also considered using a full 3D description of the environment, along with an XYZRPY pose, but this is not the focus of the thesis (although it is discussed in Chapter 9).

A camera/laser combination uses a synchronized laser and camera pair to get the information that we require. The laser data is passed to a scan-matching SLAM component developed in the RVSN group. This component gives us a map of the environment (consisting of walls) and a pose estimate (X, Y, Yaw) within it. Please see Figure 2-1 for an example of this data. Using the camera/laser combination has the advantage of being very stable, but more limiting in the range of motion allowed and sensor suite cost and bulk. The scan-matcher component assumes that variations in roll and pitch are minimal (under 10-20 degrees), so we have to keep these to a minimum when using the rig, and we need to use an IMU to keep the camera and laser in sync.

2.2.2 Tiling and Warping

We take the walls (represented by the black line segments in Figure 2-1) and divide them into meter-wide tiles. Please see Figure 2-2. The background on the left is the 2D description of the walls from Figure 2-1 — the output of the scan-matcher — and the yellow lines represent the walls that we tiled. The tiles themselves are represented as red dots that indicate the tile origins and green line segments that indicate the tile normal vectors.

After getting a map and pose estimate, we project the camera image onto the

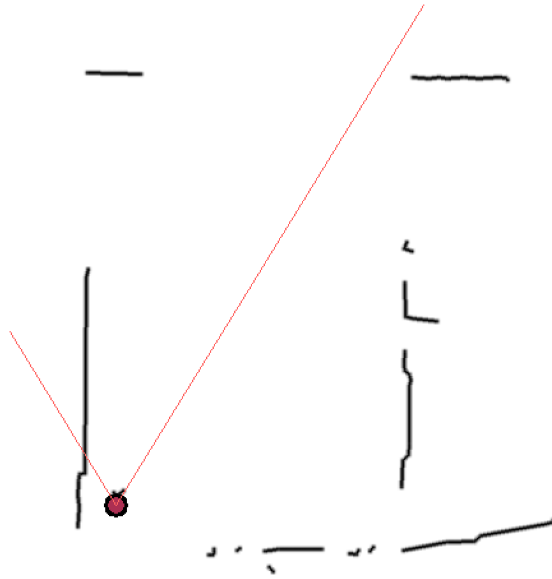


Figure 2-1: Example of a map and pose estimate pair generated by the scan-matcher. The black lines are walls, the red disk is the position, and the red lines show the boundaries of the field of view.

tiles from our current pose. We begin by extending the 2D tile line segments into 3D tile plane segments by giving them an arbitrary height, resulting in a set of tiles that are represented by four 3D corners. The 3D tiles can be seen as an overlay on a scene image on the right side of Figure 2-2. In this case, we created two layers of tiles, stacked on top of one another, where each tile is one meter tall. Then, we use projective texture mapping to project the camera image onto these tiles. Imagine a projector that has the same pose as the sensor suite (adjusted for the height at which the sensor suite was carried and for the roll and pitch reported by the IMU) that is projecting the camera-captured image onto the 3D wall planes. This allows us to match the pixels from the image with the 3D model, giving us the regions within the image that correspond to the walls that we detected. In addition, we can save the pixels that belong to each tile separately, creating a fronto-parallel view of each tile.

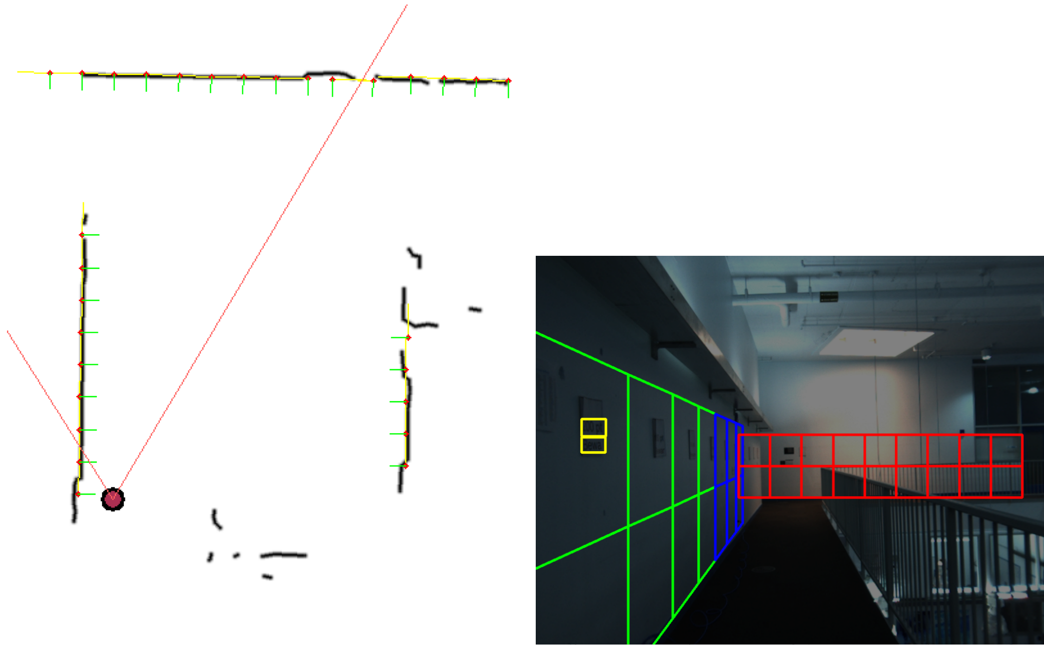


Figure 2-2: Left: Wall tiler running on walls that were detected by the SLAM component. The red dots represent tile origins and the green line segments represent the tile normal vectors. Right: The same tiles, overlaid on the scene image.

2.2.3 Prioritizing

Now that we have tile corners, along with fronto-parallel images for each tile, we can choose which tiles are the most important for text spotting. The factors that we consider include the distance from the sensor suite to the tile, the angle from which the tile was viewed (head-on or nearly-parallel), the sensor suite translational and rotational velocities (fast motion creates a blurrier image, so we'd want to de-prioritize that observation), and how long ago we've last seen the tile. For example, the priorities for the tiles from Figure 2-2 are shown in Figure 2-3. Please note that there is a processed tile in this figure, as evidenced by the yellow text bounding boxes and green indicator above the tile labeled '18'. In this situation, the prioritizer decided that the most important tiles are the closest un-inspected head-height tiles, followed by two distant but head-on, head-height tiles at the end of the hallway, followed by closer, knee-height tiles. The exact priority function is described in Section 5.

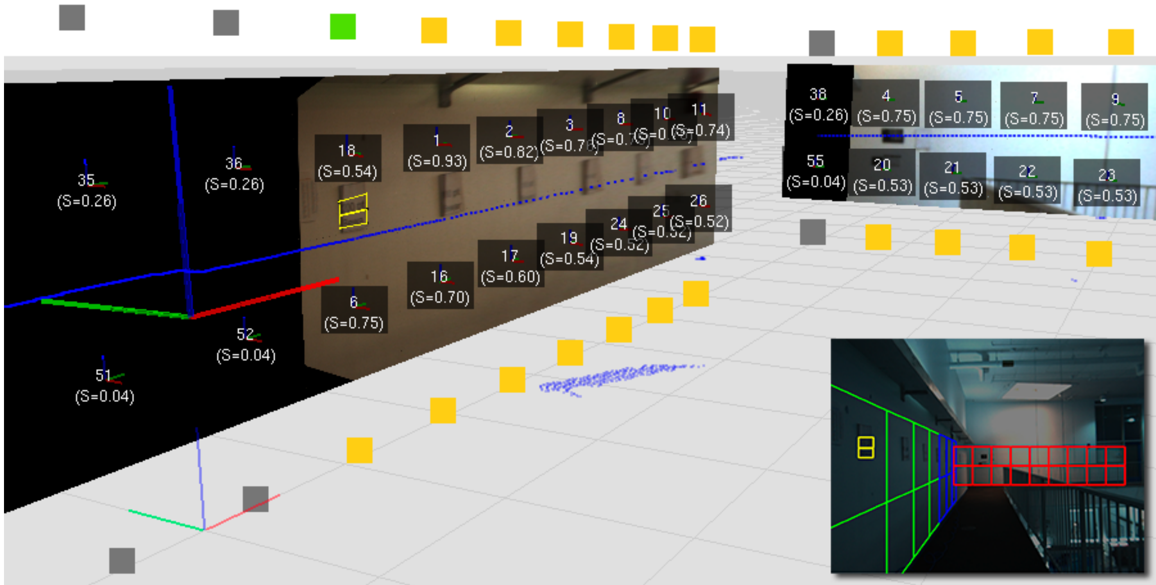


Figure 2-3: An example of tile rankings produced by the prioritizer. The numbers within the tiles show the prioritizer ranking (where 1 is the highest-priority tile). The prioritizer has chosen the three closest, un-inspected tiles as the highest-priority tiles and a tile that was viewed head-on (but from far away) as the next most-important tile. Please note: the tile border colors in the bottom-right corner only indicate distance, not priority.

2.2.4 Text Spotting

The last step in the process is text detection and text decoding. The text detection and decoding component receives a set of high-priority tiles from the prioritizer. Each of these tiles will contain a set of fronto-parallel tile images, all of which are used by the text detector and decoder to extract the text on the tile. First the detector looks for areas in the image that contain text, and then the decoder tries to find the actual letters in these areas. Then, a clustering and voting process is used to return the words on the tile. This process is described in more detail in Section 6. The final results can be seen in Figure 2-4.

2.3 System Design and Implementation Notes

The system is designed as a collection of modular components that communicate through a system called Lightweight Communications and Marshalling (LCM) [12].

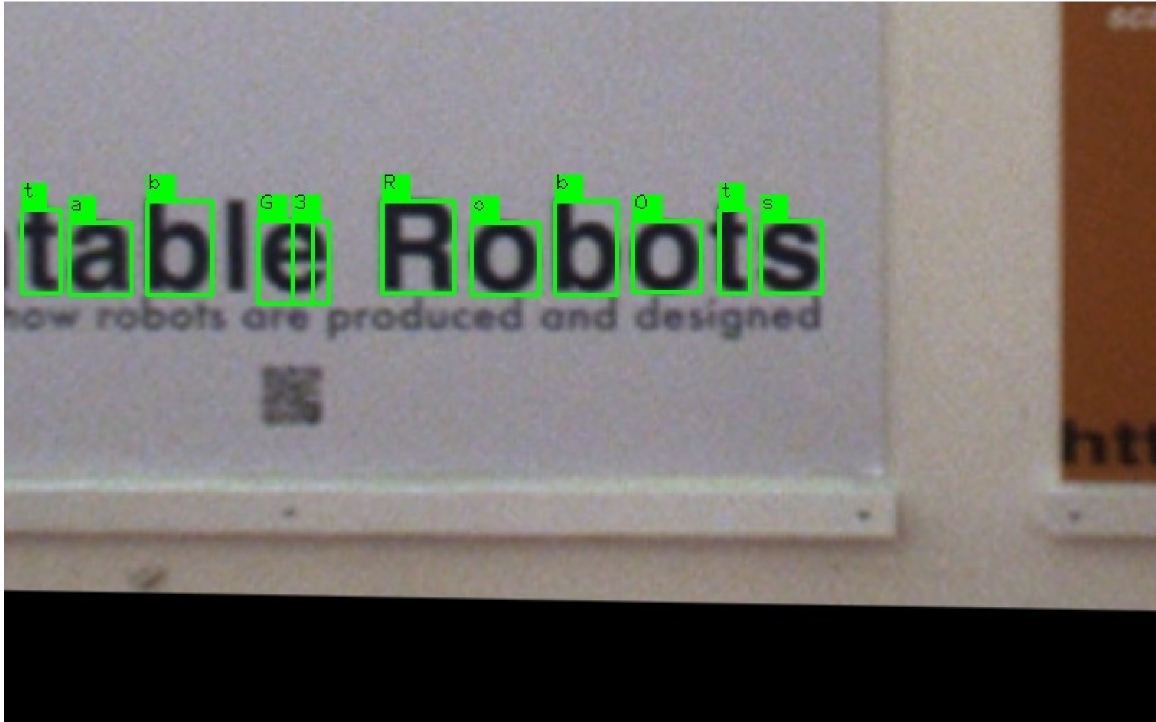


Figure 2-4: Final text spotting results from a tile in one of our experiments.

LCM allows us to describe a set of data types. These data types are converted to code in C, C++, Python, and MATLAB, and allows us to send data between processes on the same machine, processes on different machines, and programs written in any of the LCM-supported languages. The LCM types that we used will be described in this section, but listings are also available in Appendix B. Since all of our code was written in C and C++, we did not really need this functionality, but it was nice to have in case we ever wanted to test some text detection algorithms in MATLAB, for example.

Please look at the system diagram in Figure 2-5, which outlines the different system components and their data interconnections. Directed arrows between two components represent data flow, and the labels on them describe the kind of data that is transmitted. The red ovals represent sensors, blue rectangles represent system components, and green rectangles represent output components that produce data for us to see (either a visualization or a data dump).

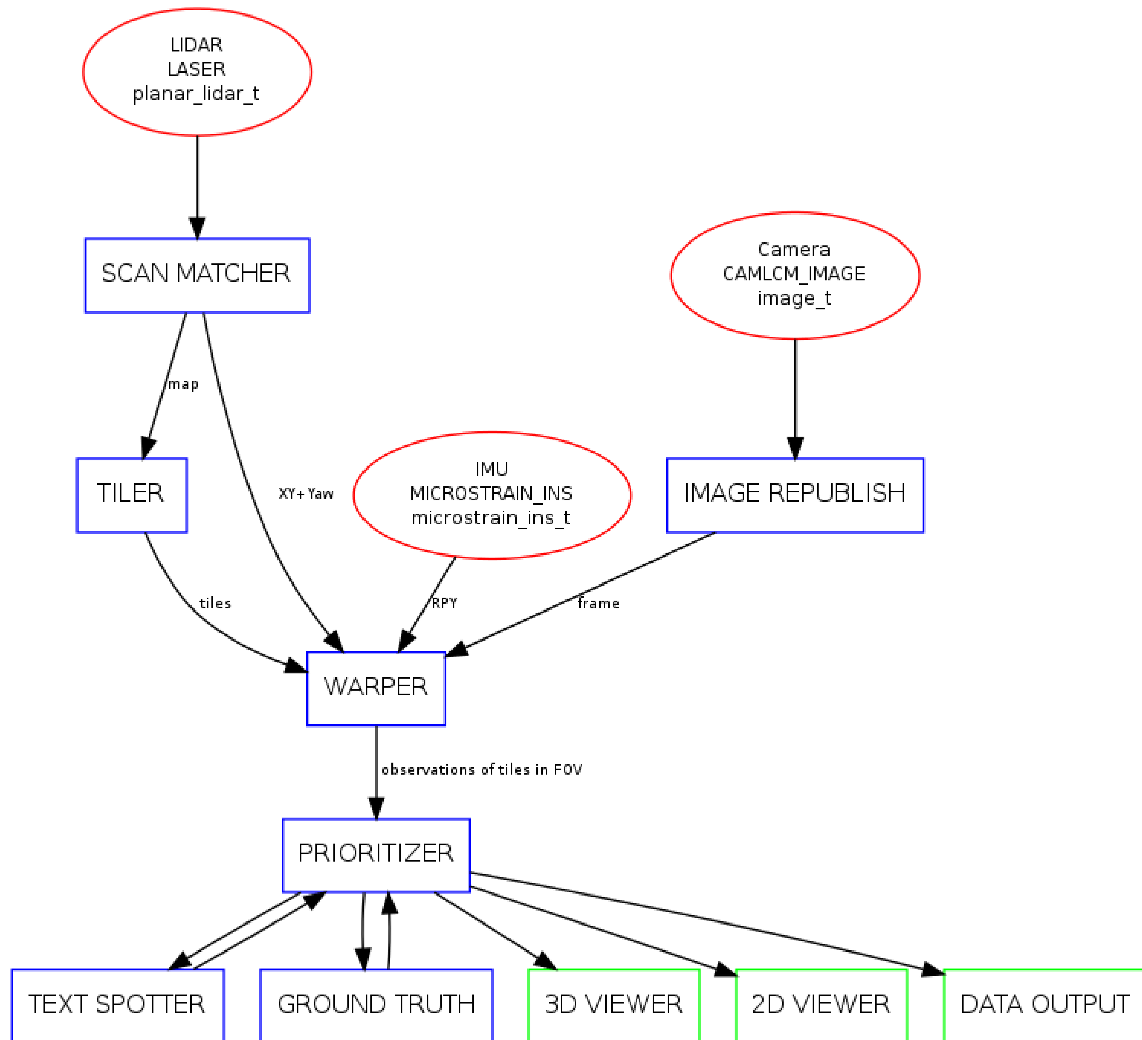


Figure 2-5: System diagram. Red ovals indicate sensors, blue rectangles represent system components that process data, and green rectangles represent system components that behave as outputs.

Chapter 3

Sensors and Hardware Prototypes

3.1 Sensors

We use several sensors in our system. This section describes each sensor in detail, including the model, sampling rate, cost, and physical dimensions.

3.1.1 LIDAR

We use a Hokuyo UTM-30LX planar LIDAR as the input for the scan-matching module. The Hokuyo laser has a detection range of 270° and is accurate between 0.1 meters to 30 meters, and its angular resolution is 0.25°. The Hokuyo laser costs around $5,600, and it weighs about 370 grams. Its size is 60 by 60 by 87 millimeters, and it is shown in Figure 3-1, alongside a visualization of some sample data from our experiments. The data is collected through USB at 40 Hz.

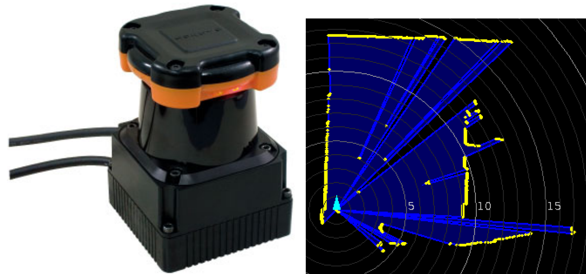


Figure 3-1: The Hokuyo planar LIDAR (left) and an example of the data it produces (right). The data shows the ranges detected by the laser as yellow points.

3.1.2 Cameras

We had two cameras: a Bumblebee2 stereo camera and a Dragonfly2 monocular camera, both made by Point Grey. Although the Bumblebee2 is a stereo camera, we only used the left image from the stereo pair, and chose to use it mainly because we had a really good calibration for it already from the group. The Dragonfly2 camera was later added because we wanted higher-quality images, which the Dragonfly2 can provide.

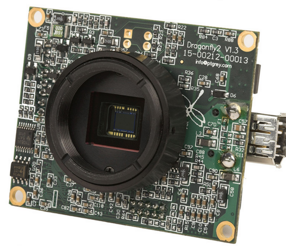


Figure 3-2: The Dragonfly2 monocular camera.

The Bumblebee2 camera is shown in Figure 3-3. It produces two 1024 by 768 images at 15 Hz with a horizontal field of view of 65 degrees, and has a global shutter. The Dragonfly2 camera produced a single 1296 by 964 image at 20 Hz, and the field of view could be controlled with an external lens. The Bumblebee2 camera is bulkier than the Dragonfly2 at 157 by 36 by 47.4 mm and 342 grams compared to the Dragonfly2's 44 mm by 34 mm by 24.4 mm and 45 grams. However, we also used a lens with our Dragonfly2 camera, which added a few grams and made the package slightly bigger. Even with the lens, the Dragonfly2 was lighter and smaller, and hence more portable, than the Bumblebee2.

Figure 3-3: The Bumblebee2 stereo camera.

3.1.3 IMU

The IMU we used was the MicroStrain 3DM-GX3-25. It costs $1,895 and provides roll, pitch, and yaw data (or a quaternion) at up to 1,000 Hz (we used it at 100 Hz). It has built-in filters that eliminate long-term drift. The MicroStrain is very light and small, measuring just 44 mm by 24 mm by 11 mm and weighing only 18 grams.

Figure 3-4: The MicroStrain IMU.

3.2 Sensor Data Preprocessing

3.2.1 Camera Image Preprocessing

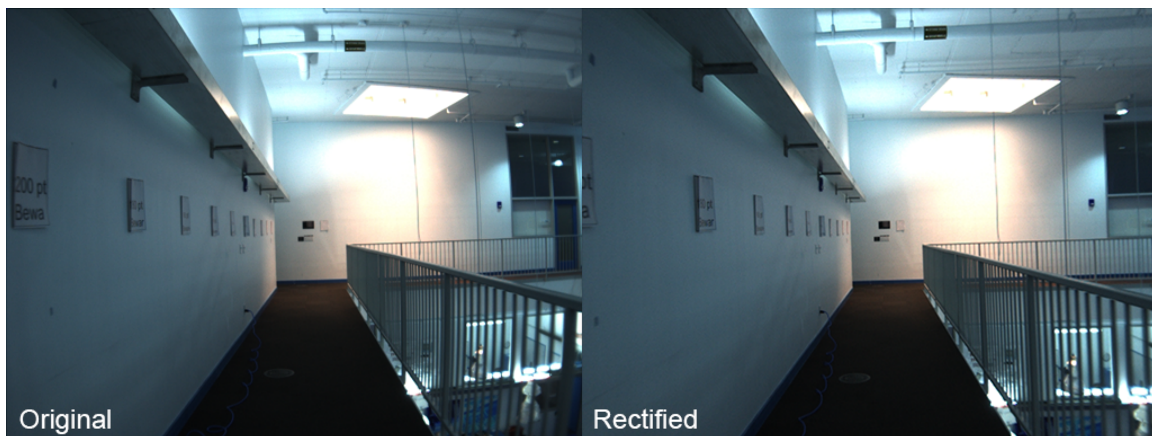


Figure 3-5: Image rectification for the left Bumblebee2 camera. Notice how the barrel distortion is removed by looking at the curvature in the pipes near the ceiling.

Our data did not require much preprocessing. The only preprocessing that we had to do was to get rid of the right image in the stereo image pair and remove the

barrel distortion introduced by the lens. The Bumblebee2 camera driver produces two vertically-stacked images as an output, where the left image is on top and the right image is on the bottom. To retain only the left image, we crop the top 768 pixels. We use an existing calibration file that was created in the group to remove the barrel lens distortion, as can be seen most easily in the overhead pipes in Figure 3-5.

3.2.2 Sensor Fusion

It is important to make sure that all of the sensors are synchronized in time. Each sensor is sampling data at a different rate, and in order to warp the image correctly, we need to make sure that the pose estimate reported by the scan-matcher is accurate for the time at which the camera image was captured. In order to make sure that this is the case, we kept a circular buffer of all of the (X, Y, Yaw) triplets generated by the scan-matcher, along with the timestamp of the laser scan for which this triplet was generated. When a camera image is received, we search through the circular buffer to find the pose that has the closest matching timestamp to the timestamp of the image. In addition, we need to make sure that we have an accurate measurement of the distances between the various sensors on our rigs, so that we can incorporate them into our calculations. For example, even though the scan-matcher produces a pose estimate, the camera is shifted by 13 centimeters to the side in relation to the laser, so our warper needs to set the camera pose to be offset by the same amount.

All of this can also be done with a tool called bot-frames, although we decided to implement our own solution for this project. Eventually, this code should be changed to use bot-frames, and it's been separated for the purpose in the most recent version of the files.

Figure 3-6 shows what two successive camera images look like when projected onto the wall in the case that our sensor timestamps are not synchronized and the yaw rate is high. Notice that the warping is quite off.

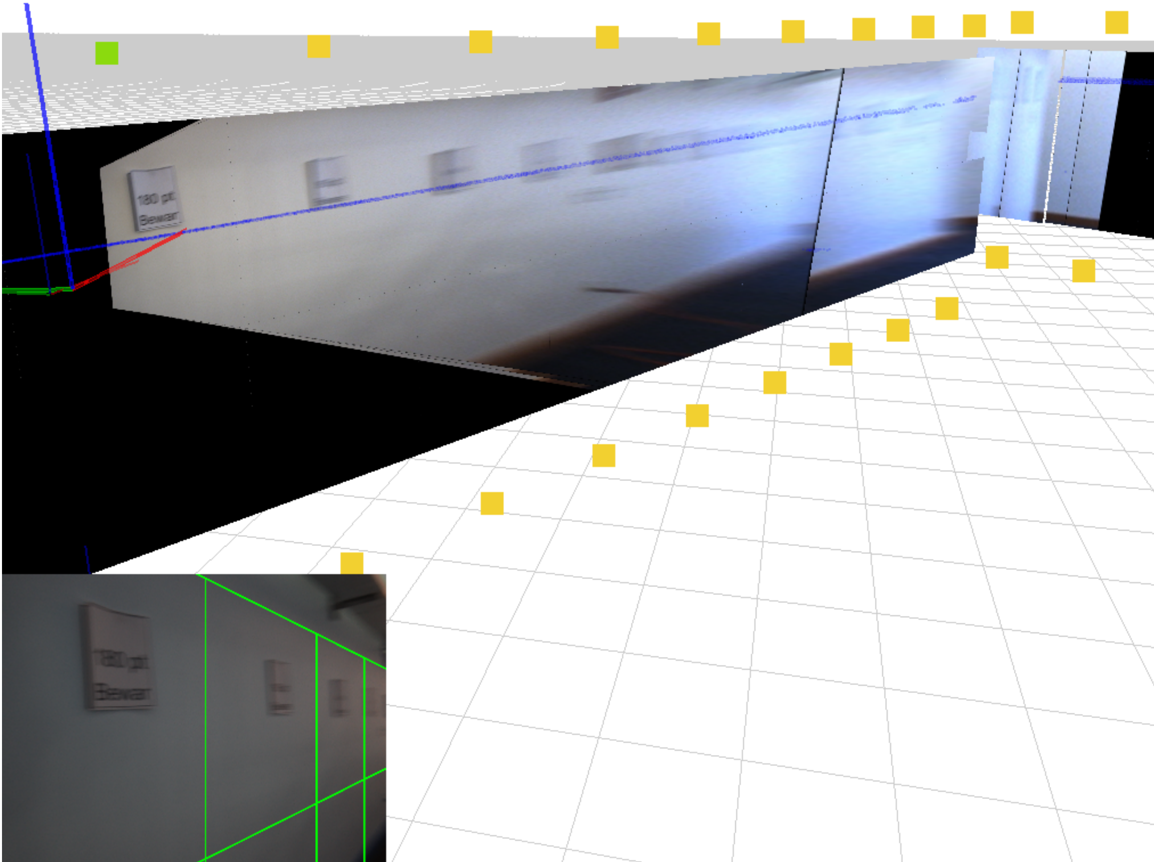


Figure 3-6: This screen shot was taken when the LIDAR and the camera data were paired at different times when the yaw rate of the sensor suite was high. Notice that the warping of the scene image to each tile is incorrect.

3.3 Hardware Prototypes

3.3.1 Hand-Carried Rig

The first rig we made was a hand-carried rig, first made by Maurice Fallon and modified by us. It was laser-cut out of plastic, and had mounting points for several sensors. The sensors that we mounted on this board were a Bumblebee2 camera, a

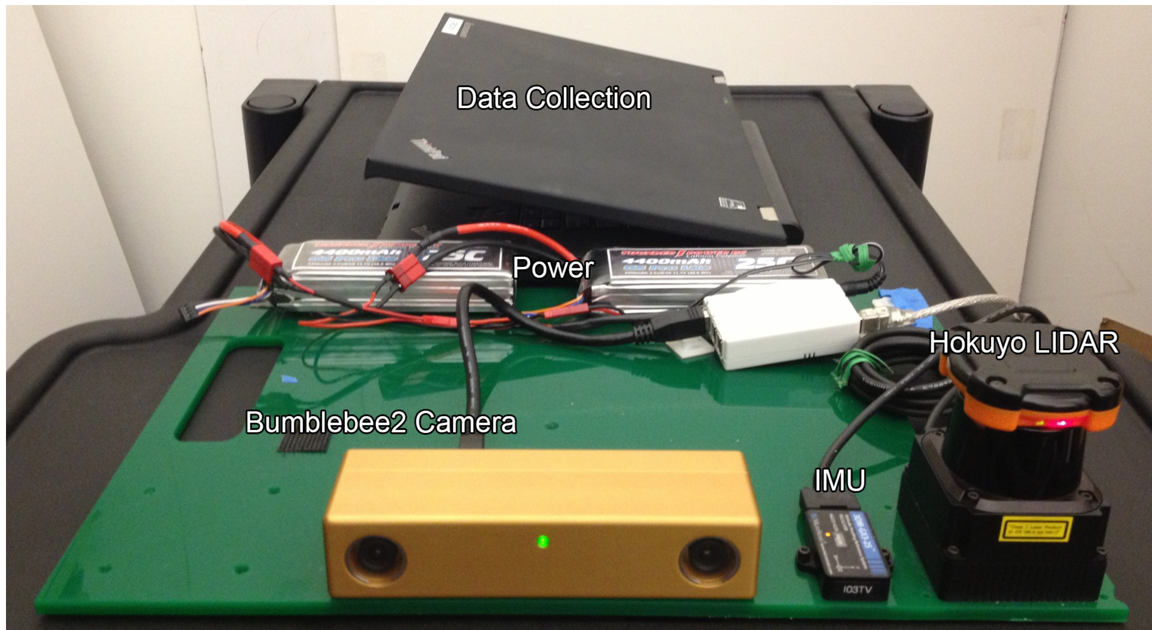


Figure 3-7: The first, hand-carried, rig with mounted Hokuyo, Bumblebee2, and IMU sensors. The rig also carries batteries, and there's a laptop in the background for data collection.

Hokuyo laser, and an IMU. The rig also carries two 12V batteries to power the Hokuyo and Bumblebee2 (the IMU is powered through USB). We carried a laptop for data collection along with the rig, but we could have used a much smaller laptop that could fit on the rig itself (e.g. the 11.6; 1.8 kg Eurocom Monster laptop, which was one of our development machines). This rig was used for both experiments described in Chapter 8.

3.3.2 Wearable Rig

The second rig was a modified version of a vest first created by Jon Brookshire. It carries a Hokuyo, a Kinect (unused in the version of the system described in the body of this thesis, but discussed in Chapter 9), an IMU, and a Dragonfly2 camera. This rig generates the same kind of data as the previous rig, and although we didn't use this rig for any experiments yet, we feel that this form-factor could be more suitable for the user.

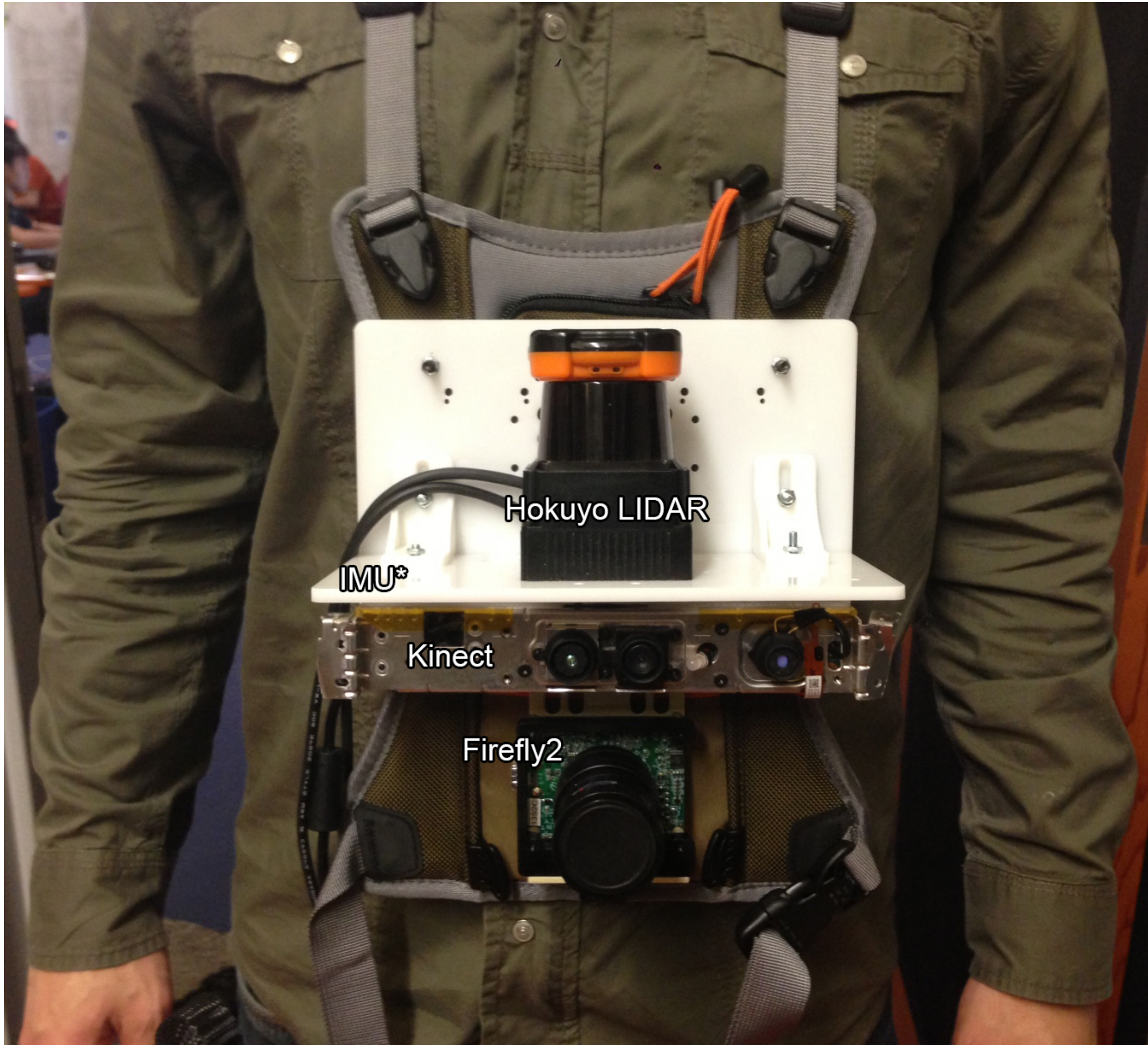


Figure 3-8: The second, wearable, rig with mounted Hokuyo and Dragonfly2 sensors. An IMU was later added where the word 'IMU*' is written on the Figure. There would also be a small laptop to record data or run the system, perhaps in a backpack.

Chapter 4

Scan Matcher, Tiler, and Warper

4.1 Scan Matcher

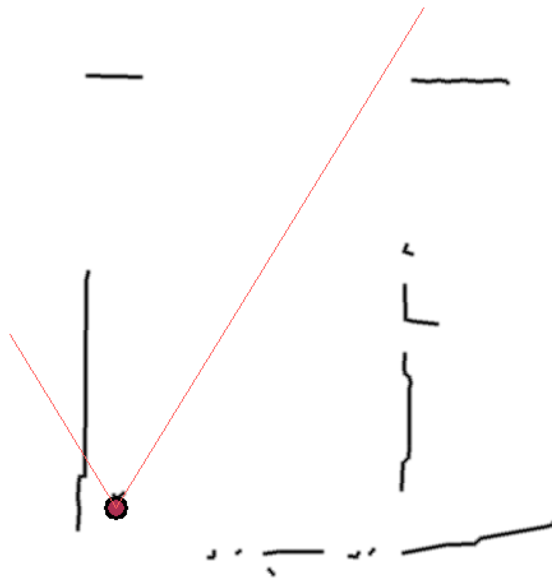


Figure 4-1: A sample output of the scan-matcher. The black lines represent the walls, the red disk represents the position of the sensor suite, and the red lines represent the field of view.

The scan matcher was used as a black-box module in this project. The scan-matcher module acts as the Simultaneous Localization and Mapping (SLAM) component in our system, providing both a map of the walls around the sensor suite along

with a pose estimate for the sensor suite. The output of the scan matcher that is used in the next steps is essentially a greyscale image of the walls in the environment. This map is passed on to the tiler, which splits each wall into smaller sections called 'tiles.'

4.2 Wall Tiler

The tiler takes the map produced by the scan matcher and breaks it up into tiles. The default tile width and height are 1 meter, although these are both adjustable parameters, and the tiles do not have to be square. The tiles are created as follows.

4.2.1 Extracting and Merging Line Segments

The tiler is triggered by the receipt of a camera image (15 Hz). The probabilistic Hough transform is run on the current map image from the scan-matcher, which produces line segment endpoints in the form of (x_1, y_1, x_2, y_2), as can be seen in Figure 4-2. The Hough parameters were chosen empirically in such a way as to detect actual walls and ignore noise (as can be seen by looking at the lines that were not detected by the Hough algorithm in Figure 4-2).

The endpoints of these line segments are then taken from the image coordinates to world coordinates. We now need to combine these new line segments with the existing line segments (for example, in the case where we saw the first part of a wall in one run and the rest of the wall in the next run). This is done as follows. We project the new line segments onto the existing line segments. If any pair of endpoints of the existing line segments and a new projected line segment are close enough and the angles of the two line segments (the existing and the new) are close enough, then the two line segments are adjacent to one another and are oriented similarly, so we merge them.

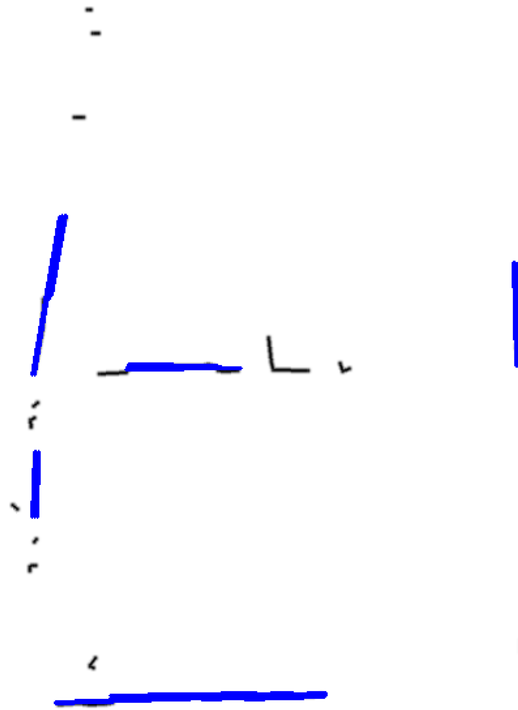


Figure 4-2: Line segments extracted from the scan-matcher output using the probabilistic Hough transform. The black lines are the walls from the scan-matcher and the blue lines are the output from the probabilistic Hough transform algorithm.

4.2.2 Tile Creation and Clustering

After the new line segments are merged with the existing line segments, we proceed to create tiles for each line segment. This is done by simply traversing the line segment and creating a new tile every *tile-width* meters (where the tile width is one meter by default). We make sure to traverse the line segment in such a way as to go through the first endpoint of the line segment so that the tiles are consistent. We do this by keeping a 'reference origin' off of which we make sure to offset all of the tiles. After the tile endpoints are created, we need to decide which way to point the normal vector. We make that decision by pointing the normal in whichever direction is closest to the camera (meaning that if we take a small step from the tile origin in the direction of the tile normal vector, we'd end up slightly closer to the camera).

After the tiles are created for each tile segment, we cluster them by simply combining two tiles if their origins are within a certain distance of one another. After

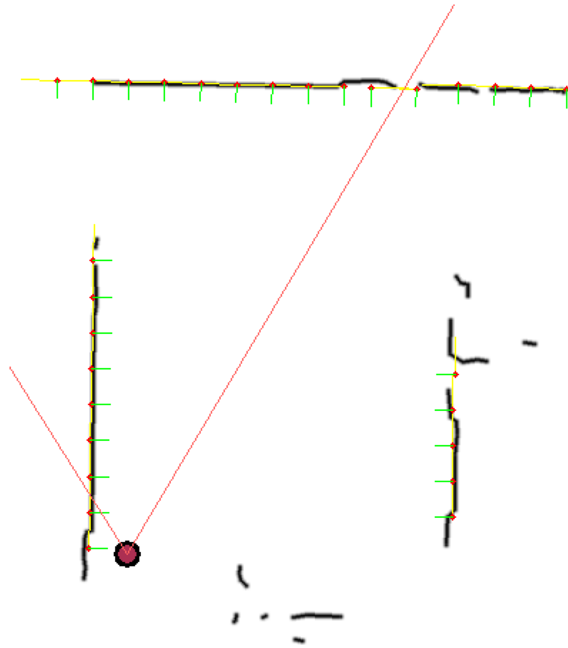


Figure 4-3: Tiles created from the extracted line segments. The red dots indicate the tile origins and the green lines indicate the normals.

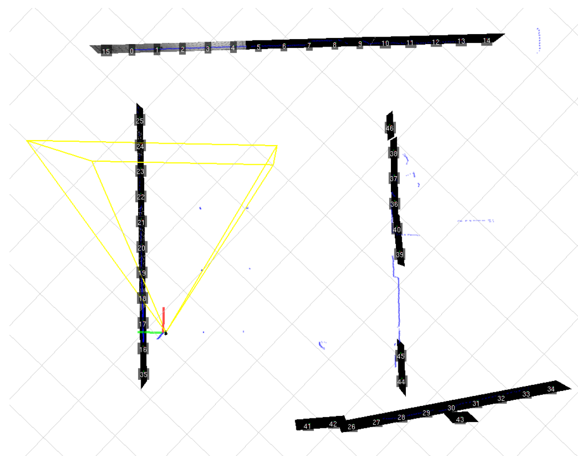


Figure 4-4: Tiles similar to the ones from Figure 4-3, viewed in the 3D viewer from above.

this process is done, we end up with the tiles that are seen in Figure 4-3. We need to cluster the tiles because the line segment code may produce a set of lines that are nearly parallel to one another, even though they all belong to the same physical wall. The clustering code ensures that if duplicate tiles are created for the same area of one physical wall, then that area will only be represented by one tile object in our

system.

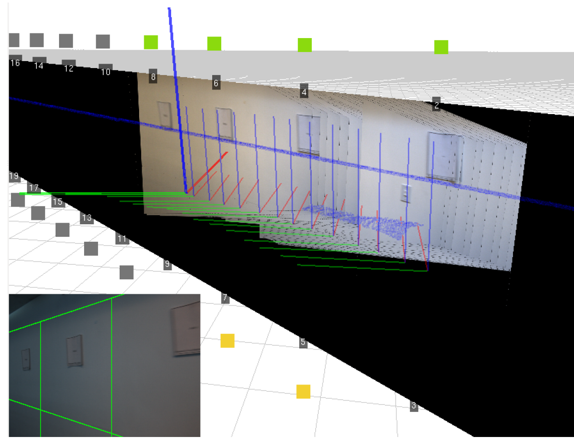


Figure 4-5: Two layers of tiles, as seen in both the 3D viewer and as tile outlines overlaid on the scene image. The bottom layer of tiles is mostly outside of the camera field of view.

4.3 Warper

The purpose of the warper is to acquire a fronto-parallel image of each tile from a scene image. As the camera is capturing images, each image must be projected onto the walls that are visible from the camera's current pose, and then the projected image must be cropped for each tile. The first implementation of the warper used OpenCV to compute a homography from the captured image to the visible walls, and then applied the homography to warp the image onto the walls. The second implementation of the warper (which eventually replaced the first version entirely) used built-in OpenGL functionality to do the warping using projective texture mapping. Both methods are discussed in detail in the sections that follow.

The output of the warper is a set of *observations*. Each observation represents one look at one tile, meaning that for each scene image captured by the camera, the warper is going to publish a number of observations equal to the number of the tiles in the camera's field of view. Each observation is going to contain a timestamp, the tile descriptor of the observed tile, the fronto-parallel image of the tile that was generated by the warping process, and the camera pose at the time that the observation was

taken. A complete description of the data types, including the observation type, can be found in Appendix B.

4.3.1 OpenCV Warper

The first version of the warper worked as follows. The first step was to trace four rays from the camera's position, through the corners of the image plane, to each known wall. The four intersection points marked where each of the corners of the captured rectangular image stretched to when projected onto a wall. Please see Figure 4-6 for an illustration. The four red disks at the corners of the green quadrilateral show these intersections in this situation. These four points can be found by solving a simple ray-plane intersection problem for each of the four rays and for the closest wall.

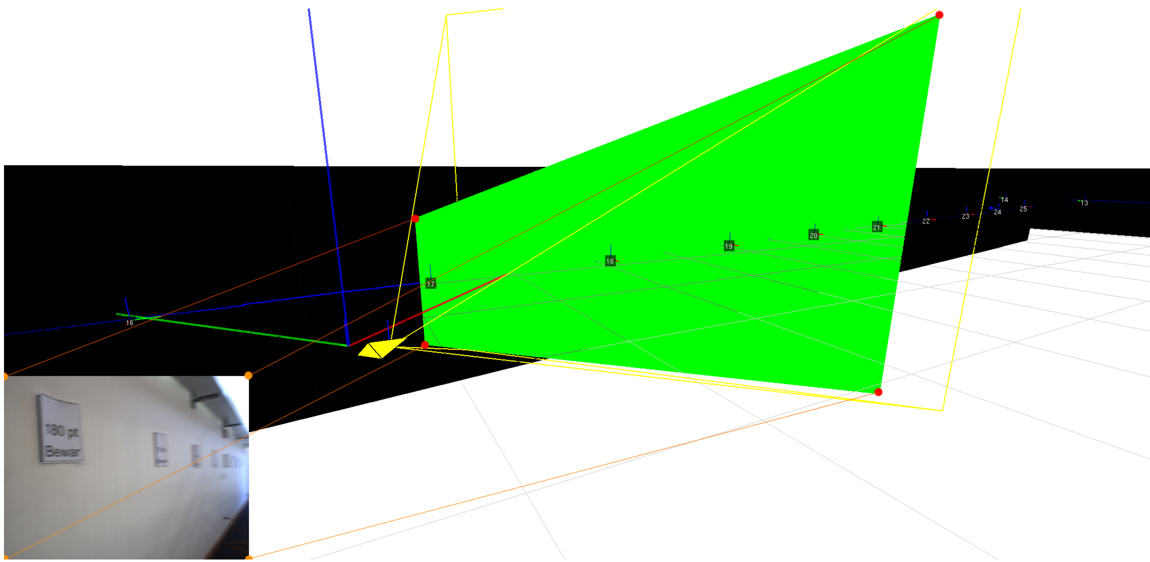


Figure 4-6: The homography that was computed to warp the scene image onto the wall.

Even though the four points all lie on one plane (the wall onto which we are projecting the image), these points have X, Y, and Z components. In order to compute the homography needed for our warping transformation, we need to get these points in two coordinates. This was done by rotating the four points by the negative of the yaw of the camera. This essentially aligns the points with the camera axis, and allows us to ignore the X component, reducing the points to two dimensions. Please

see Appendix C for an explanation of the coordinate systems that we used.

A homography was then computed using OpenCV from the four two-dimensional rectangular corners of the captured image to these four two-dimensional intersection points. After the homography was computed, it was applied to the scene image. This resulted in the warped image that is shown stretched on the wall in Figure 4-7. Then, for each tile, the tile bounding box was cropped from the warped image, producing a fronto-parallel image of the tile.

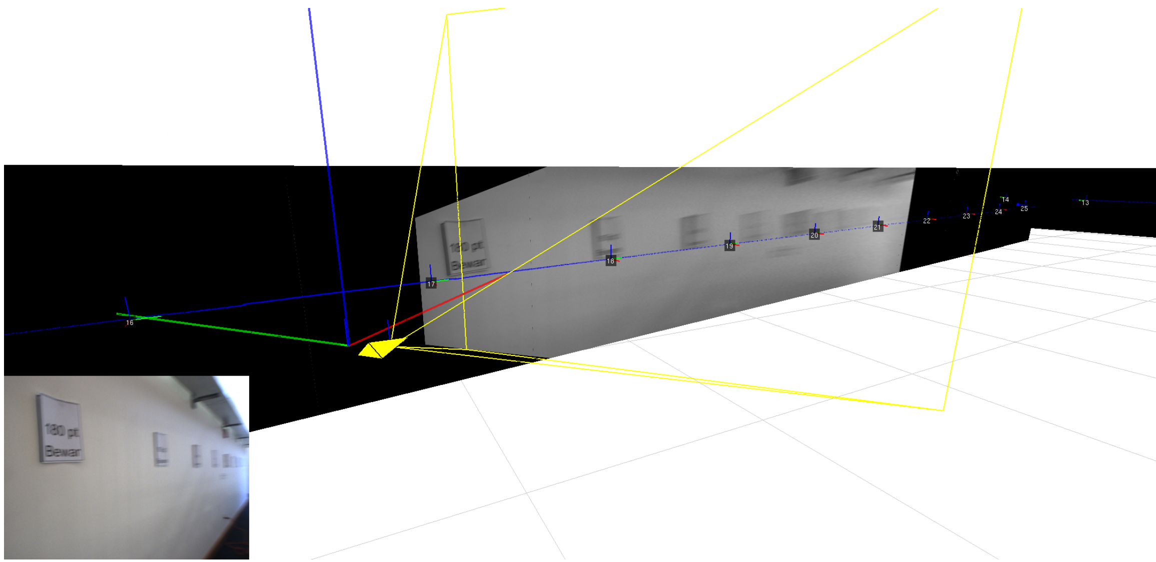


Figure 4-7: The result of applying the computed homography.

Although this method worked, it had several problems that rendered it unusable (or at least cumbersome as compared to the method that we'll describe next). First, the process of applying the homography to each scene image ran on the CPU and took about 2 seconds, which is prohibitively slow when we're trying run this operation on every frame. Second, the ray-plane intersection method would fail when some of the rays intersected behind the camera. Picture the camera pointing parallel to a wall, where the rays that trace through the left corners of the image intersect with the wall but the rays that trace through the right corners of the image never intersect. In this situation, special code must be written to detect where the wall ends, which proved to be an unnecessary bother.

4.3.2 OpenGL Warper

The second version of the warper is the version that is currently used it the system. It works by first generating all of the tiles known to the prioritizer as QUADs using their four corners. It then sets up a projector in the current location of the camera (as reported by the scan-matcher), and rotates it so that its yaw matches the yaw of the camera (also reported by the scan-matcher). The height of the projector is read from a constant. In addition, the projector roll and pitch are matched to the values that are reported by the IMU. Lastly, the projector's horizontal and vertical field of view are matched to the horizontal and vertical field of view of the camera that is capturing our scene images. The result is a projector whose parameters match those of the camera that captured the scene image.

We then create the scene image as an OpenGL texture and use OpenGL's projective texture mapping to project this texture onto the QUADs. Please see Figure 4-8 for an example of what the projected texture looks like.

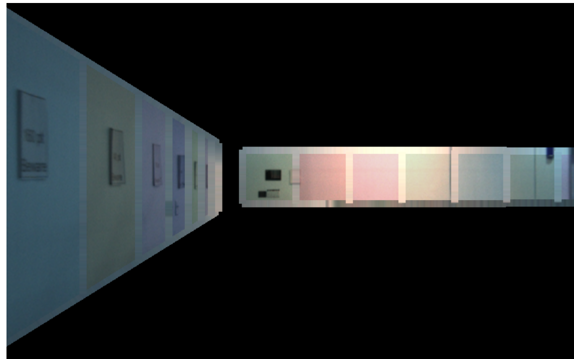


Figure 4-8: The tiles that are in the FOV are created as OpenGL QUADs and projective texture mapping is used to project the scene image onto them.

The last step is to actually generate the observations that contain a fronto-parallel tile image for each tile. This is done by setting up a virtual camera to point at each tile. The virtual camera is given horizontal and vertical field of view values that make it so that each tile fits exactly in the virtual camera's view from one meter away. This camera is pointed at each tile that is in the field of view, and the resulting buffer is saved to an observation. This process is illustrated in Figure 4-9.

The whole process – setting up the tiles, projecting the scene image onto the tiles, pointing a virtual camera at a tile, and saving the tile image – takes around 5 milliseconds per tile. That time is split about evenly between actually doing the projections and downloading the resulting pixels from the GPU into main memory.

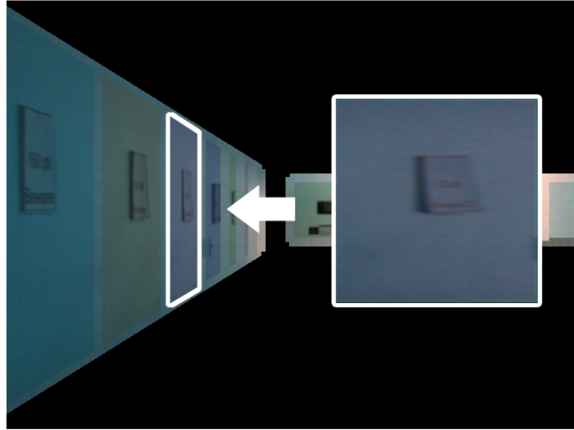


Figure 4-9: A virtual camera is configured so that each tile fits exactly in the FOV from a meter away. This camera is then placed one meter in front of each tile, and a tile image is captured.

Chapter 5

Prioritizer

5.1 Overview

The prioritizer keeps track of all of the tiles that we've seen, as well as all of the information associated with these tiles (such as the text we've spotted on them). It does so through a collection of objects called TileWithText.

TileWithText objects hold all of the information that we want to remember about a tile. They hold the tile's origin and four corners, the 'composite' image (a combination of all of the looks that we got of this tile), the text spot results that we got for this tile (bounding boxes and character values for text detection/decoding results, and bounding boxes and words for character/word clustering and language model results).

The prioritizer periodically decides which tiles to send on for more processing based on an 'importance' score. Each tile's score is determined using the factors outlined below. The prioritizer keeps a priority queue that's keyed by the priority score and tries to have outstanding text spotting queues for four tiles at a time.

5.2 Scoring

The importance score indicates how important it is to process this tile first. Its range is [0,1] and it's composed of several components, including distance, obliquity, and

other factors outlined in the subsections that follow. The total score is a weighted sum of all of these factors. The weights were chosen empirically based on a few trial runs where the tile priority orders were assessed by hand. The numbers were then tweaked to produce a good ordering in several different experiments.

5.2.1 Distance

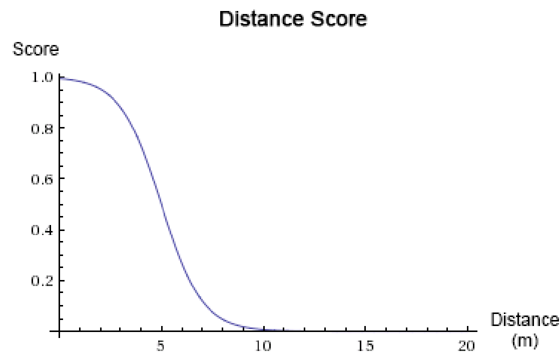


Figure 5-1: The score as a function of the distance from the camera to the tile, in meters.

How far away were we from the tile when we got the best look at it? This distance is updated only when the overall score for a closer observation is greater than the old best score (meaning that a close observation at, say, a bad angle may not end up replacing the distance for a slightly-further- away-but-ultimately-better observation. Obviously, closer observations are better than observations that from further away. The score function is a sigmoid that drops at around 5 meters, and the exact function is $1-1/(1+exp(-(x-5)))$. As with the other score components, the parameters were chosen empirically — we noticed that detection and decoding performance dropped significantly after 5 meters, so we decided to prioritize tiles that are that far away significantly.

5.2.2 Obliquity

This is defined for our purposes as the angle from the camera to the origin of the tile. Being in front of a tile is good, and being to its side is bad. This score is currently

designed so that it's 1.0 when facing the tile directly and 0.0 when facing parallel to the tile. In between, the score falls off linearly. The exact score function is $2/\pi * \theta - 1$, where the angle θ is computed by taking the arch cosine of the dot product of the camera x axis unit vector and the tile normal unit vector.

5.2.3 Speeds

How fast is the camera spinning during this observation? If the camera is spinning too fast, the image will have a lot of motion blur and won't be as useful for text detection and decoding. The X, Y, Z, and roll, pitch, yaw speeds are all recorded in the observation LCM messages that are generated by the warper. These parameters can be used to give a lower score for an observation that has a high overall speed (or a single high component, e.g. yaw rate). This metric is not currently used in the prioritizer, however, and its weight is set to 0.

5.2.4 Spatial Prior

How likely is there to be text on this tile? This score is dependent on the environment. In an office setting, text is most likely to appear at head height. In an airport, however, important text is likely to appear overhead, near the ceiling. This score encodes this information. Right now, we only extract tiles at two levels: at head height and at knee height, so it's a very simple function. It is currently a piecewise function that is 1.0 for head-height tiles and 0.75 for knee-height tiles. In the future, this function may be more complicated, and may depend on things other than the tile location (for example, we may use this score to prefer vertical planes over planes with weird orientations). We may also form the function for this score for different locales by training our system using the ground truth module (please see Section 6.1).

5.2.5 Inspection State

This component of the score is highest for tiles that have been seen with the camera but have not been sent to the text spotter, is medium for tiles that have been seen

by the camera and have already been processed by the text spotter, and is lowest for tiles that have been detected by the laser but have not been seen with the camera. In the case that the tile has been processed by the text spotter already, the score starts low but rises with the number of text detection results that the text spotter provided. The score rises with every additional text detection result until it bumps the overall score high enough so that it is sent to the entire text spotter, which will also provide bounding boxes and string values.

5.2.6 Time Decay

In addition, tiles are time-decayed. So, if we got a really good look at a tile a long time ago, and we've since moved away and got an equally good look at another tile somewhere else, the newer tile is going to be preferred over the old tile. Right now this is linear decay, but we can change it to exponential decay.

Chapter 6

Text Spotter

This is the final step of the process. The text spotter receives a request to spot text in a tile from the prioritizer, and tries to return bounding boxes along with a string guess for the value of the text within the bounding boxes. Most of the work on the text spotting components was done by Nick Wang, and much of this section is taken from our paper [35]. I would also like to thank Ben Mattinson for his contributions to the ground truth module.

6.1 Ground Truth

The ground truth module was designed to be a 'perfect' text spotter. This means that its interface is designed to look just like the regular text spotter — it takes a set of observations for a tile as an input and produces bounding boxes with their respective strings as output — but the text spotting results are provided by a human, as opposed to an algorithm.

The purpose of the ground truth module is to have the real text available as an overlay in the viewer so that the text spotting results could be compared with 'ground truth' human-generated text spotting results.

Because we have 15 captured camera frames every second, it is unreasonable to provide ground truth value for every frame. Instead, we leverage the system's ability to keep a persistent representation of the environment to keep track of ground truth

labels. We designated one frame roughly every two seconds as a 'keyframe.' These keyframes were presented to a human in a tool called LabelMe (developed by a group in CSAIL) that enabled him or her to provide bounding boxes and strings for every piece of text that can be read by a human. Please see Figure 6-1 for a screen shot of this interface in use with one of the keyframes from the first experiment (described in Section 8.1.1). In this keyframe, the text '180 Pt Bewar' can clearly be read by a human, so it is tagged in the interface.

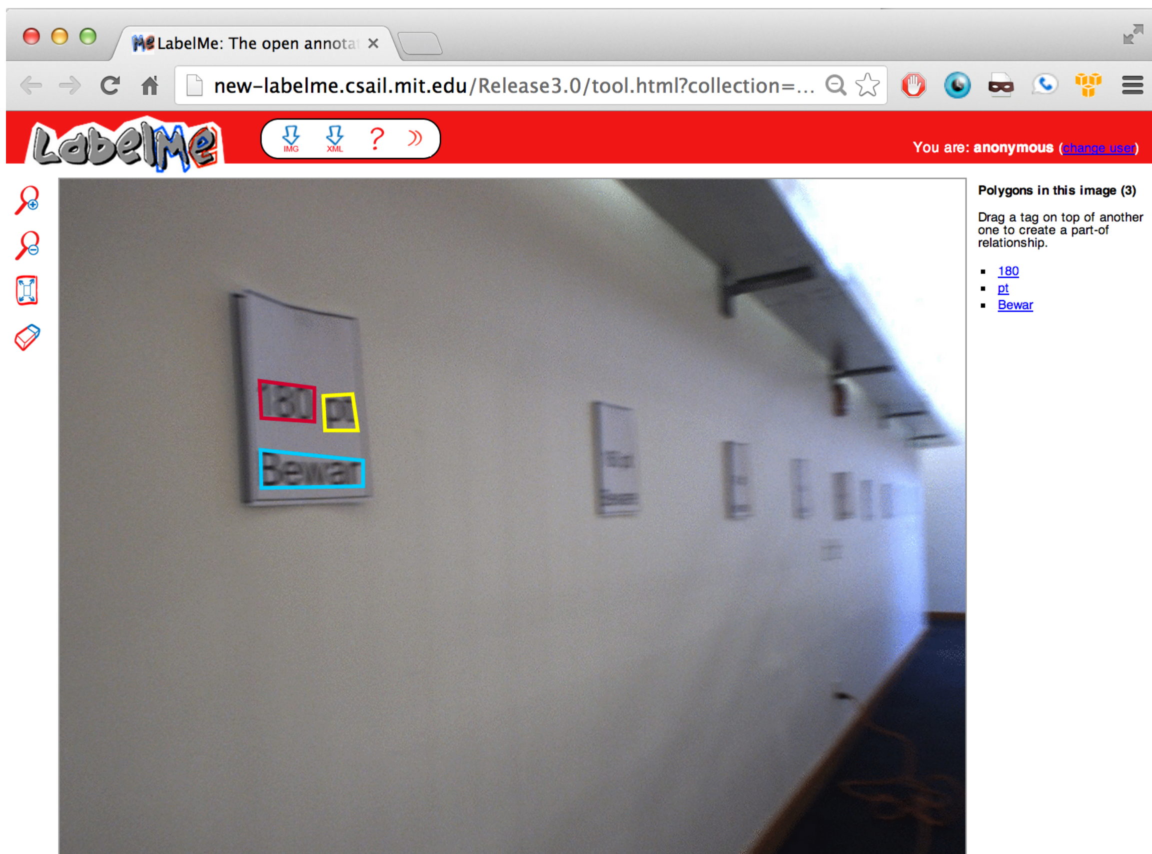


Figure 6-1: A screen shot of the LabelMe interface that was used to generate ground truth labels.

LabelMe provides an XML file with the bounding boxes and string values for every image. These XML files are parsed and imported into the system, where we begin to persist them in the environment. We persist the bounding boxes and string values by projecting the bounding box corners onto the wall on which they were tagged. After the labels are projected onto the wall, we determine which tile they belong to,

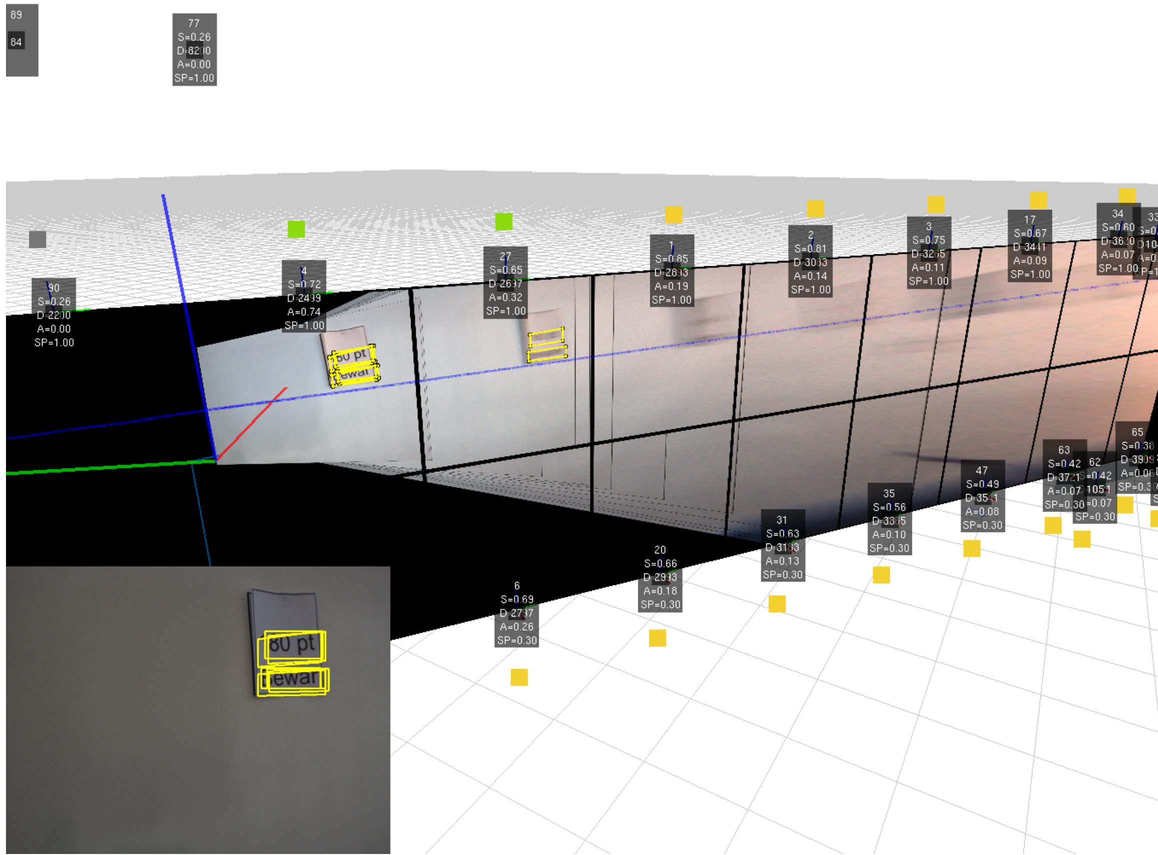


Figure 6-2: Ground truth bounding boxes, persisted on their respective tiles. Only the bounding boxes, and not the associated values, are shown in this screen shot.

and store them in the tile. Now, no matter where the sensor suite moves or how the scene is viewed in the 3D viewer, the labels will always stay in the same place on the tile to which they were assigned. Because we may have multiple labels for the same piece of text, the labels may be rendered as overlapped bounding boxes (this may be changed in the future so that multiple labels for the same text are combined based on the overlapped bounding box area). An example of such persisted ground truth labels can be seen in Figure 6-2.

6.2 Text Detector

The first stage of text detection applies an image pyramid to each tile in preparation for multi-scale DCT, with coefficients as per Crandall et al. [5]. The bounding box of

each text detection is then inspected using MSER [25] to extract shape descriptors, including aspect ratio and compactness. We set the MSER parameters as follows: aspect ratio less than 8, and compactness greater than 15. Scale-relevent parameters are estimated according to real-world setting (8 pixels per cm), corresponding to a minimum text height of 3 cm, and a minimum MSER region of 3 cm^2. The parameters for DCT detection include a minimum edge density of 8 edge-pixels per 8×8 window using Canny edge detection, with high and low hysteresis parameters equal to 100 and 200, respectively. For MSER detection, regions smaller than 5 pixels are discarded, and the parameter delta (the step size between intensity threshold levels) is set to 3 for higher sensitivity to blurry inputs. Both the DCT and MSER computations are implemented in OpenCV, with running times of about 10 msec and 300 msec, respectively.

6.3 Text Decoder

Decoding proceeds as follows. First, the image regions produced by either DCT or MSER (as gray-scale or binary images) are processed by the Tesseract OCR engine. Using the provided joined character chopping and broken character association, the binary inputs are segmented into one or multiple observations, i.e., the segmentation results from a MSER region. Tesseract outputs with too large an aspect ratio are removed. Each block is classified into a few candidates with confidence scores, for example, "B", "E" and "8" for the crop of an image of character "B." We set a minimum confidence score of 65 given by Tesseract to reduce incorrectly decoded characters. Running time depends on the number of input regions, but is usually less than 300 msec.

6.4 Character and Word Clustering

A clustering module is used to: (a) merge decoded characters across multiple observations, and (b) cluster groups of decoded characters into word candidates. For (a),

a distance predicate is implemented by Euclidean distance, text height, similarity between decoded results. Multiple observations can be obtained either across multiple frames or within a single frame. The parameters of multi-frame integration depend on system calibration. For (b), the confidence of groups of decoded characters, size of decoded characters, and Euclidean distance are applied. The confidence is determined by the number of decoded characters in the group; only groups with confidence above a threshold are selected. The threshold is $\sqrt{N_{obs}}/k$, where N_{obs} is the total number of accumulated decoded characters, and k is an arbitrary scalar. The bounding box of each decoded character in selected groups are overlaid on a density map, which is then segmented into regions. All selected groups of decoded characters are assigned to a region, representing a word candidate.

6.5 Finding Optimal Word Configuration

To extract whole words, we implemented a graph to combine spatial information (block overlaps). The output is a sequence of characters with each character comprising a small number of candidates provided by Tesseract. To recover the optimal word string each candidate from each group of decoded characters is considered as a node in a trellis, where the probability of each node arises from normalized voting using confidence scores. The prior probability is computed using bi-grams from an existing corpus [13]. We retain the top three candidates for each group of decoded characters, and use Viterbi's algorithm [30] for decoding. We seek an optimal character sequence W^*, as shown in Eq 6.1, where $P(Z|C_i)$ is the probability of nodes from the confidence-scored observations, and $P(C_i|C_{i-1})$ is the prior probability from the bi-gram.

$$W^* = \underset{w}{\operatorname{argmax}} \left(\sum P(Z|C_i) P(C_i|C_{i-1}) \right) \tag{6.1}$$

The optimal character sequence W^* is what is returned from the text spotter, along with the corresponding bounding boxes. These results are sent to the prioritizer as a response to the text spotting request for a tile.

Chapter 7

Output

There are several ways to see the text spotting results. We have modified a 3D viewer that was written by the group. The 3D viewer shows the environment, the tiles, and the sensor suite pose, and allows the user to fly around and inspect the environment from any position and angle. The 2D viewer overlays the tile boundaries and the text spotting results on the rectified camera images. The data output module provides a way to export data that the system has generated for further processing.

7.1 3D Viewer

The 3D viewer allows the user to move a virtual camera to any arbitrary X and Y position in the environment, and also set any desired virtual camera pitch and yaw. The 3D viewer accepts OpenGL-like commands over LCM that allow it to render the same types of objects as OpenGL. We use this capability to draw the wall tiles with a texture. In addition, the 3D viewer is able to display floating text labels, which we use to label the wall tiles with their tile IDs and the priority score components. The viewer also accepts an (X, Y, Z) LCM message from the scan matcher and uses it, along with a (Roll, Pitch, Yaw) LCM message from the IMU, to show the camera pose as a triad. The viewer also directly shows the point cloud that was obtained from the laser. The viewer is able to capture screen shots and videos and save them to disk. In addition, it allows the user to adjust certain parameters, such as the way the sensor

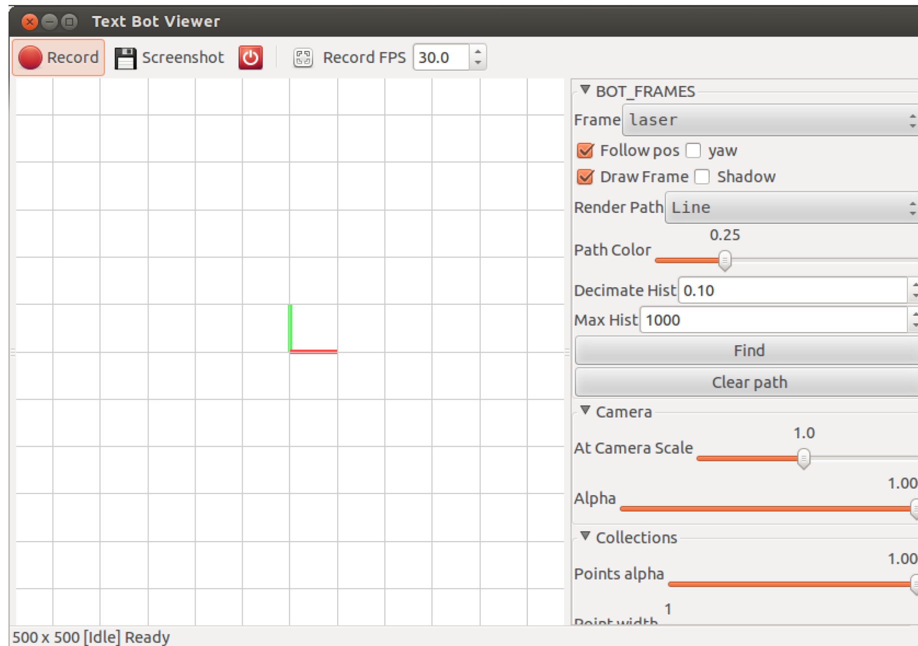


Figure 7-1: 3D viewer (no data).

suite pose is displayed, the transparency of the laser point cloud, the position of the camera images, and other parameters. Figure 7-1 shows the viewer in the state where no data has been sent to it. The viewer output with data can be seen throughout this thesis, for example in Figure 2-3.

7.2 2D Viewer

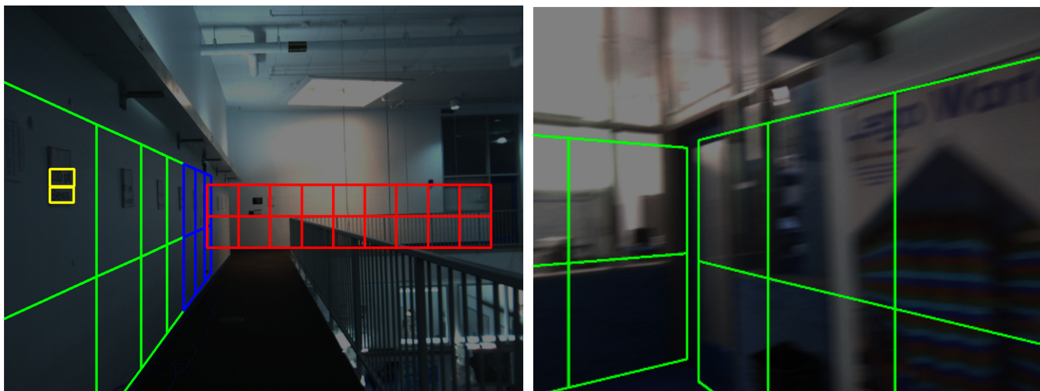


Figure 7-2: Two examples of the 2D viewer output.

The 2D viewer overlays the known tile information and text spotting results on

top of the camera-captured scene images. The four corners for each tile are projected from their actual 3D locations to the camera image plane, and the offset from the top-left corner of the image plane is computed. This is used to get four pixel locations (one for each corner) on the scene image, and connected line segments are drawn to connect these corners. The result is that the 3D tiles are seen as projections on the camera images. The end result can be seen in Figure 7-2. The same process is done to text spot bounding boxes, and they are also drawn on the scene image.

7.3 Data Output

The data output module allows us to export data out of the system that would allow us to work on the text detection and decoding algorithms without running the system. An output directory is designated for the data output module. Each image in this directory is an image as it was captured from the camera (the scene images). The image filename contains the timestamp of this image which uniquely identifies it within this data set.

The subdirectories in this directory that are named tile_XX contain three types of files.

The first type is tile_XX/corners_tileXX.txt, which describes the corners of the tile and the tile normal vector in following format:

```
1 <top left x, top left y, top left z>
2 <top right x, top right y, top right z>
3 <bottom right x, bottom right y, bottom right z>
4 <bottom left x, bottom left y, bottom left z>
5
6 <tile normal vector x, tile normal vector y, tile normal vector z>
```

The coordinate system origin is at the camera pose at the start of the log file (so it is fairly arbitrary), and the axes are oriented as described in Appendix C.

The coordinates all have units of meters. Each tile is square, and is one meter on

each side (at least in this data set). The tile origin is in the center of the tile (at the intersection of the two tile diagonals).

There is only one such file in each tile subdirectory.

The second type is tile_XX/timestamp_tileXX.png. This is an 800 by 800 pixel image of tile XX. Each tile is square, and is one meter on each side. This image is taken by projecting the scene image of the same timestamp onto all visible tiles from that camera pose, and then viewing tile XX head-on from one meter away with a virtual camera whose FOV fits the tile exactly.

The third and last type is tile_XX/timestamp_tileXX.txt which provides data on the camera pose when the tile fronto-parallel image was produced. This file includes the following data (in the order specified here):

```
camera x, camera y, camera z
camera yaw (yaw axis is +z)
speed x, speed y, speed z
yaw rate

distance from the camera to the tile in meters
speeds are in meters per second, and the yaw rate is in rad/second
dot product of the tile normal with a vector from the camera to the tile
priority order of the tile (ranging from 1 to the number of tiles)
```

Chapter 8

Experiments and Results

This chapter explains the experiments that we performed and the results that we were able to obtain. We performed three main experiments. The first experiment was performed to continuously test our system as we were developing it. We collected data from a fairly typical (and simple) office setting at the beginning of the project, and performed all of our development with this data in mind. This initial data collection is described in Section 8.1.1. After a few months of development, we collected another data set that was more realistic and more challenging. This data set had real text, whereas the first experiment had text that we printed and put up ourselves. This experiment is described in Section 8.1.2. Finally, the last experiment was a controlled movement of the sensor suite in order to get a more quantitative measure of system performance. We moved the camera in various predictable ways and recorded the text spotting performance. This experiment is discussed in Section 8.2, along with the text spotting results obtained.

8.1 Experiments

8.1.1 Stata Third Floor: Initial Data Collection

Motivation

Our team collected some initial data to aid in designing and implementing the system. The purpose of the initial experiment was to collect data that presented texts of varying difficulty. The initial data collection experiment was also conducted so that we could begin to write the infrastructure with the form of the real data in mind.

Experiment Design

Figure 8-1: The first experiment setting and a close-up of one of the signs.

This initial trial was conducted in the hallway outside of our lab area, in what looks like a fairly typical office setting. We hung signs 8.5 by 11 inch signs that had text on them describing the font and containing a short sentence. The experiment setting and a sample sign can be seen in Figure 8-1. We used the hand-held sensor suite described in Section 3.3.1, and moved it back and forth along the hallway on a cart. We moved the cart parallel along the wall (past the signs on the left) first, then we swept the cart along the wall as it was facing the signs, and then moved it along

the hallway at roughly a 45-degree angle to the signs. The purpose of moving the sensor suite in this manner was to collect multiple views of each sign from different angles and different distances.

8.1.2 Stata Second Floor: More Realistic Experiment

Motivation

The second experiment was designed to be more realistic. The first experiment was designed to collect some data to get the project started, whereas this experiment was motivated by the desire to test the system in a more realistic setting. In addition, we also wanted to make sure that we weren't engineering the system specifically to fit the first data set.

Experiment Design

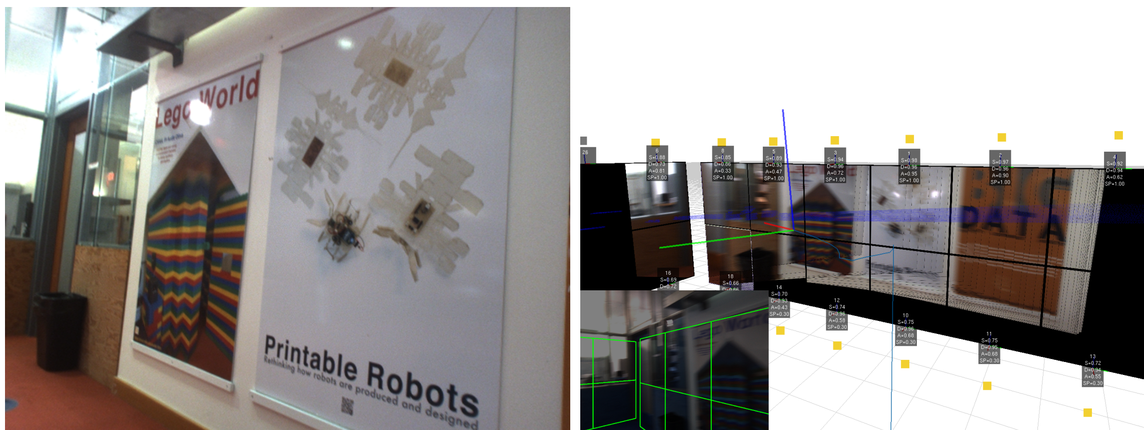


Figure 8-2: The second experiment setting and as seen through the 3D viewer.

The first experiment used printed signs that we created specifically for the purpose of testing and developing the system. This experiment, however, did not use such signs. Instead, we took the data on the second floor of the Stata Center where we saw some research posters with a lot of text. Unlike the first experiment, we carried the hand-held rig (instead of rolling it around on a cart). We walked around the second floor lounge, pausing briefly to look at each poster. We made sure to get a good look at each poster, however, before moving on to the next poster.

8.2 Evaluation Experiment and Results

We conducted one additional experiment which we will describe here. While the other two experiments were designed to provide qualitative feedback about how well our system is performing, this experiment was designed to provide a more quantitative measure of how are system compares to alternative approaches.

The evaluation is performed using a metric defined over m ground truth words and n decoded words. The $m \times n$ pairs of strings are compared using minimum edit distance d_{ij} for the ith ground truth word and the jth decoded word. A score S_{ij} for each pair is calculated as $(N_i - d_{ij})/N_i$, where N_i is the number of character of ground truth word i, when $N_i - d_{ij} > 0$, whereas S_{ij} is 0 otherwise. The accuracy is then measured by Eq 8.1, where the weight of each ground truth word w_i is set to $1/max(m,n)$ to penalize false alarms when $n > m$.

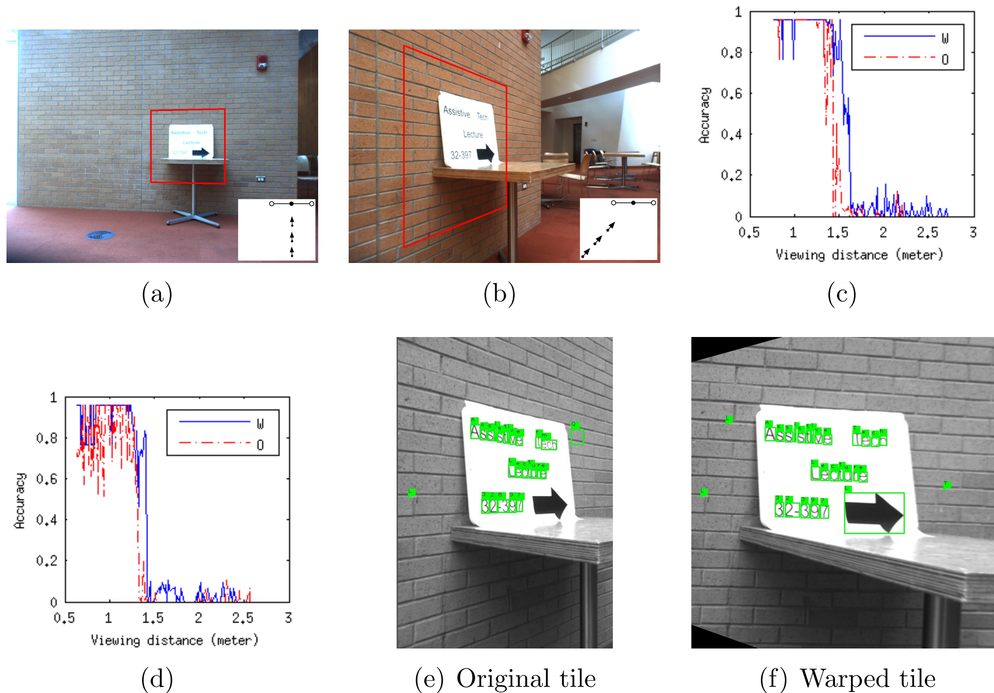
$$\text{Accuracy} = \sum_i w_i \max_j(S_{ij}) \tag{8.1}$$

8.2.1 Warping Accuracy with Distance and Obliquity

We mounted all equipment on a rig placed at waist height on a rolling cart, with the LIDAR sampling at 40 Hz and the camera sampling at 15 Hz. We attached signs with 140-point (5 cm) font at various wall locations. We pushed the cart slowly toward and by each sign to achieve varying view angles with respect to the sign's surface normal (Fig. 8-3(a) and Fig. 8-3(b)). The experiments were designed to evaluate text-spotting performance under varying viewing distance and obliquity, given that such factors effect the degree of bluriness in imagery.

Each original tile and its warped observation cropped from scene image frame was sent to Tesseract, our baseline decoder. Text spotting performance vs. the baseline is plotted as a function of viewing distance (Fig. 8-3(c) and Fig. 8-3(d)). Examples are shown in Fig. 8-3(e) and Fig. 8-3(f).

The results suggest that the baseline decoder works poorly when text is observed at distances greater than 1.5 m, and generally performs better for the warped observation



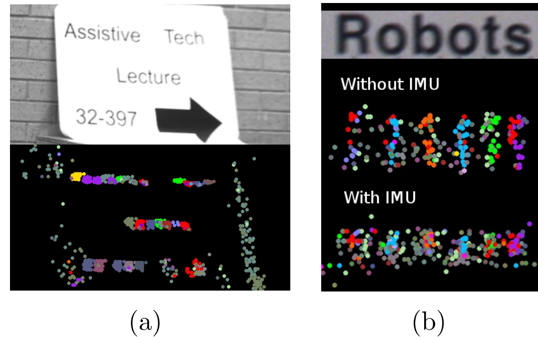
(a) (b) (c)

(d) (e) Original tile (f) Warped tile

Figure 8-3: Experiment settings and accuracy comparison of original and warped observations. (a) The normal of the surface is roughly antiparallel to the viewing direction. (b) The normal of the surface is about 45 degrees away from the viewing direction. Plots (c) and (d) show the accuracy of baseline decoding of original (O) and warped (W) tiles with respect to viewing distance for observations (a) and (b). (e) An original tile observation from 0.71 meters. (f) The warped observation corresponding to (e). The accuracy scores of (e) and (f) are 0.67 and 0.96, respectively.

than for the original ones. When the viewing direction is about 45 degrees to the surface normal, the accuracy of warping observation is more consistent than that of original, which may be due to the skewed text line and perspective transformation of characters.

Given the limitation of baseline decoder, our proposed method intends to extend the capability of detecting and decoding more blurry imagery by spatiotemporal fusion of multiple observations. One key factor for integration is: how well are the warped observations aligned? we report the alignment and the calibration of sensors in the next section.



(a) (b)

Figure 8-4: The distribution of decoded characters. (a) There were only slight vertical and horizontal shifts. (b) Comparison between data with and without IMU for the second dataset (hand-carried). There were longer vertical drifts without IMU, but use of the IMU reduces drift.

8.2.2 Alignment of Warped Observations

The distribution of decoded characters is shown in Fig. 8-4(a) and 8-4(b). Misalignment among certain decoded characters was measured manually. In 8-4(a), the logs were collected when the sensors were placed on a cart. The results suggest that the drift of decoded characters was uncertain to within about 20 pixels.

Another log was collected when the rig was hand-carried at about chest height by an observer who walked within an indoor environment. Fig. 8-4(b) demonstrates that imagery, to be aligned, required shifts of around 20 pixels horizontally and 70 pixels vertically without IMU data. When IMU data were integrated, the vertical shifts required reduced to around 35 pixels.

Given the alignment, we chose the experiment described in Fig. 8-3(b) to report the text-spotting performance of fusion of multiple observations. The parameter settings for clustering decoded characters and word candidates are shown in Table 8.1. Comparing single and multiple frame integrations, Euclidean distance is the major factor for merging decoded characters, whereas the threshold of number of decoded character per group is the major factor for grouping to word candidates.

	Single Frame	Multiple Frames
Merge decoded characters		
Euclidean distance	10	30
Text height scalar	2	2
Decoded text similarity	1	1
Group to word candidates		
Threshold of characters per group	1	$\sqrt{N_{obs}}/k$
Threshold parameter k		1.3
Size outlier scalar	5	2
Text height outlier scalar	5	2
Characters per word	3	3
Word aspect ratio min	1	1
Bounding box horizontal increment	0.3	0.3
Bounding box vertical increment	0.05	0.05

Table 8.1: *Parameter settings for clustering decoded characters and word candidates.*

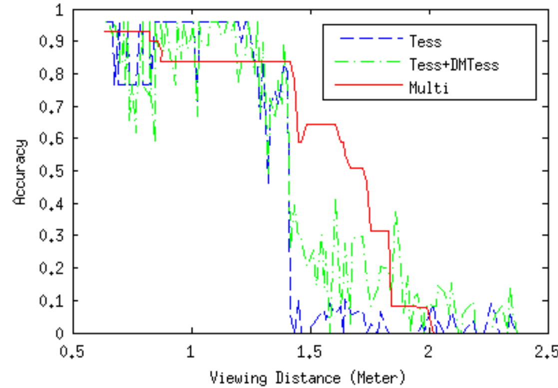


Figure 8-5: Accuracy comparison with respect to viewing distance for observations.

8.2.3 Performance with Multiple Observations

We demonstrate that the proposed method combines noisy individual observations into a higher-confidence decoding. Fig. 8-5 plots the accuracy of (a) running Tesseract on the entire tile observation (Tess), (b) combining (a) and the proposed spotting pipeline into a single-frame detector (Tess+DMTess), and (c) fusing multiple observations from the proposed pipeline (Multi). The area under curve (AUC) values are 0.71, 0.79, and 0.91, respectively; these represent the overall performance of each spotting pipeline. The results suggest that Tess+DMTess moderately extends (to

2.4m from 1.5m) the distance at which text can be decoded, and Multi moderately improves the accuracy with which blurry text can be decoded (since blur tends to increase with viewing distance). We found that reducing the rate of false positives is critical to successful fusion process, because high false-alarm rate tends to cause our clustering method to fail. We will continue to investigate our observation that Tess+DMTess outperforms Multi for close observations (1-1.5m).

Chapter 9

Future Work

The next steps in the project will be to add support for hardware and write software that will allow us to extract text from any surface (and not just walls). In order to do this, we need to get a better description of the environment. Currently, the scan-matcher gets a 2D description of the locations of the walls from the planar point cloud provided by the laser. However, in order to find and tile any arbitrary surface, we will need a point cloud that will describe an entire room.

Figure 9-1: The MultiSense SL sensor. The image was taken from the DRC website.

We can get this description by using the MultiSense SL sensor. This sensor has

been provided to the group as part of the DARPA Robotics Challenge, and has a spinning Hokuyo laser, along with a stereo camera. This sensor can be seen in Figure 9-1. In addition, the group owns a version of the sensor that was designed to have only the stereo camera (and not the spinning Hokuyo laser).

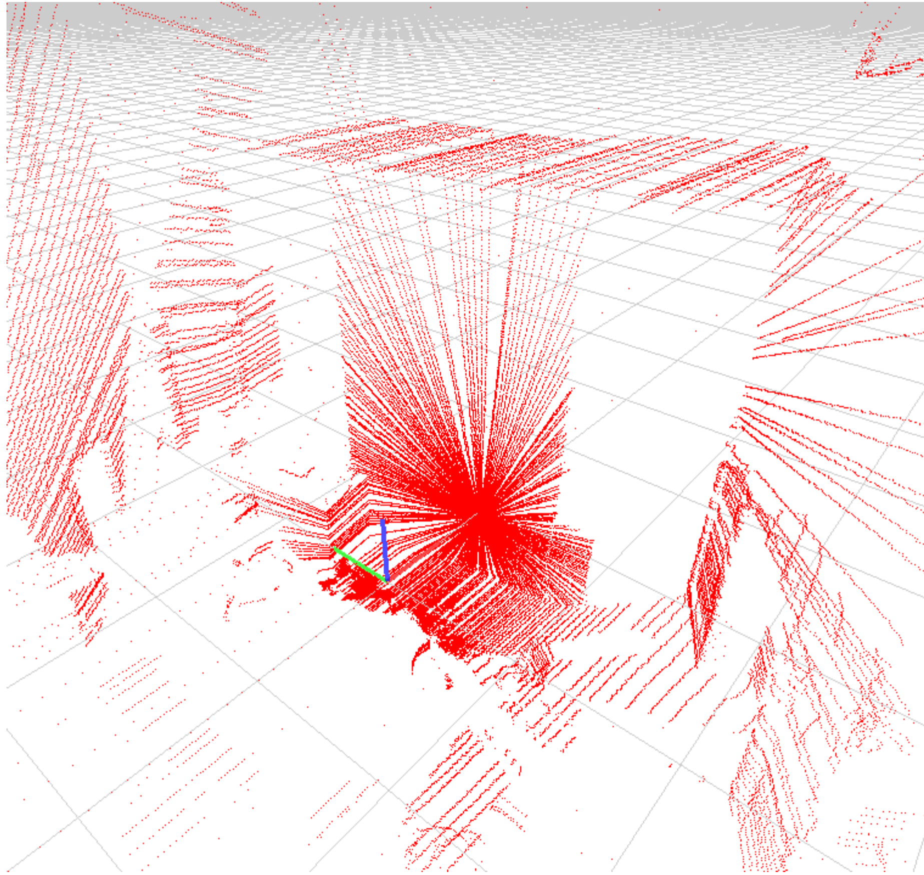


Figure 9-2: An example of a point cloud generated by the rotating Multisense laser.

If we use the spinning laser, we can get a point cloud that looks like the point cloud seen in Figure 9-2. We can then extract the plane segments in this point cloud using built-in PointCloud Library functions and tile then using a plane tiler that we've started writing. The tiler would project existing nearby tiles onto each extracted plane segment and create tiles adjacent to the existing tiles until it reaches the edge. This allows us to see different portions of the same surface and grow the tile set accordingly. A preliminary implementation of the plane segmentation and plane tiling has been written and tested with the Multisense SL laser data. The planar tiler can be seen working in Figure 9-3. In this figure, the plane boundaries were expanded

by hand to show that the tiler adapts correctly. Figure 9-4 shows the tiler running on a plane segment extracted from the MultiSense laser data.

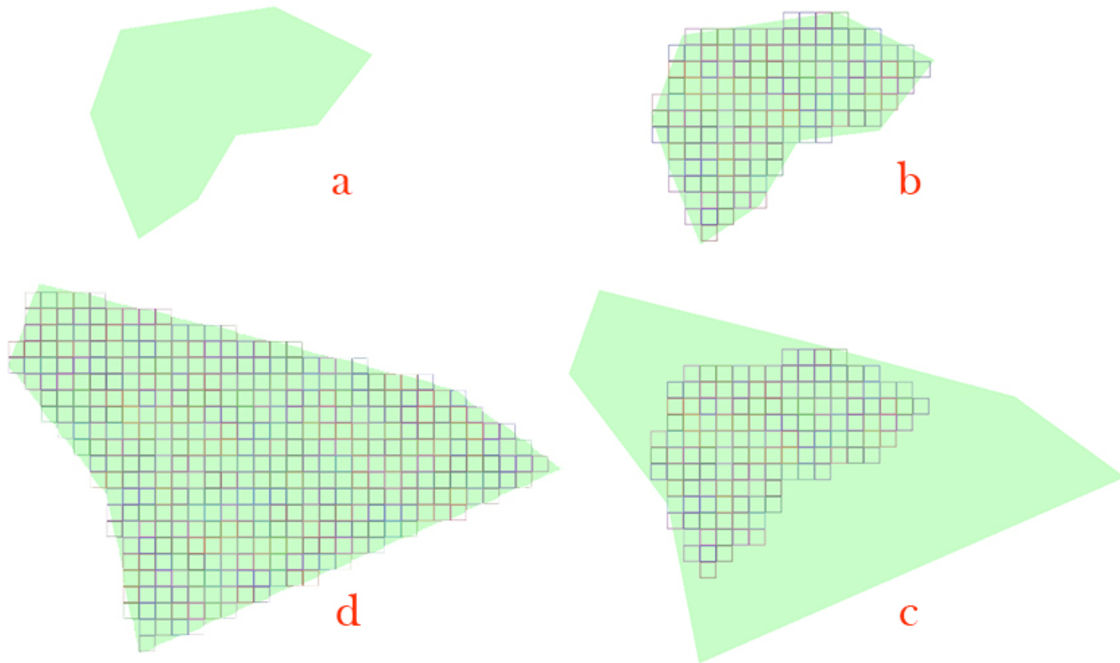


Figure 9-3: Planar tiler running on a plane segment found in the environment (in this case, a piece of a wall). We first see the plane segment in (a). The plane segment is tiled in (b) and later expands in (c) as more of it is visible. The tiles expand to cover the newly-discovered parts of the plane in (d).

Finally, we can get a point cloud from the stereo camera, instead of the laser. This approach allows us to remove the laser from the system entirely, and use the stereo camera for both tiling and for estimating the camera's pose (through the use of visual odometry). Unfortunately, the stereo camera requires the environment to have some sort of texture in order to get depth points. Some typical results can be seen in Figure 9-5, where the point cloud and the left camera image are shown side-by-side. Notice that there is no depth information for the plain white wall.

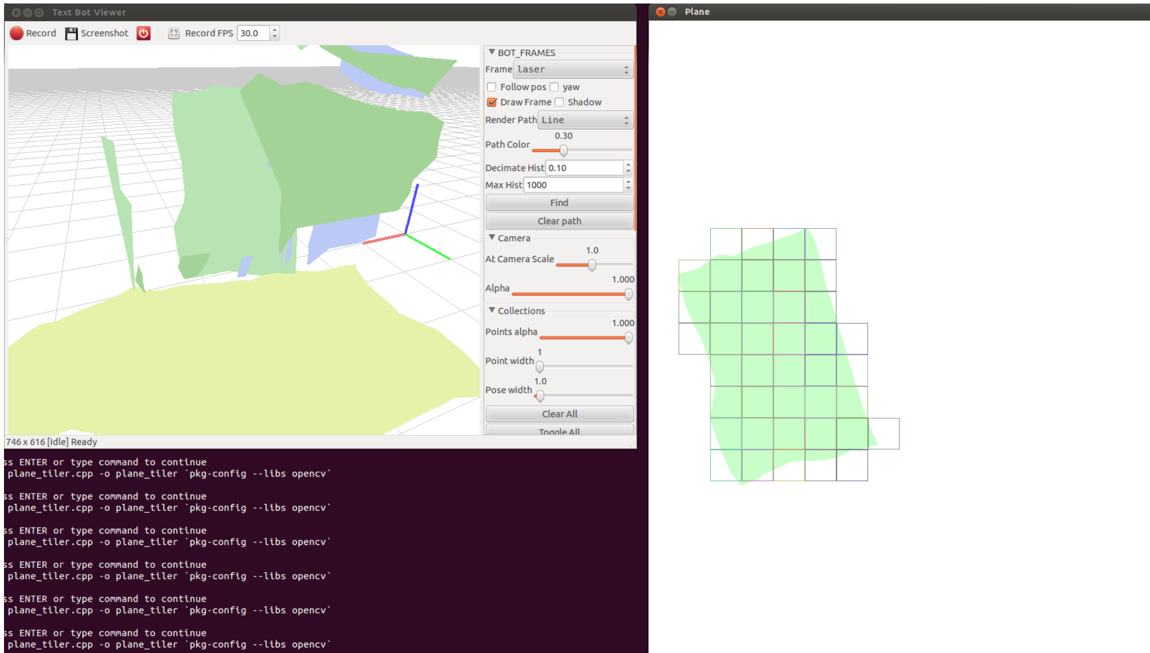


Figure 9-4: Planar tiler running on a plane segment found in the environment through the use of the MultiSense SL laser.

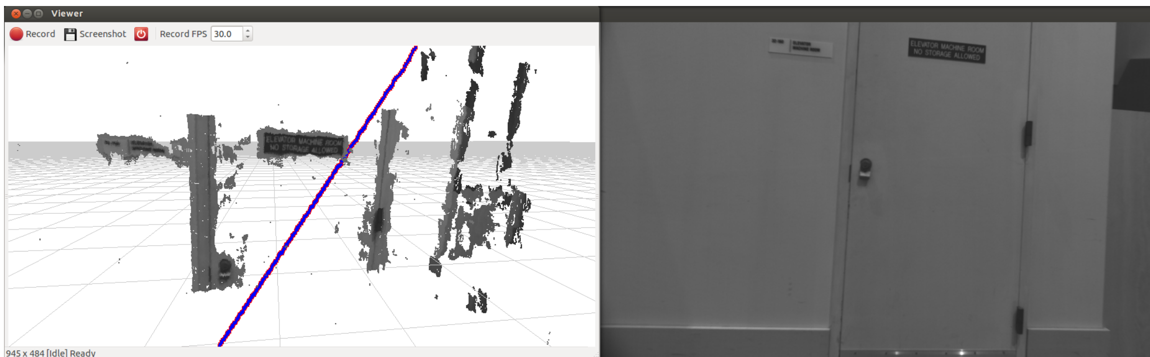


Figure 9-5: An example of a point cloud generated by the Multisense stereo pair.

Appendix A

Installation Instructions

To install the system, you first need to install all of the requirements and tools that the project depends on, and then check out the project from Subversion and build it. The system was developed for and tested on Ubuntu 12.04. Maurice and Nick use Ubuntu 12.04 natively, whereas I ran it in a virtual machine using VMWare Fusion 4.1.3 on top of Mac OS X 10.7.4. Nick used a ThinkPad W530 with 16Gb of RAM. These instructions assume that you're running Ubuntu 12.04, but newer versions will most likely work as well.

Additional information is available on the project install Wiki page in the RVSN Wiki:
`https://groups.csail.mit.edu/rvsn/wiki/index.php?title=Text-loco-getting-started`

A.1 Requirements

A.1.1 Aptitude Packages

Install these packages using Aptitude (apt-get install): cmake build-essential libglib2.0-dev openjdk-6-jdk python-dev libgsl0-dev gsl-bin libusb-1.0-0-dev libgtk2.0-dev freeglut3-dev python-gtk2 libgtk2.0-dev libjpeg8-dev (installing this may break ROS electric, which wants libjpeg-dev instead) libvtk5-dev gtk-doc-tools libqhull-dev libtinyxml-dev libboost1.48-all-dev libsoil-dev

A.1.2 LCM

Lightweight Communications and Marshalling (LCM) is a set of libraries and tools for message passing and data marshalling, targeted at real-time systems where high-bandwidth and low latency are critical. The group uses it to send data between processes and machines. You can install it by following the instructions on the project homepage: `https://code.google.com/p/lcm/`

A.1.3 Enable Large UDP Packets

If you don't run the commands contained at the bottom of the link below, LCM will complain and ask you do run them. `http://lcm.googlecode.com/svn/www/reference/lcm/multicast.html`. To avoid having to run them all the time, you can edit /etc/sysctl.conf. For Ubuntu, add these lines to '/etc/sysctl.conf' and reboot:

```
1 net.core.rmem_max=104857600
2 net.core.rmem_default=104857600
```

A.1.4 libbot

The libbot project provides a set of libraries, tools, and algorithms that are designed to facilitate robotics research. The text localization project uses libbot, and so you need to install it. Check out the code from svn. The instructions are on the 'Source' tab on this page: `http://code.google.com/p/libbot/` Then, cd into the directory where the 'tobuild.txt' file lives and run: sudo make BUILD_PREFIX=/usr/local

A.2 Check Out Bocelli Text-Loco

The code lives in an svn repository. To get it, run: svn co `https://svn.csail.mit.edu/bocelli/projects/text_loco` The project uses the pods system to organize code, and includes various dependencies like the Point Cloud Library and OpenCV,

so it's quite big. After you check it out, just run make in the top-level directory to build the project.

A.3 Running the System

We use a program called Procman to run the system. Procman is a part of libbot and should be installed after following the instructions above. The checkout will contain a configuration file for Procman. In order to run the system, first run 'bot-procman-deputy -n text$_{l}oco' tostartaProcmanDeputy.Then, runProcmanSheriff(`bot-procman-sheriff')withtheconfigurationfile, whichislocatedinbocelli/projects/text_{l}oco/config.Thisfileconta$

Appendix B

LCM Types

The types that define wall tiles and wall tile sets:

```
package text_loco;

struct tile_t
{
  int32_t tile_id;

  // First time this tile was observed:
  int64_t utime;

  // the normal vector (length normalized to 1)
  double normal_direction[3];

  // The bottom left corner, also (0,0) in the tile coordinates
  // This is an xyz point in meters in the global reference frame
  double origin[3];

  // The direction of the x and y axis of tile
  // (These are normalized to length of 1)
  double x_axis_direction[3];
  double y_axis_direction[3];

  // The width and height of the tile (in meters)
```

```
23   double width;
24   double height;
25
26   // the tile's corners in 3D
27   double top_left_corner[3];
28   double top_right_corner[3];
29   double bottom_left_corner[3];
30   double bottom_right_corner[3];
31 }
```

```
1 package text_loco;
2
3 struct tile_set_t
4 {
5   int64_t utime;
6
7   // the timestamp of the image this pertains to
8   int64_t linked_image_utime;
9
10  // an array of tiles
11  int32_t num_tiles;
12  tile_t tiles[num_tiles];
13 }
```

The type that defines the cart (X, Y, Z, Yaw) pose:

```
1 package text_loco;
2
3 struct xyzt_t
4 {
5   int64_t utime;
6   double xyz[3];
7   double t;
8 }
```

The type that defines the observations:

```
1 package text_loco;
```

```
2
3  struct tile_observation_t
4  {
5    int64_t utime;
6    tile_t tile;
7    int64_t image_utime;
8    double pose_x;
9    double pose_y;
10   double pose_z;
11   double pose_t;
12   double pose_r;
13   double pose_p;
14   double speed_x;
15   double speed_y;
16   double speed_z;
17   double speed_t;
18   bot_core.image_t tile_image;
19   bot_core.image_t scene_image;
20   double distance;
21   double obliquity;
22   double num_pixels;
23 }
```

Text spotting types:

```
1  package text_loco;
2
3  struct ocr_block_group_t
4  {
5    // tile id
6    int32_t tile_id;
7
8    // vector<OCRBlock> ocr_blocks;
9    int32_t num_ocr_blocks;
10   ocr_block_t ocr_blocks[num_ocr_blocks];
11 }
```

```
package text_loco;

struct ocr_block_t
{
  // vector<OCRChar> ocr_chars;
  int32_t num_ocr_chars;
  ocr_char_t ocr_chars[num_ocr_chars];

  // int num_obs;
  int32_t num_obs;

  // Rect bbox
  int32_t x;
  int32_t y;
  int32_t w;
  int32_t h;

  // pt in 3D
  double center[3];
}
```

```
package text_loco;

struct ocr_char_t
{
  // Rect bbox
  int32_t x;
  int32_t y;
  int32_t w;
  int32_t h;

  // only take top 3 candidates
  // vector<String> candidates;
  string candidate1;
  string candidate2;
  string candidate3;
```

```
16
17   // vector<double> confidences
18   double confidences[3];
19
20   // int group;
21   int32_t group;
22 }
```

```
1 package text_loco;
2
3 struct text_spot_t
4 {
5   int64_t utime;
6
7   // the timestamp of the image this pertains to
8   int64_t linked_image_utime;
9
10   // bounding box at the image
11   int32_t x;
12   int32_t y;
13   int32_t w;
14   int32_t h;
15
16   // an array of MSER regions (GRAY image)
17   int32_t num_regions;
18   bot_core.image_t regions[num_regions];
19
20   string ground_truth_text;
21   string ocr_text;
22
23   // tile id
24   int32_t tile_id;
25
26   // the text spot's corners in 3D
27   double top_left_corner[3];
28   double top_right_corner[3];
```

```
29    double bottom_left_corner[3];
30    double bottom_right_corner[3];
31  }
```

Appendix C

Coordinate Systems

This appendix describes the different coordinate systems that were used in the project. There are several coordinate systems, and all of them are related to one another through a rigid body transformation.

The main coordinate system is called the *laser* coordinate system. This coordinate system has an x-axis that points in the forward direction of the sensor suite, a y-axis that points to the left from the sensor suite, and a z-axis that points up from the sensor suite. This coordinate system is centered on the Hokuyo laser, and it moves along with the rig so that the origin is always at the laser location.

The second coordinate system is the *camera* coordinate system. This coordinate system is oriented so that the x-axis points to the right of the sensor suite, the y-axis points down from the sensor suite, and the z-axis points forward in the direction of the sensor suite. The origin is designed so that it always stays at the location of the Bumblebee2 camera. This was done by creating a transformation that took the laser coordinate system and rotated and offset it so that it matched the orientation described above and the physical offset on the rig. We measured the distance between the laser and the camera lens and encoded this information in the offset part of the transformation. The camera is at position (3 cm, -11.5cm, 0 cm) in the laser frame (meaning that the camera was mounted slightly forward and to the right on the board, compared to the laser). Please see Figure C-1 for an illustration of these two coordinate frames. In the figure, the two coordinate frames can be seen as the RGB

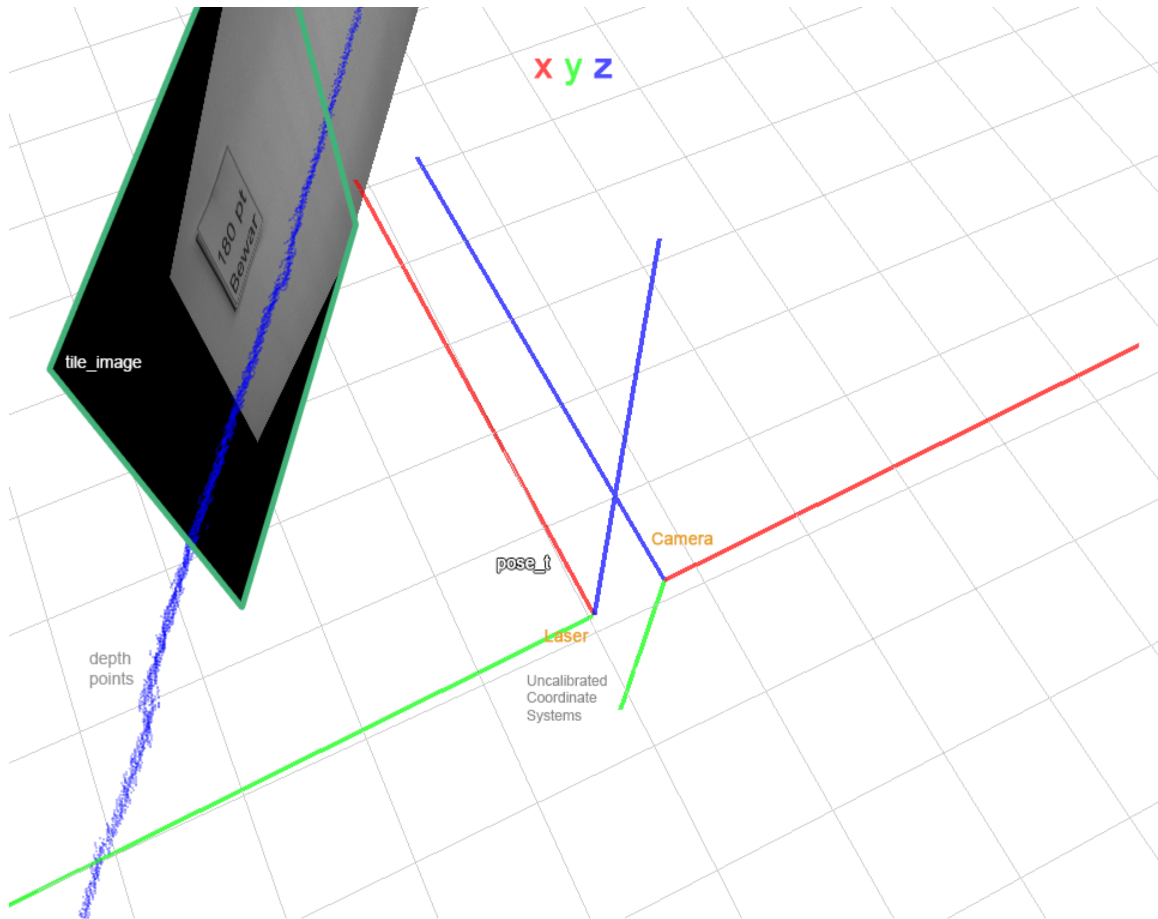


Figure C-1: The laser (left) and camera (right) coordinate frames. The sensor suite is pointing at a tile.

triads, where red is the x-axis, green is the y-axis, and blue is the z-axis. The laser frame is on the left.

Besides these coordinate frames, there is also a *global* coordinate frame. This frame assumes its origin as the very first laser origin, and has the same orientation as the laser frame. This means that as the scan-matcher updates the position of the rig, the laser frame origin moves away from the global origin (and so does the camera origin, as it's linked to the laser origin).

Bibliography

[1] A. Bachrach, S. Prentice, R. He, and N. Roy. RANGE - robust autonomous navigation in GPS-denied environments. *J. of Field Robotics*, 28(5):644–666, September 2011.

[2] A. Bonci, T. Leo, and S. Longhi. A bayesian approach to the hough transform for line detection. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(6):945–955, 2005. `doi:10.1109/TSMCA.2005.853481`.

[3] C. Case, A. Coates B. Suresh, and A. Y. Ng. Autonomous sign reading for semantic mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3297–3303, 2011.

[4] X. Chen and A. Yuille. Detecting and reading text in natural scenes. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2004.

[5] David Crandall, Sameer Antani, and Rangachar Kasturi. Extraction of special effects caption text events from digital video. *International Journal on Document Analysis and Recognition*, 5(2-3):138–157, 2003. URL: `http://dx.doi.org/10.1007/s10032-002-0091-7`, `doi:10.1007/s10032-002-0091-7`.

[6] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2963–2970, 2010.

[7] M. F. Fallon, H. Johannsson, J. Brookshire, S. Teller, and J. J. Leonard. Sensor fusion for flexible human-portable building-scale mapping. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Algarve, Portugal, 2012.

[8] S. Farsiu, M.D. Robinson, M. Elad, and P. Milanfar. Fast and robust multiframe super resolution. *Image Processing, IEEE Transactions on*, 13(10):1327–1344, 2004. `doi:10.1109/TIP.2004.834669`.

[9] H. Goto and M. Tanaka. Text-tracking wearable camera system for the blind. In *Proc. of the Intl. Conf. on Document Analysis and Recognition (ICDAR)*, 2009.

[10] D. S. Hayden, C. Vondrick, S. X. Jia, Y. Landa, R. C. Miller, A. Torralba, and S. Teller. The accuracy-obtrusiveness tradeoff for wearable vision platforms. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition (CVPR)*. Extended Abstract.

[11] Xian-Sheng Hua, Pei Yin, and Hong-Jiang Zhang. Efficient video text recognition using multiple frame integration. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 2, pages II–397–II–400 vol.2, 2002.

[12] A. Huang, E. Olson, and D. Moore. LCM: Lightweight communications and marshalling. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 2010.

[13] Michael N. Jones and D. J. K. Mewhort. Case-sensitive letter and bigram frequency counts from large-scale english corpora. *Behavior Research Methods, Instruments, & Computers*, 36(3):388–396, 2004. URL: `http://search.proquest.com/docview/85612900?accountid=12492`.

[14] Keechul Jung, Kwang In Kim, and Anil K. Jain. Text information extraction in images and video: a survey. *Pattern Recognition*, 37(5):977 – 997, 2004. URL: `http://www.sciencedirect.com/science/article/pii/S0031320303004175`, `doi:10.1016/j.patcog.2003.10.012`.

[15] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. Robotics*, 24(6):1365–1378, December 2008.

[16] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[17] Huiping Li and David Doermann. Text enhancement in digital video using multiple frame integration. In *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pages 19–22, 1999.

[18] S.M. Lucas. Icdar 2005 text locating competition results. In *Proc. of the Intl. Conf. on Document Analysis and Recognition (ICDAR)*, pages 80–84 Vol. 1, 2005. `doi:10.1109/ICDAR.2005.231`.

[19] J Matas, O Chum, M Urban, and T Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761 – 767, 2004. URL: `http://www.sciencedirect.com/science/article/pii/S0262885604000435`, `doi:10.1016/j.imavis.2004.02.006`.

[20] C. Merino-Gracia, K. Lenc, and M. Mirmehdi. A head-mounted device for recognizing text in natural scenes. In *Proc. of Camera-based Document Analysis and Recognition (CBDAR)*, pages 29–41, 2011.

[21] Anand Mishra, Karteek Alahari, and C.V. Jawahar. Top-down and bottom-up cues for scene text recognition. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2687–2694. IEEE, 2012.

[22] Gregory K Myers and Brian Burns. A robust method for tracking scene text in video imagery. *CBDAR05*, 2005.

[23] L. Neumann and J. Matas. A method for text localization and recognition in real-world images. In *Asian Conf. on Computer Vision (ACCV)*, pages 770–783, 2004.

[24] L. Neumann and J. Matas. Text localization in real-world images using efficiently pruned exhaustive search. In *Proc. of the Intl. Conf. on Document Analysis and Recognition (ICDAR)*, pages 687–691, 2011.

[25] L. Neumann and J. Matas. Real-time scene text localization and recognition. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2012.

[26] David Nister and Henrik Stewenius. Linear time maximally stable extremal regions. In *Eur. Conf. on Computer Vision (ECCV)*, pages 183–196, 2008.

[27] E. Olson. Real-time correlative scan matching. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 4387–4393, Kobe, Japan, June 2009. `doi:10.1109/ROBOT.2009.5152375`.

[28] Sung Cheol Park, Min Kyu Park, and Moon Gi Kang. Super-resolution image reconstruction: a technical overview. *Signal Processing Magazine, IEEE*, 20(3):21–36, 2003. `doi:10.1109/MSP.2003.1203207`.

[29] I. Posner, P. Corke, and P. Newman. Using text-spotting to query the world. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3181–3186, 2010.

[30] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. `doi:10.1109/5.18626`.

[31] T. Sato, T. Kanade, E.K. Hughes, and M.A. Smith. Video OCR for digital news archive. In *Content-Based Access of Image and Video Database, 1998. Proceedings., 1998 IEEE International Workshop on*, pages 52–60, 1998.

[32] R. Smith. An overview of the tesseract OCR engine. In *Proc. of the Intl. Conf. on Document Analysis and Recognition (ICDAR)*, page 629633, 2007.

[33] R. Smith. History of the Tesseract OCR engine: what worked and what didn't. In *Proc. of SPIE Document Recognition and Retrieval*, 2013.

[34] A. Torralba, B.C. Russell, and J. Yuen. Labelme: Online image annotation and applications. *Proceedings of the IEEE*, 98(8):1467–1484, 2010. `doi:10.1109/JPROC.2010.2050290`.

[35] C. H. Wang, Y. Landa, M. Fallon, and S. Teller. Spatially prioritized and persistent text detection and decoding. In *Proc. of Camera-based Document Analysis and Recognition (CBDAR)*, 2013.

[36] K. Wang, B. Babenko, and S. Belongie. End-to-end scene text recognition. In *Intl. Conf. on Computer Vision (ICCV)*, 2011.

[37] K. Wang and S. Belongie. Word spotting in the wild. In *Eur. Conf. on Computer Vision (ECCV)*, 2010.

[38] C. Yi and Y. Tian. Assistive text reading from complex background for blind persons. In *Proc. of Camera-based Document Analysis and Recognition (CBDAR)*, pages 15–28, 2011.