Abstract

This thesis considers the problem of automatically collecting semantic labels during robotic mapping by extending the mapping system to include text detection and recognition modules. In particular, it describes a system by which a SLAM-generated map of an office environment can be annotated with text labels such as room numbers and the names of office occupants. These labels are acquired automatically from signs posted on walls throughout a building using a feed-forward system that collects a set of building images; uses a trained classifier to detect candidate text regions on those images; runs optical character recognition on those regions to extract the textual content; and post-processes the text string set with alignment information before placing each string at its original location on the map.

Such a system faces difficulties using current text recognition techniques, as standard approaches to optical character recognition fail when faced with text from natural images as opposed to document text. Despite these difficulties, our system provides a series of additions to the typical mapping pipeline which provide practically useful results. In fact, our text detection and recognition system, combined with other ingredients, allow the robot to generate an annotated map from which it can recognize named locations specified by a user 84% of the time.
Acknowledgements

I first want to thank Andrew Ng for his classes, his mentorship, and just letting me hack in his lab. Undergraduate research has often been stressful, but he has made it a joy. Thanks also to Adam Coates for his guidance and ideas to bring this project to life and for pushing me through to its completion.

I particularly highlight my thanks to Bipin Suresh who really ought to be a co-author of this thesis. The work it describes was the result of an excellent collaboration with him. Frankly, the best technical work here is likely his. Any errors or confusions are, of course, my own.

I would never have gotten into this robot business were it not for the enthusiasm of Olga Russakovsky. She first brought me into Andrew’s lab and guided me through my first work on the STAIR robot.

This project would not have been possible without the generosity of Willow Garage in providing the lab with the PR2 robot and continued support and development of ROS. Thanks also to Morgan Quigley for his frontline help on all matters of hardware and robotics in general.

Thanks goes to the “AI crew” of the Class of 2011. Andrew, Beyang, Dan, and Micol: you guys made 228T problem sets bearable.

And my deepest thanks is to my family. Sarah, you are the best sister a boy could hope for. I know your college career will dwarf mine. Mom and Dad, your support over my four years in California has been wonderful and yet merely a continuation of all that which came before. Whether I listened or not, your advice has been unimpeachable. This thesis is for you.
# Contents

Abstract ii

Acknowledgements iii

1 Introduction 1

2 Background and Related Work 3

3 Overview of Application 6

4 Methods 9
   4.1 Map building, path planning and navigation 9
   4.2 Text detection and recognition 11
      4.2.1 Detection 11
      4.2.2 Recognition 13
   4.3 Map annotation and post-processing 15
   4.4 User Queries 17

5 Experiments and Results 18
   5.1 Hardware 18
   5.2 Evaluation 18

6 Conclusion 21

Bibliography 22
List of Tables

4.1  Precision and Recall for text detection ........................................... 13
4.2  Text Recognition Accuracy ............................................................. 14

5.1  Sign Reading Results — Environment 1 ........................................... 19
5.2  Sign Reading Results — Environment 2 ........................................... 20
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>The complete pipeline of proposed system.</td>
</tr>
<tr>
<td>4.1</td>
<td>Path planning process.</td>
</tr>
<tr>
<td>4.2</td>
<td>Text Detection.</td>
</tr>
<tr>
<td>4.3</td>
<td>PR curve for text detection.</td>
</tr>
<tr>
<td>4.4</td>
<td>Sample Text Patches</td>
</tr>
<tr>
<td>4.5</td>
<td>Two images of the same piece of text, the OCR result of reading the detected text, and the final resulting string created by merging them using a sequence alignment algorithm. See text for details.</td>
</tr>
<tr>
<td>5.1</td>
<td>Portion of map with typical annotations found by our system. The red arrows denote the position and normal vector of the detected signs. The image snippets show the detected text at each location, along with the OCR results.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

*I rarely use oxygen myself, sir. It promotes rust.*

– Robby the Robot, Forbidden Planet (1956)

Since our introduction to Robby the Robot and his dry wit in Fred Wilcox’ 1956 movie *Forbidden Planet*, the public imagination has held on to the dream of a true service robot that might liberate us from the tedium of quotidian chores. Researchers in Artificial Intelligence have held the same dream, though they also hold a healthy dose of scientific skepticism.

One of the core capabilities for any service robot (and one which Robby displays flawlessly) is autonomous mapping and navigation. Humans have developed systems of signage to help ourselves with this task. In an office environment, one of the natural homes of a service robot, we provide numbers for every room and place labels to identify those rooms’ occupants. Given the utility of this textual information for humans, it is surprising that current robotic mapping systems do not consider it. This makes life particularly difficult for the user of the robot, as he cannot say (or type) “Go to John’s office,” or “Get the papers in room 108.”

In this thesis, I describe a system to allow a robot to autonomously discover these place labels by reading the textual content of the signs around a building. I show that this text-extraction component makes a valuable addition to the standard mapping pipeline on modern robots. In particular, the completed map enables the robot to identify named
locations throughout the building with high reliability, thereby allowing it to satisfy user requests which refer to those places by name.

The system consists of several components which jointly accomplish the mapping task. First, the robot autonomously explores the building to collect static images of all the walls. Second, it detects and reads characters from these images that correspond to signs and placards. Finally, it attaches these character strings — themselves room annotations — to the robot’s internal map of the building.

The most challenging portion of the system is detecting and reading text from natural images the robot captures. Off-the-shelf components are available for optical character recognition (OCR), but their performance is generally poor when applied to non-document images. It is possible, however, to use these components — imperfections and all — in a complete end-to-end application with high performance. In the final system, the robot can respond successfully to 95 out of 113 user queries, far beyond the raw performance of the OCR systems themselves.

The remainder of this thesis proceeds as follows. I begin in Chapter 2 by discussing background and related work in robotic mapping, OCR, and semantic mapping specifically. Chapter 3 provides a bird’s-eye overview of the mapping application. Chapter 4 extends upon that to provide details underlying each component. In particular, Section 4.1 describes how the robot collects building images and Sections 4.2 and 4.3 present the methods for detecting, recognizing, and localizing text. Finally, Chapter 5 contains experimental results of the complete system. The performance of the system is assessed by measuring results for each algorithmic component in isolation and then measuring the performance of the full pipeline in both familiar and novel environments.

This thesis is based on [2] and portions of the text have previously appeared there.
Chapter 2

Background and Related Work

The problem of mapping and navigating within an unknown environment has been studied thoroughly in the robotics literature. The most common approach is to construct a “metric map” of free space [25]. This is usually a grid-like map in which each square corresponds to a fixed-size cell in the real world (say, 10cm x 10cm). Each cell on the map is marked as one of free, occupied, or unknown.

Given a map, the task of empowering a robot to autonomously locate itself on that map is known as localization. Usually, the map is constructed by allowing the robot to roam and find free and occupied space with a laser range finder. With those data, the robot must jointly construct the map and infer the path it took through that map to merge the laser scans into a single coherent plan. This problem is called simultaneous localization and mapping (or SLAM).

Having constructed a map and localized itself, the next task we expect of a robot is navigation. To navigate from one point to another, the robot will first plan a static path through free space to follow. As it begins to follow that path, it must choose appropriate dynamics (such as wheel velocities) to remain on the path and avoid obstacles [25].

Robots can now routinely build metric maps of a new environment [10] and navigate from one location to another reliably [6, 9]. Such functions are a part of ROS [20], the robot software package used on our robot. Thus the text-based “semantic” additions to the mapping pipeline described in this thesis function as extensions to these core mapping and navigation modules.
Prior work has shown that higher level semantic knowledge can be extracted directly from these highly accurate (usually grid-based) maps without any additional work. Examples include pinpointing the locations of doors and corridors, two concepts not present in the binary world of free and occupied space [1, 22].

There has been a good deal of recent research focused on this question of “semantic mapping.” One thread of such work has focused on creating linkages between spatial and geometric information (e.g., laser scans) and object class labels (such as “door” or “wall”) so that automated reasoning methods may be applied to those entities. The authors of [19], for instance, use laser scans to detect planar surfaces in the world, then classify those planes as walls, tables, ceilings, etc. using a set of simple constraints of the form “walls must be orthogonal to floors.” In [8] the authors extend this work by creating hierarchical spatial and topological maps and then establishing links between the two.

Other researchers have used various tools from machine learning to identify properties of spaces and recognize higher-level layouts. For example, [29] shows how a mobile robot can learn to classify regions of a 2D map as navigable/non-navigable or as sidewalk/street. Friedman et al. [7] present an algorithm based on conditional random fields and Voronoi diagrams to automatically extract the topological structure of a metric map and label every cell as room, hallway, door, or junction.

A handful of researchers have considered applications of text reading in robotic navigation. For instance, the authors of [16] present an edge-based text detection algorithm applicable to office or home environments. (They suggest it may be useful for robotic navigation but do not demonstrate such an application which would have required navigation, mapping, and text-recognition components.) In [17], an algorithm is presented to detect a set of known (pre-specified) landmarks where some include text such as room numbers, and the authors of [26] provide a model of their environment’s hallways and doors so that the robot can search for a given room by first matching the door model and then using it to locate the target text. In contrast to these last two, the system described herein has no prior knowledge of what rooms are present nor any model of the environment beyond very basic assumptions about the general locations of signs. Finally, the authors of [13] describe a robot that can read text from signs of known font, size, and background in a controlled lab environment (with no mapping component). Our system assumes nothing about the text
and depends on machine learning to handle novel or unusual environments.

Setting aside robotics research, the other core component of our system is text detection and recognition. The former consists of locating text within its surroundings. The latter involves translating an image of text characters to the corresponding text itself. Traditionally, optical character recognition (OCR) research focused on document OCR: translating scanned images of printed text back to the text itself. The Tesseract OCR engine [23] which our system uses is an example of one such document OCR system.

When one moves to natural images, the OCR problem becomes more difficult. First, detection is no longer straightforward, as many or most image elements are not text. Second, we can longer expect uniformity in size, font, or color of the text. Third, the images themselves introduce difficulties such as low resolution and partial occlusion of the text. Examples of prior work on OCR in images include [28, 5, 14, 27, 21].

The text detection problem is of particular importance in our system. Some early research used texture segmentation with heuristics to detect text in non-document settings [30]. Other work has applied machine learning as in [3] where a boosted classifier is trained to detect text in street images. [28] shows that generic object detection techniques (in this case, deformable parts models) are also applicable to “word spotting” — finding words from a fixed lexicon in a natural scene. The system described in this thesis uses a supervised learning method similar to that in [3], though other methods might also be applicable.
Chapter 3

Overview of Application

In the remainder of this thesis, I will develop a text-aware mapping system that fulfills the needs of a simple application: enable a user to direct the robot’s navigation by text queries containing room numbers or occupant names. For instance, when the user types “john smith,” the robot will locate the point on the map associated with this name (i.e., the location of John Smith’s nameplate). Thus, the system must construct a map and then annotate it with text data corresponding to room numbers and any names associated with rooms or offices.

In developing the system, I make two simplifying assumptions. First, I expect the environment to follow ADA guidelines with respect to room signage\(^1\). Most notably, signs must be posted five feet above the ground (modulo the size of the sign itself), thus limiting the areas that the robot must search for valid text data. Second, I separate the usual SLAM map-making task from the semantic mapping one; that is, I presuppose the existence of a navigable map for the target environment. This second assumption is made for simplicity of implementation: since the text modules do not use prior knowledge about the environment, they could reasonably run simultaneously with typical SLAM systems during map building.

Several components comprise the final end-to-end application (the full pipeline is shown in Figure 3.1). The first stage of the system plans a path through the environment’s free space, roughly following the walls. The robot executes this plan and stops every few feet to capture a photograph using a high-resolution camera. The path and photos must suffice

\(^1\)http://www.access-board.gov/adaag/html/adaag.htm
to cover the length of each wall which might contain room signage.

The second stage detects and reads text on each image captured during exploration. A machine learning-based classifier detects candidate text regions, then these regions are coalesced into bounding boxes (rectangles) around each line of text. An optical character recognition system reads each image patch contained by a bounding box and outputs the corresponding text.

Finally, these text outputs are collated and recorded on the map. Using depth data the robot captures during exploration, we can place the text precisely where it lies on the wall along with a normal vector representing the direction the sign faces. The final map with attached text data is saved for later queries by a user.

At query time, the system matches user requests like “john smith” or “room 108” to the map coordinates with the closest matching name or number. The robot can then navigate immediately to those coordinates without further direction from the user.

It is important to keep in mind the limitations of the system. In particular, it makes
no attempt to discriminate between incidental environment text and text that really is on a room sign. For example, a poster of the basketball player Michael Jordan with his name printed on is treated the same as a room sign for an actual occupant named Michael Jordan. This limitation is less restrictive than it may seem. After all, no user will request the name of someone not present, and the chance of “collisions” between random environment text and real names is low. Thus the system limitation does not percolate to the user level.

In the next chapter, I describe each of components in detail and evaluate each’s performance in isolation from the full system. I then move on to demonstrate the performance of the full application, evaluating its accuracy when associating user queries with their corresponding locations on the map. As explained in the acknowledgements, the system I describe is the result of collaboration with all of Bipin Suresh, Adam Coates, and Andrew Ng. Thus I describe the system as “our system” and say that “we did x, y, and z.”
Chapter 4

Methods

4.1 Map building, path planning and navigation

Our system is built on top of the ROS software framework [20]. In all experiments I describe, we began by manually driving the robot around the lab, and then built a world map using the standard GMapping SLAM toolkit [10]. The resulting map is used to plan a path from which to collect images of the walls. As mentioned in Chapter 3, we left this mapping portion external to our system for simplicity of implementation. There is no reason the two cannot autonomously run in sequence (or jointly) as part of a unified system.

The first stage of the pipeline is path planning. Its function is to establish a set of “way points” on the metric map of the lab where the robot will collect images of the walls. The points define the robot’s path during image collection, and they must satisfy three goals: (i) provide full coverage of the building, (ii) keep the robot a safe distance from the walls, and (iii) allow the robot to complete the task reasonably efficiently.

Given a grid-based map produced by the SLAM toolkit, the system performs the following steps:

1. Binarize the map using a fixed threshold, yielding a binary map of free space.

2. Erode the result by a fixed amount to eliminate positions that are too close to the walls. The added distance ensures that each image covers a reasonable area of the wall, while also making the path safer to follow.
3. Apply a Hough Transform [4] to find long straight lines in the eroded binary image. These correspond to sections of wall long enough to incorporate signs, culling out cluttered areas not likely to contain useful information.

4. Create a set of way points equally spaced along these lines. These are the locations where the robot will stop to take a picture of the walls. At each point, the robot orient its camera orthogonal to the line (and hence head-on to the nearby wall).

This process is illustrated in Figure 4.1, and the result is shown in Figure 4.1(c): we have a set of way points for the robot to follow through the map’s free space. Given these way points and a starting location, the robot greedily visits the nearest unvisited way point until none remain. At each, it captures two high resolution images facing along each perpendicular to the path’s line. It also captures depth data for each image with a structured light projector. The complete set of images captured are inputs to the next stage of text detection.

(a) Original image of the map  
(b) After erosion

(c) Way points along lines of Hough Transform

Figure 4.1: Path planning process.
CHAPTER 4. METHODS

4.2 Text detection and recognition

4.2.1 Detection

The output from the previous stage is a set images of the building’s walls. We must now locate the text on each of these images before OCR can read it. A significant body of work focuses on detecting text in natural scenes and video frames. For comprehensive surveys of text detection, see [14, 12]. Our system uses a logistic regression classifier with a variety of text features, similar to the approaches described in [15] and [3].

A logistic regression classifier is a function $f_\theta(x) : \mathbb{R}^n \rightarrow [0, 1]$ that takes $n$-dimensional input (say, a small patch of an image) and maps it to the range between zero and one. In particular, $f_\theta(x) \equiv g(\theta^T x)$ where $\theta$ is an $n$-dimensional parameter vector and $g(z)$ is the logistic function $\frac{1}{1+e^{-z}}$. We can interpret this as a binary classifier by assigning label ‘0’ to all $x$ such that $f_\theta(x) < t$ for some threshold $t$ and label ‘1’ to the rest.

In the case of text detection, we would like to classify patches of the image as text (‘1’) or non-text (‘0’). Our system extracts 10px by 10px patches from an image (striding 3px at a time), and each of these patches is classified with the same logistic regression classifier. Rather than use the 100 raw pixel values, we use several text features from the text detection literature: local variance, local edge density, and horizontal and vertical edge strength.\footnote{If $I$ is the image, $S$ is a circular disk filter, and $D$ is an edge filter like the Sobel filter, then local variance $V = S \ast (I - S \ast I)^2$; local edge strength $E = S \ast |D \ast I|$; and the horizontal and vertical edges are detected using Canny edge detection. We keep only edges shorter than a fixed threshold, as these tend to be associated with text.} The features provided to our classifier are the max, min, and average value of these features within the 10x10 window being classified. Thus, for each window we have a total of $4 \times 3 = 12$ features. The parameter vector $\theta$ (containing 12 weights) is trained from hand-labeled data.

Given a new image, the text detector steps through every 10x10 window and applies the classifier to obtain a score in $[0, 1]$ which we can interpret as the probability that window contains text. We apply a threshold as above to generate a binary image where ‘1’ valued windows are believed to contain text. These windows are coalesced using the connected components algorithm [11]. As a post-processing step, we reject coalesced regions whose areas are less than a predefined threshold ($\frac{1}{2000} \times $ImageArea in our experiments) and regions...
whose heights are greater than their widths. Finally, we place tight bounding boxes around the remaining connected components. The image patches bounded by these boxes are those passed on to OCR for recognition. Figure 4.2 shows example output for text detection. The first row highlights the patches on the image labeled to contain text. The second row shows the corresponding bounding boxes after post-processing.

![Figure 4.2: Text Detection.](image)

We can test the performance of text detection in isolation from the rest of the system. We collected images with the robot from two different buildings. The images exhibit variations in lighting conditions, font style, background, font size, and relative location of text within the image. We hand-annotated 23 images, from which we can extract 5,000 image patches containing text and 150,000 patches containing no text. These patches are used to train the logistic regression classifier.

We tested this classifier on a novel set of 80,000 patches from 13 new images. The classifier's performance on both train and test sets is listed in Table 4.2.1, and the precision / recall curve is given in Figure 4.3. (For the purposes of evaluating the classifier, extraneous text strings in the images that are not parts of signs are counted as negative instances even though at this stage of the pipeline there is no way to distinguish between these classes.)
The results are not, of course, perfect, but they are encouraging. In particular, note that recall is more vital at this point in the pipeline (given reasonable precision). We can correct for false detections later on, as OCR will see that the text is garbage. We cannot, however, back up later and find missed detections. Furthermore, detections missed at the patch level are often subsumed by a bounding box anyway.

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>0.65</td>
<td>0.73</td>
</tr>
<tr>
<td>Recall</td>
<td>0.86</td>
<td>0.80</td>
</tr>
</tbody>
</table>

Figure 4.3: PR curve for text detection.

4.2.2 Recognition

The text detection module outputs a set of image regions believed to contain textual information. The next step is to extract text strings from these regions. Given the candidate
CHAPTER 4. METHODS

Table 4.2: Text Recognition Accuracy

<table>
<thead>
<tr>
<th>Edit Dist.</th>
<th>Nameplate Data</th>
<th>ICDAR ’03</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>66%</td>
<td>59%</td>
</tr>
<tr>
<td>1</td>
<td>76%</td>
<td>66%</td>
</tr>
<tr>
<td>2</td>
<td>80%</td>
<td>72%</td>
</tr>
</tbody>
</table>

image regions, our approach to reading the text follows that of [3]: we binarize the image and then pass its output to an off-the-shelf OCR engine. In our experiments we use the freely available Tesseract [23] engine.

Given a candidate image region containing text, it is usually possible to separate the text from the background using a simple constant threshold across the image. This is because signs intended for humans are generally monotone text on monotone backgrounds (as opposed to posters and stylized logos). Due to lighting conditions and variation in color choices, however, a fixed binarization threshold for all regions often yields poor results that confuse the OCR engine. Instead, we compute multiple candidate binarizations (nine in our experiments) and run the OCR engine on each. We then choose to keep the resulting text string whose confidence score (output by Tesseract) is highest.

We tested the performance of our text recognition module using two datasets. The first is a dataset of our own construction consisting of 50 images of hand-cropped text from a wide variety of nameplates — a sampling is shown in Figure 4.4. We additionally test the text recognition on the ICDAR 2003 Robust Word Recognition dataset with 896 of its images\(^2\). Note that the latter dataset contains script, handwritten, and other non-standard text styles which we make no explicit effort to handle. The results are shown in Table 4.2. We also show how the accuracy changes as the number of errors (measured by edit distance) between the OCR result and the true string are allowed to increase. Thus, zero distance corresponds to the accuracy when requiring an exact match, and distance of one corresponds to the accuracy when allowing one “mistake”, and so on. We will see that counting partial matches in this way is valuable in our final application.

\(^2\)We use the Trial Test Dataset without non-ASCII and very low-resolution (height < 25px) images (removing 214), available at http://algoval.essex.ac.uk/icdar/Datasets.html
4.3 Map annotation and post-processing

At the end of the recognition phase, we have most of the text data we desire: a set of text strings associated with bounding boxes in images captured from various locations on the map. The final step is to place these text strings accurately on the map by using the depth and position data acquired during the mapping run.

Since the robot includes a textured light projector and a stereo camera pair, each captured visual image includes depth data precisely registered with that image. Thus, we can simply take the center of each text region and find its 3D position relative to the camera (mounted on the robot’s head). Using the robot’s localization system this point can then be transformed into the map coordinate frame and stored. In addition to recording the location of the text, we also include a surface normal vector computed by fitting a plane to the depth data in a small neighborhood around the text region.

This step is easy to carry out when each string of text is fully contained in exactly one image. However, text inevitably appears in multiple images and this occurrence can lead to errors in two ways. First and more commonly, a label will be present in two images and will be recognized as the same string both times. Second, it is possible for a name to be split at the border of two sequential images. In this case, we end up with partial overlap between the strings — for example, we may detect ”John Sn” and ”in Smith”. Thus, as a final post-processing stage, we combine these redundant bits of text by matching up parts.
of strings that occupy the same 3D location.

More specifically, for any string $s_i$ we define its neighbors $\{n_i^{(j)}\}$ to be all detections within a radius $r$ on the map (with $r = 50cm$ in our experiments). We then search for an optimal semi-global alignment of $s_i$ with each $n_i^{(j)}$. This is done using the Needleman-Wunsch sequence alignment algorithm [18], but with no penalties for gaps at the beginnings or ends of the strings.\(^3\) If the alignment score of the best match is above a threshold (in practice, 4 works well), we discard the two partial strings and replace $s_i$ with the optimal alignment of the two strings: We insert/delete characters as suggested by the alignment, and replace mismatched characters with whichever character is further from the end of its original string (as these characters tend to be more reliable). This process can be repeated on the remaining strings until no more merges are possible. The result is a reduced set

\(^3\)In our implementation, we use a score of 2 for matching characters, -1 for mismatched/inserted/deleted characters, and -2 for inserted/deleted characters at the ends of strings.
of strings where duplicates and partial strings have been combined to yield an improved
set of detections. See Figure 4.5 for an example of two strings merged together using this
process.

## 4.4 User Queries

In our final application a robot operator can enter a name or room number and the robot
navigates there without further instruction. During this process we must map the user’s
free-text query (“john smith”) to a localized detection. It is possible to require an exact
string match, but this requirement is inconvenient for the user who should be able to specify,
for example, only a first name as long as it is unique. Furthermore, such a requirement
degrades performance, as more intelligent string matching allows the system to recover
from minor OCR errors such as one mis-read character. (See Table 4.2.)

Since the user might enter a substring of the target, a global edit distance metric is
not appropriate. Instead, we compute a local sequence alignment score between the query
string and every detected string using the Smith-Waterman algorithm [24]. The string with
the greatest alignment score is used. Note that an exact match necessarily has the maximum
possible alignment score, so performance cannot degrade over using exact matches only.
Chapter 5

Experiments and Results

I now demonstrate the performance of the nameplate mapping application on the STanford AI Robot (STAIR) platform. I show that it can successfully detect and read the nameplates present in a novel environment with high accuracy which provides excellent performance for an end user.

5.1 Hardware

The robotic platform is the STAIR platform, which is based on the PR2 robot platform from Willow Garage [31]. A Prosilica GC2450C camera on the head takes the high-resolution (2448x2050) images for text detection and recognition. Also mounted on the head are a textured light projector and two wide-angle cameras (approximately 90° field of view) used to gather depth information.

5.2 Evaluation

To measure the performance of the application, we provide STAIR with a map of a novel environment from which no data was collected for classifier training. We run the entire system described above, the output of which is a set of strings (classified as room number or name), each associated with an \((x, y)\) coordinate on the provided map.
CHAPTER 5. EXPERIMENTS AND RESULTS

We can define two metrics of accuracy to measure performance. Let \( \{s^*_i\}_{i=1}^n \) be the \( n \) ground-truth text strings (names or numbers) present on nameplates in the given environment and let \( \{t_j\}_{j=1}^m \) be the \( m \) strings returned by our system. The first metric is string-level accuracy: for each \( s^*_i \), a correct value is a \( t_j \) in the correct location\(^1\) that matches \( s^*_i \) exactly (edit distance zero). The second metric is user-level accuracy: for each \( s^*_i \), a correct value is a \( t_j \) in the correct location such that \( s^*_i \) has maximal overlap with \( t_j \) (as described in Section 4.4) of all \( \{t_j\} \). This definition corresponds to the user’s entering each \( s^*_i \) in sequence and measuring the robot’s navigational accuracy.

To test the system in full we run the entire application in two different novel environments. For each of these environments we report separate results for room numbers and room names. Within these two categories we report the total number of detections, false positives and negatives, and the overall accuracy at both a string-level and user-level as defined above. For room numbers, any detection that is not text or not a room number counts as a false positive; for names, we count only non-text and detections that should have been room numbers. The accuracy numbers are a percentage of all the text in the environment — not only of detected text. The results for the two environments are in Tables 5.1 and 5.2.

Table 5.1: Sign Reading Results — Environment 1

<table>
<thead>
<tr>
<th></th>
<th>Room Numbers</th>
<th>Room Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Env. Total</td>
<td>30</td>
<td>41</td>
</tr>
<tr>
<td>Total Detections</td>
<td>30</td>
<td>147</td>
</tr>
<tr>
<td>False Positives</td>
<td>5</td>
<td>58</td>
</tr>
<tr>
<td>False Negatives</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>String-level Accuracy</td>
<td>50%</td>
<td>71%</td>
</tr>
<tr>
<td>User-level Accuracy</td>
<td>77%</td>
<td>93%</td>
</tr>
</tbody>
</table>

We see that while the low-level measures of accuracy are relatively poor, we nevertheless associate a significant number of correct strings with their correct locations on the map. More importantly, from the point of view of a user, queries are mapped to the correct locations in the great majority of cases in both buildings (since exact matches are not necessary.

\(^1\)A correct placement is defined as one within 50cm of the coordinates given on a hand-labeled map
Table 5.2: Sign Reading Results — Environment 2

<table>
<thead>
<tr>
<th>Room Numbers</th>
<th>Room Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Env. Total</td>
<td>19</td>
</tr>
<tr>
<td>Total Detections</td>
<td>21</td>
</tr>
<tr>
<td>False Positives</td>
<td>4</td>
</tr>
<tr>
<td>False Negatives</td>
<td>2</td>
</tr>
<tr>
<td>String-level Accuracy</td>
<td>74%</td>
</tr>
<tr>
<td>User-level Accuracy</td>
<td>89%</td>
</tr>
</tbody>
</table>

to get the correct destination). A portion of the map from the first building showing several typical detections and their locations (as recorded by our system) are shown in Figure 5.1.

Figure 5.1: Portion of map with typical annotations found by our system. The red arrows denote the position and normal vector of the detected signs. The image snippets show the detected text at each location, along with the OCR results.
Chapter 6

Conclusion

This thesis presented a system that automatically collects semantic labels associated with places on a map by reading them from signs posted on walls in the environment. The output of the system is a map annotated by dozens of accurate text labels suitable for use in various applications.

The system accomplishes this task by subdividing it into a sequence of modules and operating in a feed-forward fashion. The robot collects a set of wall images; uses a trained classifier to detect candidate text regions on those images; runs OCR on those regions to extract the textual content; and post-processes the text string set with alignment information before placing each string at its original location on the map. A user can then query locations by name or number.

The problem of detecting and reading the text from natural images remains difficult and is ripe for future research. However, our system shows that a careful construction of imperfect textual components is already a valuable addition to the standard mapping pipeline, as it provides high accuracy to the user’s queries in the face of low-level errors.

That we can achieve these results using off-the-shelf OCR software and virtually no assumptions about language structure or spatial layout of the text is a strong positive result and suggests that even better results are readily accessible in the future by improving the basic components laid out in our work.
Bibliography


