SAM: Semantic Autonomous Mapping

Mateo Guaman Castro, Chris Mitsopoulos, Brandon Chung

Abstract

This project seeks to create an autonomous robot that independently circumnavigates a certain region of Halligan Labs, reads the door badges employing an external camera, and then utilizes computer vision technologies to semantically identify each of the rooms the badges are associated with.The purpose of this project is not only to generate a functional semantic autonomous mapping, but to also explore how exactly autonomous robots can act more intelligently through simiulated human perception.

1 Introduction

Semantic identification is one of the most fundamental byproducts of human perception. This natural ability to distinguish area A from area B is particularly important when considering human perception in regards to interpersonal communications. More importantly, this nuanced comprehension of area and space enables humans to more readily navigate certain locales despite a dearth of pre-information.

Although these natural abilities seem rather trivial to any functioning human, imbuing this advanced perception onto robots has historically proven to be quite a challenge. After all, robots cannot inherently detect nor infer these aforementioned nuance or subtleties in any given environment. For example, consider the task of finding "Room 10" in a hallway that possesses ten doors where each door hangs a badge identifying itself as a room from 1 to 10. To humans, this concept is rather elementary; the person in question would merely have to navigate to each door until finding the room with the appropriate badge that reads "Room 10". A robot, on the other hand, would not be able to distinguish "Room 1" from "Room 10" because it does not inherently understand these largely human social constructs. In fact, a robot wouldn't even be able to distinguish a wall from a door without some predetermined distinction indicator.

Herein lies our proposition: what if we were able to create a robot that navigated through a given locale and would be able to label certain locations with salient information independently (similar to that of a normal human)?

By doing so, we could potentially open a communications channel between human and robot such that our commands did not have to be exclusively inputted as arbitrary data or numbers. Instead, a human could relay commands to a robot in a way to similar how a human would express those commands to another person.

Furthermore, accomplishing this particular project would open doors to a number of more complex operations that could be executed as a result. For instance, a fully autonomous mail delivery robot could detect which door in Halligan it would need to approach without needing exact coordinate locations; rather, the robot would be able to simply semantically apply labels on certain locations and independently determine its goals based on a sparse command. By and large, this project serves to make inroads into smarter autonomous design that simulates human movement and perception — a foundational step in ultimately understanding how robots can begin to interact more intelligently in a way that fuses robot and human perception.

2 Related Work

Below are a number of related works that both inspired as well as aided in the development of our own project:

2.1 Semantic Mapping

The project described in [1] by Sun, Hao, et al chose to further explore a lot of the dynamism in robotic semantic perception. In particular, this project sought to understand how recognizing salient data in real time could be both smarter and faster if implemented in a way such that the robot could adapt its knowledge to a wider array of objects and items.
The work described in [2] works almost as an extension of the previous article; however, this particular paper seeks to further explore the mutability of objects. In other words, the robot could more intelligently identify objects based on certain object classifications that the robot could store in its own memory. In short, a robot could readily identify two chairs as "chairs" even if the two were aesthetically very different.

2.2 Computer Vision

The computer vision project described in [3] by Marroquin and Dubois outlined how semantic mapping and computer vision could work in tandem to provide robots with a deeper understanding of simulated human perception. In short, this article explained what computer vision is, how it works, and, most importantly, how this function could be used to empower robots for more autonomous design.

3 Methodology

The problem of navigating a map and autonomously creating a semantic map was divided into three parts: autonomous navigation, sign detection, and semantic layer creation.

3.1 Problem Formulation

The navigation problem was motivated by wanting to visit the largest number of points in a map, in order to get a semantic layer of as much of the map as possible. In this case, the problem was approached using a series of way points strategically placed to guide the robot through as much of the map as possible. In the case of this project, the map used was the map of the second floor of Halligan Hall, the ECE and CS departments building. This map has the feature that it has long hallways and three common collaboration areas. These collaboration areas contain little to no signs to be detected, so in the interest of time, the only areas that were navigated by the robot were the long hallways.
Two common features to all the door signs in Halligan Hall are that they are of a specific blueish-green color and that they are all rectangular. This commonality was taken advantage of to detect the signs autonomously. It provided a way to check if a specific object in a frame was a sign using a two-step process: detecting color, and then detecting shape. Optical Character Recognition was then used to extract the text from an image containing a sign.
The semantic layer creation was done by generating a YAML file that would include all the information necessary to later create markers for each point on RVIZ. These markers could then be overlaid on top of the 2D coordinate map for the floor.

3.2 Technical Approach

The entire system ran on ROS[1] packages and on the Turtlebot platform.

3.2.1 Navigation

For navigation, we used ROS Actionlib and move_base packages in order to generate move actions. These actions require a destination pose, and they provide feedback as to what the current status of the goal is. Additionally, they allow actions to be canceled. All of these features made actions appealing as a way to navigate the map.
A series of way points were obtained from the existing map, and they were added to a queue. These way points were selected such that the robot would move along a corridor staying close to the wall to the right of the robot, so that the camera could pick up door signs with more ease. These

[1] www.ros.org

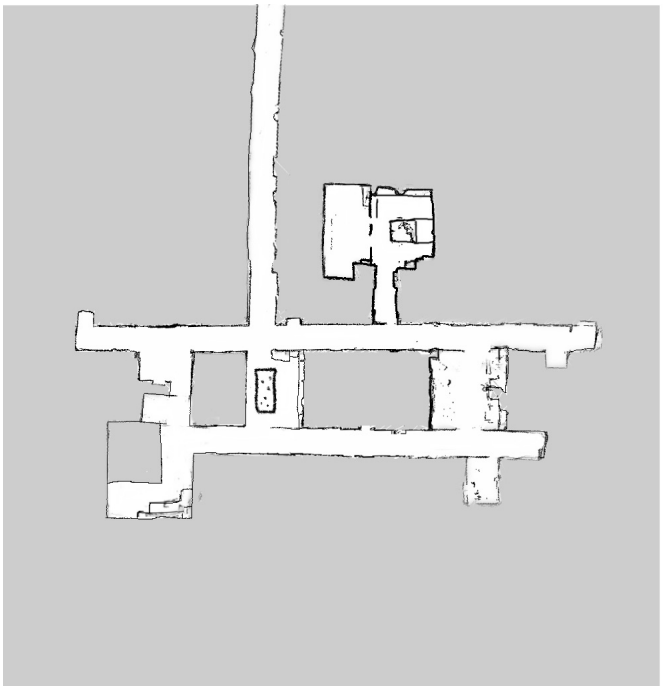


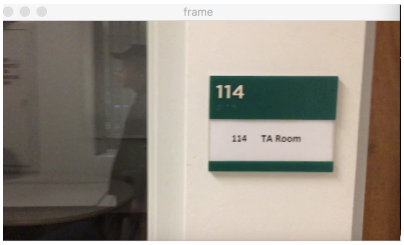
Figure 1: Map of the floor to be navigated

way points were chosen along two of the main corridors of the second floor, as they contained the largest number of door signs. As the robot reached each goal, the new goal would be sent, making the robot continuously navigate the map. Whenever a sign was found, the x and y coordinates of the robot in the map frame of reference were published alongside the sign information in order to keep track of what the location of a certain sign was.

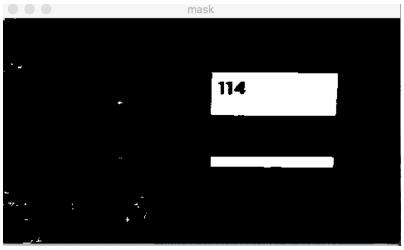
3.2.2 Sign Detection

Sign detection was divided in three parts: color recognition, shape recognition, and optical character recognition. All of the computer vision algorithms and methods used in this project used the OpenCV package[2].
The first step in detecting signs was to perform color thresholding on each frame of a video feed. The values for the color thresholding were chosen empirically, knowing that all the signs would have the same color between green and blue. This thresholding was done using a function called inRange() in the OpenCV package. This function generated a binary mask that showed white pixels where the original frame had pixels of this greenish color, and black pixels for any other color. This process is shown in figure 2.
This mask was then used to detect shapes of interest in the binary mask generated. This method relies heavily on the color thresholding part of the system, since it looks for shapes of interest in the binary mask and not in the original frame. Several different methods were tested to try to detect rectangles in the frame. Some of these were Canny edges, Hough transform and Hough lines, and Harris corner detection. However, the easiest method in the end was the use of OpenCV's findContours() function. This function returns the number of contours for a single object, for all objects in the frame. These contours can then be counted in order to find an object with four contour lines. Then, another mask could be generated made of only rectangular objects of the desired color. Figure 3 shows the color-shape mask alongside the original frame, with red rectangles overlaid on top of what the algorithm believes is a frame.
Finally, this second mask was filtered to reduce the amount of noise in the frame. The black and white mask was eroded in order to remove the individual pixels that did not make up signs, which were the only objects of interest. This was done since, later in the process, the area of the white section of the binary mask would be used, and the noise pixels could affect the area detected. The final frame, which was used to determine whether an object was a sign or not, looked like the one

[2] https://opencv.org



(a) Original frame



(b) Binary mask after color thresholding

Figure 2: Color thresholding

(a) Mask based on color and shape

(b) Mask overlaid onto original frame

Figure 3: Shape detection

in figure 4.

It can be seen from figure 4 that a simple threshold can be set in order to determine whether a frame contains a sign or not by using the area of the white region of the frame. Assuming that the size of the sign in the frame will always be similar to the one used in the test picture shown in figure 4, the total number of white pixels in the frame can be counted and, if the ratio of this area to the area of the whole frame is greater than a certain threshold, then the robot can safely assume that it found a sign. In the case of this project, the threshold was set to be low, since it was determined that most of the frames collected where a sign was detected would have a high confidence.

While this process detected whenever there was a big enough object in the frame with the right characteristics to be determined a sign, there were a few challenges that needed to be taken into account when the source of the frames was a video feed. Since the video feed would capture the same sign for multiple frames, there were a few conditions that needed to be checked for when detecting a sign. First of all, the algorithm would wait for 0.2 seconds before capturing the first frame of the sign, to get a more centered frame. After waiting, one enable variable would be set off whenever the first frame was received. This enable variable would be set to true again when there were no more white pixels left on the frame. This would make the algorithm detect only one

Figure 4: Final filtered frame

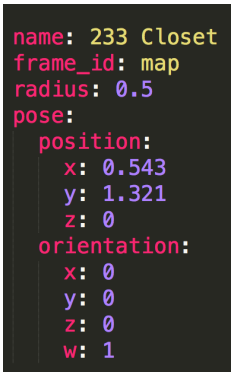


Figure 5: Desired YAML file

frame for the same sign. Because the code would sometimes jitter when it detected the sign, a timer was set so that if, for any reason, the code stopped detecting a sign while it was still in the field of vision, and then detected it again, if the timer was below a certain value, the code would not recognize this frame as a new sign. Instead, it would recognize that since not enough time had elapsed between two signs, it was still probably detecting the same sign.
After obtaining a frame with a door sign in it, the picture was transformed into a base64 encoded string. This string was used to send an HTTP POST request to the Google Cloud Vision server, which has a great OCR system. The server returned a JSON string containing a lot of information from the image sent, such as text identified, bounding boxes in the frame, confidence levels, and other parameters.

3.2.3 Semantic Layer

After the information was received, it was important to display only the useful parts of what was returned by the Google Cloud Vision server in a way that could then be used to display on RVIZ. The desired format looked like Figure 5. This was obtained by converting the JSON string into a Python dictionary and accessing only the relevant values of the dictionary using standard Python methods. Then, the relevant information would be written into a YAML file with the most current information.

4 Experiments and Results

4.1 Example Run

In order to have a measure the performance of the system, the robot was given way points along one of the corridors next to the robotics laboratory. Three separate ROS nodes created for this software were running in parallel: the movement node with the way points, the sign identification node, and the YAML generator. The robot successfully navigated through the given way points. The robot went past seven different signs, and it recognized all of the signs properly. However, it only stored the frames for four different signs, and it mistakenly stored the same frame An example of the frames captured by the robot can be seen in Figure 6. One of the frames recognized was repeated, but all of the other frames stored were stored and taken into account only once. This showed that the algorithms used to make sure that each sign was only captured once worked properly.
The robot stored five frames for four different signs, but it only stored the information for three different frames as well as the repeated frame in the YAML file after the parsing was done. Parsing the frames obtained allowed to successfully retrieve only the most relevant information returned by the Google API. This information was either the most significant word if there were not any numbers in the sign, or just the room number if there was a number associated with the sign. Figure 7 shows an example of the YAML file generated.

(a) Example of frame captured

(b) Example of frame captured

Figure 6: Frames autonomously captured by robot

4.2 Quantitative Results

4.2.1 Navigation

In terms of navigation, the robot completed its path 100% of the times, although there were two to three points in every run where the robot would get lost for about twenty seconds and then recover its path. It is possible that this was caused as a result of the map being imperfect, or some objects around those certain areas having been moved around, for example, trash cans.

4.2.2 Sign Detection

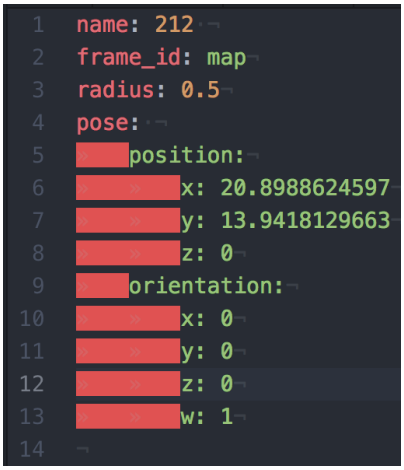
The results for the sign detection depended on whether the sign detection was done on the robot or on a computer. When running the algorithms on a computer external to the robot, the algorithm successfully recognized 100% of the signs in the video feed. In terms of the Google API, about 70% of the frames sent returned a JSON file with useful information about the sign in the frame. In some occasions, the Google API failed to recognize part of the information.It was easier for the software to recognize signs that were completely green and had only numbers, but it still recognized most signs without much issue. When running the code on the Turtlebot, the robot identified successfully around 60% of the signs it encountered, and from this 60%, about 70% were parsed correctly into the YAML file.

4.2.3 Semantic Layer

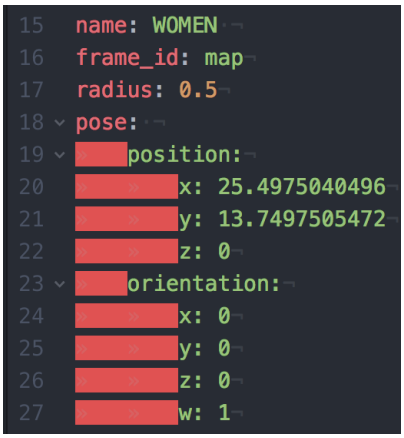
As for the semantic layer, the software put all of the frames captured into a YAML file. The software took care of correcting the issue were the character "B" was represented as the digit 8. For signs that contained no numeric characters, the most relevant word was displayed properly, such as "WOMEN" for the women's restroom. One issue that was not taken into account was handling repeated entries for the same sign, since it was assumed that the sign identification part of the software would take care of it.

4.3 Discussion

Overall, the results were satisfactory for a perception autonomous system. More than half the signs were recognized properly in a single run, which should mean that in multiple runs, the system should be able to identify most of the signs in a certain floor. Some additional safeguards could be implemented in order to make sure that only one entry per sign is added to the YAML file. The sign identification software worked in most cases, which was one strength of the software, however there were also a few weaknesses. One weakness was that, while the signs were recognized every time, they were not properly processed by the Google API, which could be a result of the HTTP requests timing out or the on board computer requiring more processing power. Another issue was that the captured frames did not have a good resolution, which could have impacted the number of signs recognized. This might have been caused by the lack of a proper support for the camera or by the robot moving too fast.



(a) Example of YAML entry



(b) Example of YAML entry

Figure 7: Examples of YAML Entries

5 Conclusion and Future Work

We introduced a method for a Turtlebot to navigate through a given locale and autonomously label locations with salient information. This allowed us to create a secondary layer of information on top of the original two-dimensional map. One limitation of our method was that the only source of this information comes from the door signs the robot identifies. In future work, we plan to enrich the semantic map by evolving the capabilities of the robot to identify other objects such as chairs, tables and other objects in space. We also hope to improve the accuracy of optical character recognition as well as implementing the sign detection locally.

References

[1] Sun, Hao, et al. "Semantic Mapping and Semantics-Boosted Navigation with Path Creation on a Mobile Robot." ieeexplore-ieee-org.ezproxy.library.tufts.edu/document/8274775/.

[2] Sünderhauf, Niko, et al. "Meaningful Maps with Object-Oriented Semantic Mapping." ieeexplore-ieee-org.ezproxy.library.tufts.edu/document/8206392/authors.

[3] Marroquin, Roberto, and Julien Dubois. "Know Beyond Seeing: Combining Computer Vision with Semantic Reasoning." ieeexplore-ieee-org.ezproxy.library.tufts.edu/document/8334484/authors.