ABSTRACT

SERVICE ROBOT FOR THE VISUALLY IMPAIRED: PROVIDING NAVIGATIONAL ASSISTANCE USING DEEP LEARNING

by Amlaan Shakeel

Assistive technology helps improve the day to day activities for people with disabilities. One of the methods utilized by assistive technologists employs the use of robots. These are called service robots. This thesis explores the idea of a service robot for the visually impaired to assist with navigation and is inspired by the use of guide dogs. The focus of this thesis is to develop a robot to achieve autonomous indoor navigation using computer vision to identify image based goals in an unfamiliar environment. The method presented in this thesis utilizes a deep learning framework, called Faster R-CNN, to train a computer to classify and localize exit signs in real time. A proof of concept is presented using NVIDIA Jetson, and TurtleBot, a robot kit, which runs a robot software development framework Robot Operating System (ROS). The model is trained successfully using Faster R-CNN and is validated. The model is used for real-time object classification on the prototype robot.
SERVICE ROBOT FOR THE VISUALLY IMPAIRED: PROVIDING NAVIGATIONAL ASSISTANCE USING DEEP LEARNING

Thesis

Submitted to the
Faculty of Miami University
in partial fulfillment of
the requirements for the degree of
Master of Science in Computational Electrical and Computer Engineering
by
Amlaan Shakeel
Miami University
Oxford, Ohio
2017

Advisor: Dr. Yamuna Rajasekhar
Reader: Dr. John Femiani
Reader: Dr. Donald Ucci

©2017 Amlaan Shakeel
This thesis titled

SERVICE ROBOT FOR THE VISUALLY IMPAIRED: PROVIDING NAVIGATIONAL ASSISTANCE USING DEEP LEARNING

by

Amlaan Shakeel

has been approved for publication by

College of Engineering and Computing

and

Department of Electrical and Computer Engineering

________________________

Dr. Yamuna Rajasekhar

________________________

Dr. John Femiani

________________________

Dr. Donald Ucci
## Contents

1 Introduction .................................................. 1
   1.1 Motivation and Objectives ........................................ 1
   1.2 Statement of Originality ........................................... 2
   1.3 Contributions ................................................... 3

2 Literature Review ........................................... 4
   2.1 Passive Assistance .............................................. 4
   2.2 Service Robots ................................................ 5
   2.3 Indoor Navigation .............................................. 5
   2.4 Signage Detection .............................................. 6

3 Background .................................................. 7
   3.1 Robot Operating System (ROS) ..................................... 7
   3.2 Simultaneous Localization And Mapping (SLAM) ............... 8
   3.3 TurtleBot ..................................................... 9
   3.4 Depth Perception ............................................... 10
   3.5 Deep learning in Computer Vision .............................. 12
   3.6 Controller Board ............................................... 14

4 Objectives .................................................. 16
   4.1 Training and model validation ................................... 16
   4.2 Prototyping .................................................. 17
List of Tables

3.1 Controller board specification comparison ........................................ 15
5.1 Classes in dataset with examples ...................................................... 20
6.1 Confusion matrix .............................................................................. 30
6.2 Results of model validation ............................................................... 32
6.3 Comparison against existing methods ................................................. 35
6.4 Execution time vs. number of region proposals (data averaged over 10 consecutive frames) .............................................................. 37
6.5 Electrical measurements for Jetson TX1 board .................................... 39
# List of Figures

3.1 Simplified stereo vision system ............................................. 10
3.2 Disparity image computed using stereo images ................................. 12
3.3 R-CNN: Regions with CNN features ............................................. 13

5.1 Computer vision pipeline using Faster R-CNN .................................... 21
5.2 ROS Navigational Stack Setup .................................................. 22
5.3 Navigational controller flow ....................................................... 24
5.4 Initial setup with TurtleBot and Raspberry Pi .................................. 25
5.5 Current robotic setup ............................................................... 26

6.1 Path visualization of TurtleBot using Rviz ..................................... 28
6.2 Object classification using Caffe .................................................. 29
6.3 Example graphic for estimation of Intersection over Union (IoU) *Manually annotated for purpose of example* .............................................. 31
6.4 Sample PR curves for some object classes ...................................... 32
6.5 Selected image examples of object detection results from R-CNN trained model for the eight object classes .................................................. 33
6.6 Sample false positive detections .................................................... 33
6.7 Sample output from pipeline running on prototype robot (using Kinect RGB camera) ................................................................. 36
6.8 Result of mapping bounding box coordinates from HD camera to Kinect camera ................................................................. 36
Acknowledgements

I would like to thank my adviser Dr. Yamuna Rajasekhar for her encouragement and support during the completion of this thesis. I am grateful for her willingness to invest in all the equipment I required. And above all, I am thankful for her belief in me and my research idea.

A special thanks to Dr. John Femiani for his technical support and his precious time. He not only provided me with a robotic development kit but also assisted me with the process of machine learning for this thesis. He was always ready to help and encourage me.

Additionally, I would like to acknowledge my committee member Dr. Donald Ucci for his support and the constructive criticism during my proposal phase that allowed me to improve the quality of content in this thesis.

A big thank you to my friends and family, for being a source of encouragement, especially to Yosuf Elsaadany, Julius Higiro, Yeshwanth Venkatesh and Divyalakshmi Soundararajan who helped me with data collection. A special thanks to Julius Higiro for introducing me to Dr. Femiani’s work sphere at a crucial juncture of my research.
Chapter 1

Introduction

Since the discovery of fire and the invention of the wheel, technology has been responsible for dramatically improving human lives. Depending on the needs of humankind, technology has evolved and branched into categories that are focused on different objectives. One such category is assistive technology that focuses on improving the independence of people with disabilities, and creates new ways for their interaction with the environment. Some examples of the most successful and commonly available assistive technologies are the electronic hearing aid, the braille system and the white cane.

1.1 Motivation and Objectives

For the visually impaired, research into assistive technologies has led to the making of the white cane. Additionally, several attempts to create a better navigational experience have been made in recent times. Ultrasound sensors and vibro-tactile feedback have been used to make the traditional white cane a “smart” cane. A commercial version of this idea is the Ultracane by Sound Foresight Technology Ltd. Some researchers have tested the idea of addition of more smart features such as wireless connection to a smartphone that provides useful information through an audio interface.

Another novel approach is to embed object avoidance systems into a wearable device, rather than a white cane, in order to provide a better experience to explore indoor spaces. An example
of this is the Project BLAID by Toyota [2], which includes identification of useful signage, stairs, elevators, etc, with audio feedback.

In comparison to the traditional white cane, guide dogs have been found to be more successful in increasing the mobility and independence of blind people [3]. The increased ease of mobility is simply because the guide dogs can see the path ahead of them, and steer a person in the right direction, especially in an unfamiliar place. This leads to idea of a robotic assistant that potentially can do the same critical things, i.e., see the surroundings to determine a path while avoiding obstacles and steer a person through its own motion.

Existing literature [4–6] in robotic assistance is targeted more towards people with physical disabilities. These robots are designed to detect and correct the posture of the person, and in some cases provide directional force when the person is likely to fall. The idea of robotic assistance for the visually impaired, however, seems rather limited. This is because such a robot would need to be capable of autonomous navigation in an unfamiliar environment. The technologies required to achieve autonomy, such as computer vision and depth perception, are relatively young and have recently become more accessible due to advances in deep learning. One of the key challenges is indoor navigation, which is denied navigation technologies such as Global Positioning System (GPS), but often relies on computer vision and depth perception.

1.2 Statement of Originality

This thesis presents proof-of-concept of a robot capable of autonomous indoor navigation in an unfamiliar environment using computer vision to identify navigational goals such as street signs, walkways, etc. Most robots capable of indoor navigation either require a pre-mapped environment [5] or have been programmed to patrol randomly in an unknown environment while avoiding obstacles [6]. This thesis presents a method that makes use of very recent implementations of neural networks and region proposals in computer vision to classify and localize image based goal. Additionally, an algorithm for a navigational controller has been developed that eliminates aimless wandering of the robot in an unfamiliar environment. These features are essential to achieve a truly autonomous and self reliant robot.
1.3 Contributions

The main contribution of this thesis is a proof-of-concept autonomous robot that is ultimately capable of guiding a visually impaired person by “tugging” them in the right direction. A test case is proposed to validate the robot which involves making it follow exit signs in an unfamiliar indoor environment until it reaches an exit door or an elevator. In order to achieve this, there are several stages. These stages are:

- Compilation of images of exit signs, doors and elevators into a annotated dataset.
- Training a CNN model to classify exit signs.
- Validation of the trained model.
- Developing an algorithm for navigational controller that would follow image based goals.
- Performance analysis, including the following:
  - Measurement of the success rate of object identification and localization
  - Evaluation of the effective real-time frame rate
  - Analysis of the accuracy of depth perception
  - Estimation of power consumption
  - Investigation of the effect on performance with variation in computer vision parameters
Chapter 2

Literature Review

This chapter discusses the literature related to the work presented in this thesis. Publications are categorized into four sections, called passive assistance, service robots, indoor navigation, and signage detection based on their target application.

2.1 Passive Assistance

Several researchers have looked into enhancing the experience of the visually impaired by adding sensors and feedback to the traditional white cane and other wearable devices. Since these systems rely on the user to take action based on feedback, these systems can be classified as passive assistance. In [7], vibrotactile and audio feedback are provided for obstacles approaching the user. Object motion is detected via an ultrasonic range finder and the intensity of vibration increases with the object’s speed of approach. Work done in [8] is a similar example of a passive assistance cane. The method presented in this thesis differs because it provides active assistance; in addition to detecting and avoiding obstacles using a range finder, the prototype robot also moves the user towards a goal (exiting a building).
2.2 Service Robots

A system that can react to sensor feedback with some nature of motion can be classified as active assistance. Since these systems usually make use of robotic parts, they may be termed as service robots.

The guide robot presented in [9] is capable of tugging the user while avoiding obstacles using laser range sensors and a depth camera. The velocity of the robot is controlled by a force sensor on the handle. The user can estimate the direction of travel by the pose of the robot. The paper focuses on making the movement user-friendly by making the motion of robot as smooth as possible. One limitation is that the robot does not have any navigational goals.

Another work [6] presents a design for a social assistive robot and a proof-of-concept using a robotic setup consisting of a mobile base, Kinect sensor, laser range finder, Sound Navigation And Ranging (SONAR) and a Pan-Tilt-Zoom (PTZ) camera. The prototype robot was capable of patrolling indoor environments by wall following and obstacle avoidance. Simple colored markers were used to identify end goals. Although the design mentions the use of artificial neural networks to identify objects, signs and faces, it does not give any further details about its implementation.

The co-robotic cane [5] is an example of active assistance that uses computer vision for pose estimation and object detection. A servo is attached to the bottom of the cane that can steer the user in a certain direction. The limitation of this device is that it requires a floor plan for navigation. This thesis aims to use indoor signs posted in the building in order to navigate towards an exit, in much the same way that a sighted person might navigate in an unfamiliar environment.

2.3 Indoor Navigation

Indoor navigation has been investigated by G. Adorni et al. [10] using shape markers and signs. The markers were identified with the help of neural networks and a black and white Charge-Coupled Device (CCD) camera. The neural network was trained with 200 images for each shape class, along with some negative samples. The approach presented in this thesis detects signs that already exist in an environment, rather than specially placed markers developed for the robot.
The authors in [11] explore the idea to make a modified Parrot drone to follow exit signs in a hallway. They attempt to determine the location of an exit sign in the camera image using a color threshold. A bounding box is made around areas with the right threshold, that is then fed into an Support Vector Machine (SVM). The SVM was trained with 150 images of exit signs and 150 images of empty hallways. According to their experimental results, their approach was very sensitive to color threshold. Exit signs are often red or green, may exist on a variety of backgrounds, and may or may not include the word *exit*, but this approach was aimed only to detect red signs. In addition, the approach in this thesis not only aims to detect exit signs, but also determine the direction indicated by the exit sign.

### 2.4 Signage Detection

Wang et al. [12] proposed a method to detect indoor signage to help blind people find their destinations in unfamiliar environments. Their method uses a saliency maps [13, 14] extracted from images by using intensity, color, and edge orientations to reduce search space for template matching to recognize signs. The recognition is done using query pattern of their symbols. The method for image recognition in this thesis does not use such templates to identify exit signs but rather aims to detect broad classes of exit signs. Experimental results are compared with this paper in the chapter 6.
Chapter 3

Background

This chapter explains concepts required to fully understand this work. First, we describe the concepts of robotics used in this work, namely, Robot Operating System (ROS) and Simultaneous Localization And Mapping (SLAM). Then we move on to the TurtleBot, which is a vendor development kit that has been modified to suit our purpose. Lastly, we explain the concepts of computer vision used in this thesis, including depth perception, deep learning in computer vision. An overview of the controller board that runs the turtlebot, and choices considered, is also included.

If the reader is familiar with these concepts, it is advised to proceed to chapter 4.

3.1 Robot Operating System (ROS)

ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [15].

The ROS is being adopted widely as the preferred framework by developers for research in robotics and now supports a few industrial robots as well. There are several advantages that explain its success and is therefore the choice for this thesis. These are:

- Inclusion of drivers and libraries for many off-the-shelf robotic kits
- Integration of tools for debugging, 3D visualization and simulation
• Support for processes written in python and C++

• Modularity allowing multiple processes to run, hence a crash in a single process does not force the entire system to crash.

• Existence of an active community of developers contributing to expand capabilities and adding support for new equipment.

A basic unit in ROS is called a **package**. It contains nodes, libraries, configuration files, etc. Several nodes may run simultaneously and may even communicate with each other. These communications occur via messages, topics, and services.

Messages are data structures that hold any kind of data and are transported via topics. Nodes may send messages via topics (publish) or receive messages (subscribe). Nodes that subscribe to topics need not be aware of the publishing node. The advantage of the publish/subscribe system is that many nodes can simultaneously subscribe to topics. An example for a ROS topic is ‘/camera/rgb’, which give an RGB image from the camera.

For some models the publish/subscribe system is not sufficient as it is a one-way kind of communication. A request/response based system, called **ROS services**, is more useful in such cases. Nodes can create a service under a name, making them a **server** node, while a node that sends a request is called a **client** node. To establish communication a client node sends a request to a server node, which sends back a response.

There have been several distributions of **ROS** since its availability. The distribution used throughout this thesis is the Kinetic distribution, which was released in May, 2016.

### 3.2 Simultaneous Localization And Mapping (SLAM)

**SLAM** as the name suggests, is the process of constructing a map of an unknown environment while also estimating the location and pose of a robot within it. This is very common and useful for autonomy in machines such as autonomous robots, self-driving cars, planetary rovers, etc.

Mapping is performed using information from sensors such as stereoscopic cameras, laser range finders, ultrasonic range finders and Light Detection And Ranging (LiDAR). All these
sensors typically give depths or distances to everything around a robot. Objects have to be closer
than a sensor’s operating range to be detected.

Localization is typically done using data from odometers, accelerometers, or GPS modules. These sensors can give an estimate of how far a robot has moved along with its pose. Accuracy depends on the sensor. For example, GPS is useful only for outdoor localization and has an accuracy of about 10 meters. Odometer estimates the distance traveled based on the rotation of the wheels, which has better accuracy than GPS for a relative distance measurement with respect to the initial position. However, accuracy may vary depending on the surface, as wheels tend to slip several times on a smooth floor.

SLAM uses many of these sensors to build a map while localizing the robot with respect to a virtual frame. Localization helps expand built map as the robot moves around, since distances are always obtained relative to the robot, and then points are computed relative to a global frame.

3.3 TurtleBot

The TurtleBot is a development robotic kit consisting of a mobile base and a 3D camera sensor. In this thesis the TurtleBot used came with a Kokuki mobile base and a Microsoft Kinect sensor.

The advantage of using TurtleBot is that it is well integrated with ROS which contains many useful packages and examples. Using a ROS package, called gmapping, TurtleBot is capable of mapping an environment using SLAM and generating a map. It can then autonomously navigate using the generated map with Adaptive Monte Carlo Localization (AMCL). AMCL is a probabilistic localization system based on Monte Carlo localization [16] for robots in a 2D map and is used to track pose of a robot against a known map.

ROS contains many drivers and tools that work with TurtleBot. This includes the transfer function of TurtleBot, which is the dimensions of the robot, positional information of base relative to the camera sensor, etc.


3.4 Depth Perception

Depth perception is an essential feature for autonomous robotic navigation. Knowledge of distance from objects in the robot’s surroundings allows path planning algorithms to navigate the robot while avoiding obstacles. Most common methods to obtain such depth data are based on laser scanners and stereoscopic cameras. The Kinect is an example of depth sensor using stereoscopic camera.

A digital camera gives a two dimensional matrix of pixels. Each pixel contains some information depending on the type of camera sensor. A monochromatic sensor, for example, would give intensity in a pixel. Lets say the sensor gives an 8-bit value for a monochromatic image pixel, then the pixel would be black for the value of zero and white for the value of 255 with all other values representing corresponding shades of gray. The similar concept applies for RGB camera sensors. Each pixel contains three intensity values for each, red, green and blue colors.

A stereoscopic camera is a device consisting two camera sensors with a fixed distance between them. A diagram of a stereoscopic system is shown in Fig. 3.1. Since the two sensors face in the same direction, but are apart by a small distance, both will produce images with objects that are slightly shifted relative to one another. The depth at a pixel can be computed by its disparity in the two images, provided that you can match pixels between the two images.

![Simplified stereo vision system](image)

Figure 3.1: Simplified stereo vision system [17]
Equations 3.1 can be used to compute the depth based on Fig. 3.1:

\[ u_L = f \times \frac{X}{Z} \quad (3.1a) \]
\[ u_R = f \times \frac{(X - b)}{Z} \quad (3.1b) \]
\[ \text{disparity} = u_L - u_R = f \times \frac{b}{Z} \quad (3.1c) \]
\[ \text{depth} = f \times \frac{b}{\text{disparity}} \quad (3.1d) \]

where:

\( b \) = distance between the cameras,
\( f \) = focal length of a camera,
\( X_A \) = X-axis of a camera,
\( Z_A \) = optical axis of a camera,
\( P \) = real-world point defined by the coordinates \( X, Y, \) and \( Z, \)
\( u_L \) = projection of the real-world point \( P \) in an image acquired by the left camera,
\( u_R \) = projection of the real-world point \( P \) in an image acquired by the right camera.

In computer vision, an algorithm identifies the shift of each pixel in an image with respect to the other by matching similarities between objects. The output of such an algorithm is another image called the **disparity image**, which is a 2D matrix of intensities that correspond to the depth. OpenCV [18] has an inbuilt algorithm that accepts two images (left and right images from a stereoscopic camera) and camera parameters such as distance between them, focal length, etc, and gives a disparity image as an output. An example is shown in Fig. 3.2. The images are from two infrared camera sensors on a Leap Motion controller and the disparity image was computed using OpenCV.

The Microsoft Kinect sensor has one RGB camera and a pair of InfraRed (IR) cameras. It uses the IR stereo cameras to compute depth and overlays it with the RGB information from the RGB camera. This results in new matrix called **registered depth image** that has RGB-D information per
pixel. The advantage of using the Kinect is that it provides this registered image directly, reducing computation and programming cost on the computer. Hence it is widely used by developers in robotics.

### 3.5 Deep learning in Computer Vision

Machine Learning is a field in computer science that focuses on creating models by training features from training data and makes predictions on new sets of data. The training may be supervised or unsupervised. In supervised training, data is associated with annotations that represent features of the data needed to be learned. In unsupervised training, the algorithm decides features that should be extracted from data, and uses that for learning.

**Deep learning** is a type of machine learning with many layers of processing and feature extraction. The algorithms are designed to learn from multiple levels of feature extractions and pattern recognitions. Each level uses outputs from precious layers.
**Convolutional Neural Networks** (CNNs) are deep learning architectures based on artificial neural networks that are inspired from biological neural networks, especially the visual cortex of the brain (which handles processing of visual information in the brain). They use approximated mathematical models of stimuli on individual neurons. CNNs are hence widely used for image and video recognition and natural language processing. **Caffe** is an example of deep learning framework, developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors [19], based on CNNs. It is fairly popular in computer vision research and is also open source.

For the purpose of object identification, a combination of region proposal networks and object classification using CNN has been implemented and is called **R-CNN** [20]. The region proposal network is responsible for predicting object bounds in an image. A predicted region where there is likely an object is called a region proposal. The system extracts around 2000 region proposals in a given input image and uses a CNN trained for feature extraction on each region. It then classifies each region using an SVM with confidence values.

![Figure 3.3: R-CNN: Regions with CNN features.](image)

Figure 3.3 shows R-CNN process with a sample image. Region proposals are extracted from the input image and warped to a standard dimension, defined by the CNN framework. Each region is then fed into a CNN to compute features in it and predict its class. In the sample input image, one of the regions is finally classified positive for ‘exit’ while also classifying it as negative for ‘door’ and ‘elevator’.

A further improved version of Regional Convolutional Neural Network (R-CNN) is proposed and implemented in [21]. The paper shows significant improvement in object classification as well as model training. Caffe is the CNN framework used in the implementation of Faster R-CNN.
The project is also open source and can be integrated with ROS as a part of computer vision pipeline.

3.6 Controller Board

The brain of the robot is the controller board. It needs to be capable of running ROS along with computer vision processes and navigational controller. To install ROS, the controller board needs to run Linux OS, preferably Ubuntu 16.0.4. The computer vision process involves feeding live video from the robot’s camera through a trained neural network model to classify and localize objects. This requires dedicated GPU cores to maintain a relatively decent frame rate that should be able to process multiple frames per second.

Two boards of similar costs were considered for the job and compared so as to finalize one. The two boards are NVIDIA Jetson TK1 and the Intel RealSense board. Table 3.1 shows the comparison of the specifications of the two boards.

Considering the requirement for dedicated GPU cores for computer vision process and CUDA support for Caffe, the NVIDIA TK1 board seems a much better choice than Intel RealSense. For this thesis, an upgraded version of NVIDIA TK1 board was chosen. This board is the NVIDIA TX1 board. Its key features are listed below:

- Graphics - 256 CUDA cores
- Processor - Quad ARM A57/2 MB L2 CPU
- Memory - 4 GB 64 bit LPDDR4 25.6 GB/s
- Capable of running Ubuntu 16.0.4
<table>
<thead>
<tr>
<th>Specification Names</th>
<th>NVIDIA TK1</th>
<th>Intel RealSense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SoC</strong></td>
<td>Tegra K1 Based on ARM Cortex-A15</td>
<td>Intel Atom x5-Z8550</td>
</tr>
<tr>
<td>CPU cores</td>
<td>Quad core + 1 companion low power core</td>
<td>Quad core</td>
</tr>
<tr>
<td>Clock</td>
<td>2.3 GHz</td>
<td>1.44 GHz - 1.92 GHz</td>
</tr>
<tr>
<td>64-bit</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Memory</td>
<td>2GB</td>
<td>4GB</td>
</tr>
<tr>
<td>Graphics</td>
<td>Kepler GPU</td>
<td>Integrated Intel HD 400</td>
</tr>
<tr>
<td>Graphic Units</td>
<td>192</td>
<td>12</td>
</tr>
<tr>
<td>CUDA Support</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>OpenCL Support</td>
<td>Yes</td>
<td>Yes, with Intel version of OpenCL</td>
</tr>
<tr>
<td>Storage</td>
<td>16GB</td>
<td>32GB</td>
</tr>
<tr>
<td>USB 2</td>
<td>1 Port - micro</td>
<td>4 Ports - Full size</td>
</tr>
<tr>
<td>USB 3</td>
<td>1 Port - Full size</td>
<td>1 Port - micro</td>
</tr>
<tr>
<td>Network</td>
<td>Ethernet, no WiFi</td>
<td>Ethernet, no WiFi</td>
</tr>
</tbody>
</table>

Table 3.1: Controller board specification comparison
Chapter 4

Objectives

The objective of this thesis is to create a proof of concept that demonstrates indoor navigation in an unfamiliar environment using only computer vision. For this purpose, a specific scenario to exit the building is considered. The prototype robot has to recognize exit signs in an indoor environment and follow them until it reaches an exit door or an elevator. This particular scenario is chosen because it is similar to how people would find their way out of an unfamiliar building. The advantage of using exit signs is that they provide directions at hallway intersections. This reduces the risk of aimless wandering and back tracing.

Building this prototype requires a computer vision system capable of image recognition as well as localization. The robot needs to recognize the exit signs in an image frame as well as localize its position in the image. With the location of the exit sign, the distance to the exit sign can be estimated from the depth information provided by the Kinect. The image recognition and localization can be achieved with the help of Faster R-CNN implementation in Caffe and a trained model for exit signs.

4.1 Training and model validation

Ideally, an existing pre-trained model could be used if available. As no such model is readily available currently, this thesis includes manually training a model. The training requires large image datasets of exit signs with corresponding labels. ImageNet [23] is a large database of image
datasets that is made easily accessible to researchers. Although there are a large number of image sets on ImageNet, there are not any specific ones for the ‘Exit Sign’.

In order to train a model, a large dataset of annotated images is required. It would be advantageous to include images of objects taken in multiple angles and under multiple lighting conditions. There are several ways of creating a data set that is large enough and exhaustive enough to create a well trained model:

1. Images of desired object, such that the frame is tightly bound around it.
2. Images that represent other similar objects, as negative markers.
3. Images sliced from an annotated video using software such as vatic \cite{24}.

According to the authors of Faster R-CNN, the training needs to be done on a system with dedicated graphics cards like NVIDIA Titan, K20, K40 etc. and with at least 3GB of memory.

Once a trained model is obtained, it needs to be validated using some test data. Validation is performed by separating out a portion of the training data as test data. Faster R-CNN provides scripts to do validation on test data.

4.2 Prototyping

A robot prototype needs to be built that integrates computer vision and navigation. This includes the creation of a computer vision pipeline that can use the trained model to recognize and localize objects and estimate real world distance. It also involves writing of a navigational controller that makes decisions based on objects that are recognized.

For the propose of prototyping, the TurtleBot robot development kit has been chosen. It includes a Microsoft Kinect that satisfies the requirement for depth perception. ROS will be used for writing software to communicate with the TurtleBot and integrate Faster R-CNN into computer vision pipeline.
4.3 Performance Analysis

With the trained model and the control system deployed on the prototype robot, several performances are analyzed. The analysis includes:

1. Accuracy measurements of identification and localization of exit signs using trained model.
2. Comparison of model with other literature
3. Accuracy of estimation of depth of signs
4. Computation time for each frame of video; effective frame-rate in real-time.
5. Effect on frame-rate by number of region proposals in Faster R-CNN.
6. Power consumption by the on-board computer.

This thesis does not present or analyze the power consumption of the TurtleBot Kobuki mobile base since that information is available in its data-sheet, and none of its electrical parameters have been modified.
Chapter 5

Implementation

This chapter describes the procedure followed for implementation of objectives of this thesis mentioned in chapter 4. This includes the following: model training, computer vision pipeline, navigational control system, and prototype setup.

5.1 Model Training

Training requires a large set of images along with annotations, for example many applications of R-CNN are based on models initially created by training on ImageNet [23], and then refined for new tasks. Although there are a large number of image sets on ImageNet, none of them are specific to exit signs. An image dataset was therefore created consisting of about 8,000 images for 8 classes of objects mentioned in Table 5.1. Images were collected by taking pictures and videos in a variety of buildings. This was then supplemented with more images taken from the Internet.

The annotation was done on the still images using LabelImg [25]. It provides a Graphical User Interface (GUI) that is used to draw bounding boxes around objects and label them with the class names. Its output is an Extensible Markup Language (XML) file with object class name and bounding box description that can be parsed by the training tool. The format of this markup is defined by the PASCAL VOC format used in ImageNet [23].

The annotation of videos was performed using Vatic [24]. The interface is similar to that of
Table 5.1: Classes in dataset with examples

<table>
<thead>
<tr>
<th>Object Class</th>
<th>Example Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit sign</td>
<td><img src="image1" alt="Exit sign" /></td>
</tr>
<tr>
<td>Exit sign with left arrow</td>
<td><img src="image2" alt="Exit left arrow" /></td>
</tr>
<tr>
<td>Exit sign with right arrow</td>
<td><img src="image3" alt="Exit right arrow" /></td>
</tr>
<tr>
<td>Exit sign with left &amp; right arrows</td>
<td><img src="image4" alt="Exit left &amp; right arrows" /></td>
</tr>
<tr>
<td>Stairway sign</td>
<td><img src="image5" alt="Stairway sign" /></td>
</tr>
<tr>
<td>Elevator sign</td>
<td><img src="image6" alt="Elevator sign" /></td>
</tr>
<tr>
<td>Elevator door</td>
<td><img src="image7" alt="Elevator door" /></td>
</tr>
<tr>
<td>Door</td>
<td><img src="image8" alt="Door" /></td>
</tr>
</tbody>
</table>

LabelImg and allows drawing bounding boxes around objects at multiple frames throughout the length of the video. The output is also in XML with object descriptions in every frame. For the purpose of training, the video was sliced into images for every frame and the annotation file was extracted to create separate XML file for every frame.

The training was performed using the python implementation of Faster R-CNN called py-faster-
5.2 Computer Vision Pipeline

The object classification and localization process is shown in Fig. 5.1. The inputs are the RGB image and the registered depth image (RBG+D) from the Kinect sensor. One could use RBG+D for classification but depth data was not used in the training as the process of data collection and annotation is far more complicated and performance improvement may not be as significant. A Caffe model obtained from the training process is also an input to the Faster R-CNN process.

Only the RGB image is used for obtaining object bounding boxes and classifications. The distance of the sign from the robot is computed by using the 2D position of object on the RGB image and finding its 3D position on the corresponding registered depth image.

Due to the low resolution of the RGB camera on the Kinect, a Full High Definition (FHD) camera is used to feed RGB images to the pipeline. Another advantage of using a separate camera is that it can be positioned to see exit signs that are predominantly placed near the ceiling.

5.3 Navigation Control System

The navigational control system is responsible for making a navigational decision for the robot when it has or does not have goals from the computer vision pipeline. The control system can be
divided into two sections: the navigational stack in [ROS] and the controller algorithm.

5.3.1 Navigational Stack

[ROS] provides a navigational stack setup that is responsible for generating costmaps from sensor data and planning paths for the robot with these costmaps.

Figure 5.2: ROS Navigational Stack Setup [26]

Figure 5.2 shows the navigation stack setup in [ROS]. The stack receives inputs such as sensor data and navigational goals. It gives the base controller output commands. Inputs that are platform dependent (TurtleBot in this case) are the sensor data, odometry data, and sensor transforms. Sensor data is used to generate local costmaps while odometry data is used for relative position estimation. The sensor transforms contain geometric information of the robot such as the relative position between the mobile base and the stereoscopic sensor.

The stack contains a node called move_base, that accepts simple goals (X and Y coordinates and rotation angles), plans a path avoiding obstacles and boundaries and sends velocity commands to the base controller (the mobile base driver). There are some optional nodes that can be seen in Fig. 5.2 which are external maps and AMCL parameters for navigation in a pre-mapped environment.
5.3.2 Controller Algorithm

The controller algorithm is responsible to receive data from the computer vision pipeline and make decisions for the next move of the robot. It keeps track of goals and updates them when required.

An algorithm is proposed based on the test case of following exit signs till the robot reaches a door or an elevator that is represented as a flow diagram shown in Fig. 5.3. The image goals are hence divided into global goals (exit doors or elevators) and local goals (exit signs). The controller waits for receiving valid data from the computer vision pipeline. Once the data is received, based on the object classification, it will decide if it can see a global goal or a local goal. If a global goal is detected, the controller tries to move the robot close to it and alert the user. If a local goal is detected, the controller tries to make the robot follow directions based on the suggested direction of the sign and then tries to look for the next local goal. This process repeats till it detects a global goal.
5.4 Prototype Setup

The initial setup included a TurtleBot, a laptop and a Raspberry Pi 2. The laptop runs on Ubuntu 16.0.4 while the Raspberry Pi runs on Ubuntu MATE. ROS Kinetic full desktop version was installed on the laptop. This included all ROS libraries, visualization tools like Rviz and rqt, 2D/3D

Figure 5.3: Navigational controller flow
simulators, and 2D/3D perception. ROS Kinetic base version was installed on Raspberry Pi that includes ROS package, build, and communication libraries without any GUI tools.

The Raspberry Pi is placed on the TurtleBot with USB connections to the Kobuki mobile base and the Microsoft Kinect sensor. It is connected to the same Wi-Fi network as the laptop to communicate with it. Figure 5.4 shows the initial robotic setup.

Several ROS packages were installed to communicate and interact with the TurtleBot. These packages contain drivers for the Kobuki mobile base and the Kinect. They are used to send instructions to move the robot and to get information from the Kinect; like the RGB images, depth images, registered depth images and laser scans.

The packages for TurtleBot also contain a node, called gmapping, for the purpose of mapping an environment and saving it. The gmapping node can also take in simple goals (x and y coordinates and rotation angle) and plan a path avoiding obstacles. The planning is based on Dijkstra’s algorithm. It uses the odometry data to make an estimation of the location of the robot after movement. The node is also responsible for the creation of costmaps using the laser scan information from the Kinect. The costmap is a raster that contains distance information around the robot. It has parameters that can be manually set such as minimum safe distance from obstacles, radius-of-turn around obstacles etc.

The current setup, shown in Fig. 5.5 replaces the Raspberry Pi with the NVIDIA Jetson TX1
board. The Jetson TX1 board is powered using a separate 14.8 V Li-ion battery. Although the TurtleBot has several power outlets, none of them were able to provide sufficient power to the board.

Another important addition to the setup is a Full HD camera. A higher resolution of RGB image is essential to recognize distant objects as they appear small relative to the image frame. The height of the Kinect has also been increased to make sure objects placed at a greater height are visible. This was required as exit signs are usually placed near the ceiling.

Figure 5.5: Current robotic setup
Chapter 6

Results

This chapter presents the results of this work. It includes comparisons with other literature.

6.1 Obstacle Avoidance

A C++ program was written to send simple navigational goals to the ROS navigational stack. These navigational goals are defined by distances in X and Y axes and the Z axis rotation. This allows the TurtleBot to reach its goal by avoiding any obstacles in its path. To test the object avoidance, a simple goal of moving 2 meters forward and then turning 180 degrees was given.

The images in Fig. 6.1 are screen shots of Rviz visualization software taken while the TurtleBot moved towards its goal. The blobs of pink, blue and purple in the images represent the costmap created by the TurtleBot. This costmap contains the real boundaries of objects in pink and the inflated space around it in blue. Inflated space is the minimum distance to be maintained between the object and the center of the robot, so that its edges do not collide with objects.

The TurtleBot can be identified as the gray circle-like figure with a green line near one edge. The screen shots were captured at different stages of the robot’s movement to its goal. In the first image the TurtleBot is facing upwards with respect to the frame. We can see that there is some obstacle directly ahead of it (pink blob) and the goal is beyond it. The node generates a path around this obstacle, using the information from the costmap, which is represented by a thin green line.
The TurtleBot can be seen following the green line and ultimately reaching its goal by the 5th image. Notice it has also turned nearly 180 degrees with respect to its initial orientation.
6.2 Object Classification

A Caffe node was built using code from [28]. This node receives a stream of images from the Kinect sensor of the TurtleBot and performs image recognition using a pre-trained Caffe model. The model used for image recognition is the reference model provided by authors of Caffe that is based on AlexNet [29].

![Kinect image of an acoustic guitar](image1)

![Object prediction by Caffe for acoustic guitar](image2)

![Kinect image of a folding chair](image3)

![Object prediction by Caffe for folding chair](image4)

Figure 6.2: Object classification using Caffe

From the screen shots in Fig. 6.2 it can be seen that Caffe predicts the object class from the pre-trained model with a very high confidence. For the acoustic guitar its confidence is an average of 0.822 and for the folding chair, an average of 0.941. The confidence scores should not be confused with probabilities. They are numerical estimates made by the system based on how close the image is to its training data. These values are useful for making soft decisions rather than hard
decisions. A predication can be made if the confidence score of a class is relatively higher than the scores for other classes. Hence, an important observation in the screen shots is that the confidence scores for objects that are not in the image are extremely low.

6.3 Model Evaluation

The model evaluation was performed using a testing script that is a part of py-faster-rcnn [22] on test data-set that was not a part of the training process. The test data-set consists of nearly 800 images, which is about 10% the size of training dataset.

The evaluation script uses the trained model to identify bounding boxes and classes on the images from the test data-set and compares them with the ground truth (given by human annotations). Based on this comparison detections are categorized as True Positive (TP), False Positive (FP), etc as shown in Table 6.1.

<table>
<thead>
<tr>
<th>Actual Negative</th>
<th>Predicted Negative</th>
<th>Predicted Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>TN</td>
<td>FP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actual Positive</th>
<th>Predicted Negative</th>
<th>Predicted Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN</td>
<td>TP</td>
<td></td>
</tr>
</tbody>
</table>

A detection is considered to be a TP when Intersection over Union (IoU) match percentage of the predicted bounding box and the ground truth bounding box is above a set threshold. The default match percentage threshold used in the test script is set at 50%. An example of this estimation for IoU is illustrated in Fig. 6.3.

Precision and recall are two parameters computed using equations 6.1 & 6.2. Precision is a ratio of the number of correct predictions to the number of predictions made by the system.
is a ratio of the number of correct predictions to the number of desired predictions.

\[
\text{precision} = \frac{\text{Number of correct predictions}}{\text{Number of retrieved predictions}} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{6.1}
\]

\[
\text{recall} = \frac{\text{Number of correct predictions}}{\text{Number of desired predictions}} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6.2}
\]

Neither of the two metrics alone give a complete picture of the success of the model. In order to understand the models performance better, a plot representing precision and recall curve is generated. This plot is called the PR curve. The points of the curve are value pairs of precision and recall computed at a certain confidence score threshold. The trained model is used for prediction on test data with values of confidence thresholds ranging between (0, 1). Plots representing the PR curves for some of the classes are shown in Fig. 6.4. An ideal PR curve would be rectangular with a right angle at (1, 1).

The script also provides the Average Precision (AveP), which is the area under the PR curve, for each of the classes. Table 6.2 shows the AveP for each of the classes and the mean Average
Figure 6.4: Sample PR curves for some object classes

Precision (mAP) across all classes. A higher value of AveP means a better performance of the model for that class.

Table 6.2: Results of model validation

<table>
<thead>
<tr>
<th>Object Class</th>
<th>AveP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit forward</td>
<td>89.14</td>
</tr>
<tr>
<td>Exit left</td>
<td>94.73</td>
</tr>
<tr>
<td>Exit right</td>
<td>91.94</td>
</tr>
<tr>
<td>Exit both</td>
<td>87.76</td>
</tr>
<tr>
<td>Stairs</td>
<td>76.02</td>
</tr>
<tr>
<td>Elevator</td>
<td>65.17</td>
</tr>
<tr>
<td>Elevator door</td>
<td>95.02</td>
</tr>
<tr>
<td>Door</td>
<td>89.03</td>
</tr>
<tr>
<td><strong>mAP</strong></td>
<td><strong>86.10</strong></td>
</tr>
</tbody>
</table>

The trained model fares well based on the validation results. Some of the results of detection on test images using the trained model can be seen in Fig. 6.5. Yet, there are many instances of false
positives when using the model in real world application. The most prominent errors are observed in the cases of directional exit signs. This is quite understandable considering the fact that arrows in exit signs are very small and are barely visible from a distance. The reliability however does increase when images are taken from a closer distance. Examples of false positives are shown in Fig. 6.6

Figure 6.5: Selected image examples of object detection results from R-CNN trained model for the eight object classes

Figure 6.6: Sample false positive detections
It also helps if the resolution of input RGB image is high. Since the exit signs use up a smaller portion of the image frame, a high resolution image is required for it to be detected as an object. Most of the testing was done on [FHD] images (1920 x 1080).

### 6.4 Model Comparison

In this thesis, a measure of a test’s accuracy, called the F-score, is used for comparison with other literature. The F-score is the harmonic mean of the precision and recall, defined in Eq. 6.3

\[
F = 2 \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\]  

(6.3)

For the purpose of comparison of metrics, two papers [11, 12] have been chosen that have explored the idea of recognizing exit signs. The papers, however, do not use an approach that can be directly compared with this work. They use metrics such as a success rate, that is not exactly the same as F-score, but it is the closest metric for comparison with the work in this thesis.

The authors in [11] use 10-fold cross validation method to validate their trained model. One hundred and fifty images were used in this case to train the model. An ideal comparison to this paper would have required doing a 10-fold cross validation as well but, due to the excessive computational and time cost of training with Faster R-CNN, the comparison is made based on a single validation using the test dataset.

The method employed in [12] does not involve machine learning, but instead uses saliency maps and query patterns to perform detection. Validation is done by running their algorithm on a set of images and reporting the number of TP and FP. Results are compared using F-score for similar classes, computed based on Eq. 6.3. A higher score indicates a better reliability.

Aaron et al. [11] provide only one statistic that is useful for comparison with this work that is the success rate. The authors do not provide an exact definition for success rate, but it is most likely the number of times an exit sign is detected correctly. Another factor is that they do not differentiate directional exit signs. To make a fair comparison, we only consider the results from our exit forward object class. As shown in Table 6.3, we have an accuracy of 89.66% compared to
Table 6.3: Comparison against existing methods

<table>
<thead>
<tr>
<th>Object Class</th>
<th>Aaron et al.[11] (Success%)</th>
<th>Wang and Tian[12] (F-score%)</th>
<th>This thesis (F-score%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit Forward</td>
<td>80</td>
<td>94.74&lt;sup&gt;1&lt;/sup&gt;</td>
<td>89.66&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Exit Left</td>
<td>NA</td>
<td>81.08&lt;sup&gt;2&lt;/sup&gt;</td>
<td>93.73&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>Exit Right</td>
<td>NA</td>
<td>90.47&lt;sup&gt;3&lt;/sup&gt;</td>
<td>93.33&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

<sup>1</sup>Based on 20 images in Wang and Tian vs 347 images in this thesis
<sup>2</sup>Based on 20 images in Wang and Tian vs 189 images in this thesis
<sup>3</sup>Based on 22 images in Wang and Tian vs 78 images in this thesis

their 80%, hence we can say that our model fares much better than their work.

Wang and Tian [12] focus on only signage detection and they have presented results for more indoor signs than exit signs. We only consider results from their forward, left, and right arrow signs. We compare those with our exit forward, exit left, and exit right signs. As shown in Table 6.3, our model fares better than theirs for exit left and exit right signs even though their model was tested on a very small data set compared to ours. For the exit forward case, our score is nearly 5% less, but an important factor to consider is that the model presented in this thesis was tested on a much larger set of data (347 images) than their model (20 images). Overall, we believe our model is better as our test data set is much larger, at the very least 4 times larger, with the exit forward dataset being 17 times that of their data set.

### 6.5 Computer Vision Pipeline Analysis

The trained model is run on the NVIDIA Jetson TX1 within the [ROS](https://www.ros.org) framework with the help of OpenCV [18] which are all installed on a TurtleBot. This allows feeding images in real-time to Faster [R-CNN](https://arxiv.org/abs/1506.01497) as a part of the computer vision pipeline. An example of the output from the computer vision pipeline is shown in Fig. 6.7.

Figure 6.7 shows bounding boxes around objects successfully classified by the system along with the classification names. The numbers beside the names are the confidence scores for the
Figure 6.7: Sample output from pipeline running on prototype robot (using Kinect RGB camera) classification. Below the bounding box in yellow text is the distance of the objects from the camera in meters. Kinect RGB camera was used as the RGB image input to the pipeline.

Figure 6.8: Result of mapping bounding box coordinates from HD camera to Kinect camera

The current setup uses a FHD camera as the RGB image input for the computer vision pipeline. The coordinates of the output bounding boxes are mapped onto the Kinect camera resolution so that the depth of the correct region is obtained. Figure 6.8 shows bounding boxes obtained by using the FHD camera and a mapped bounding box on the Kinect camera. Since the cameras are placed very close to each other in the setup, the mapping works well although it is not perfectly aligned with the object. This error does not affect the accuracy of depth perception because only the minimum
depth of all points inside the bounding box is considered.

Table 6.4: Execution time vs. number of region proposals (data averaged over 10 consecutive frames)

<table>
<thead>
<tr>
<th>Number of region proposals</th>
<th>Average Execution Time (s)</th>
<th>Average number of correct detections</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>0.953</td>
<td>3</td>
</tr>
<tr>
<td>350</td>
<td>0.830</td>
<td>3</td>
</tr>
<tr>
<td>300 (default)</td>
<td>0.789</td>
<td>3</td>
</tr>
<tr>
<td>250</td>
<td>0.909</td>
<td>3</td>
</tr>
<tr>
<td>200</td>
<td>0.872</td>
<td>3</td>
</tr>
<tr>
<td>150</td>
<td>0.732</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>0.599</td>
<td>3</td>
</tr>
<tr>
<td>90</td>
<td>0.586</td>
<td>3</td>
</tr>
<tr>
<td>80</td>
<td>0.581</td>
<td>3</td>
</tr>
<tr>
<td>70</td>
<td>0.576</td>
<td>2</td>
</tr>
<tr>
<td>60</td>
<td>0.466</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>0.460</td>
<td>1.7</td>
</tr>
<tr>
<td>40</td>
<td>0.449</td>
<td>1.9</td>
</tr>
<tr>
<td>30</td>
<td>0.436</td>
<td>1.5</td>
</tr>
<tr>
<td>20</td>
<td>0.426</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>0.362</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Using default configuration in Faster R-CNN, the average processing time for a single frame is 0.789s. There are two parameters that affect execution time significantly. These are the resolution of the image and the number of region proposals in Faster R-CNN. Lowering the resolution reduces performance significantly, hence only 1920 X 1080 image stream was used. Changing the number of region proposals affect the execution time as there are fewer regions that are processed by the CNN.

Table 6.4 shows the variation of execution time with the reduction in number of region proposals along with the number of objects detected. In this experiment, three exit signs were placed in the
camera’s field of view and the average execution time and average number of successful detections were recorded for 10 consecutive frames. For region proposals greater than 80, all three signs were detected and classified correctly. For lower number of region proposals the system failed to consistently detect all three exit signs. It missed the signs in a few of the frames hence reducing the average number of detections. The delta was reduced from 50 to 10 to for region proposals below 100 as the average number of detections were inconsistent.

The default number of region proposals is 300. Based on the results from Table 6.4 for the specific experiment, it is possible to obtain better execution time by reducing number of region proposals to 80. The minimum threshold for number of region proposals may vary for a different case and the maximum number of expected detections. In the case of exit signs, it may be assumed that the maximum number is around 3 - 4. Smaller execution times allow for quicker decisions in the navigational controller.

### 6.6 Power Analysis

In this section the power consumption by Jetson TX1 board while running all the processes is analyzed. The processes that are concurrently running include:

- TurtleBot mobile base driver
- Kinect driver
- ROS navigational stack
- Computer vision pipeline that incorporates Faster R-CNN
- Navigational controller

The measurements for electrical parameters are performed by reading monitors on-board the Jetson TX1 using specific locations in the Operating System (OS). For example, the address to read the power consumption in milliwatts (mW) is:

```
/sys/devices/platform/7000c400.i2c/i2c-1/1-0040/iio_device/in_power0_input
```
Table 6.5 shows the measurement taken for voltage, current, and power averaged over a duration of 60 seconds while running all the processes mentioned above.

Table 6.5: Electrical measurements for Jetson TX1 board

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage</td>
<td>15.48</td>
<td>V</td>
</tr>
<tr>
<td>Current</td>
<td>615</td>
<td>mA</td>
</tr>
<tr>
<td>Power</td>
<td>9.511</td>
<td>W</td>
</tr>
</tbody>
</table>

Based on the current consumption, an estimate of battery life can be made with the knowledge of the battery’s capacity. In the current setup the battery used to power the board is rated at 14.8V and has a capacity of 2200 milliampere hour (mAh). This implies that the estimated battery life is 3 hours and 57 minutes on a single charge.
Chapter 7

Conclusion

7.1 Summary

A variety of hardware and software systems were integrated to create a working prototype robot that follows exit signs to exit a building. This prototype is a proof of concept that can be further developed as an assistive technology application to help visually impaired individuals exit a building.

The work presented in this thesis is twofold: first, a model has been developed and trained using computer vision to assist with successful indoor navigation toward defined goals; and second, a prototype robot has been built and tested. The model has been successfully trained to identify classes needed for an active assistive robot using Faster R-CNN. This includes a contribution of a dataset containing about 8800 images with annotations. Validation showed good performance for the model and it competes with performance of related work. The robot has been built using a TurtleBot with a Kinect sensor controlled by an NVIDIA Jetson TX1. The prototype robot is able to recognize exit signs and estimate its distance. It is also able to make simple navigational decision based on the controller algorithm presented in this thesis.

Although this work is targeted to be a proof-of-concept, the results presented and analyzed are very encouraging. Combined with the improvements outlined in Section 7.2, this work has the potential to be a turning point in the field of assistive technology.
7.2 Future Work

The navigational controller can be improved to consider more complex scenarios. For example, if no image based goals are available, it can keep moving forward in the direction obtained by the previous goal in anticipation for the next goal. If it reaches a roadblock, it would stop and alert the user about its status so that the user may take control and decide a potential direction.

There are several cases that have not been implemented while designing the navigational controller. Some of these potential cases are listed below along with a possible solution:

- **Multiple visible exit signs detected**: The closest exit sign would be considered. This can be done either in the computer vision pipeline by comparing the size of bounding boxes or in the navigational controller by comparing the distances of the detected signs.

- **Sign is out of robot’s field of view**: This case is highly likely to happen as the robot moves closer to the sign because the Kinect is always facing forward. The navigational controller needs to be designed to keep updating its goals while the sign is visible and use the last known goal when no sign is visible. This involves not only keeping track of the position of the goal but also the suggested direction of the sign. The desired outcome here is that if the sign indicated a right turn, the robot should go under the sign and successfully turn towards indicated direction, a right turn in this case.

- **A false positive detection (noise)**: There is always a possibility of identification of something in the image that is not the desired object. One way to tackle this issue is to compare certain parameters of identified objects, such as its bounding box size, its position relative to the image frame and its distance from the robot, with the corresponding parameters of objects in previous frames. Using multiple frames to update goals would also help eliminate noise in the predictions. An identified sign would only be treated as a goal when it is seen in say 90% of the frames per set. The number of frames per set would depend on the frame-rate achieved by the computer.

For practical applications, the robot has to be redesigned to mimic the white cane. For example, the mobile base can be replaced with a smaller spherical robot while the camera and depth sensors
can be placed near the handle. The controller board can be reduced by fabricating a custom PCB
design and placed hidden under the handle. To minimize the size, the batteries may be stacked
inside the neck of the structure. Using a spherical lens camera would also be useful as it would
increase the robots field of view minimizing the need to look for exit signs. Computer vision
pipeline speeds can be improved by predicting a region where the exit sign may be in the image
frame and processing that smaller region of the image to get exact location.

For a deployable service robot, the prototype has to be scaled to cater to all mobility requirements
of the user. This involves extending the computer vision model to identify most indoor and outdoor
signs and updating the navigational controller algorithm to incorporate all the new information. The
robot would need some form of communication with the user. This could be a voice assistant that
can interpret speech and respond via verbal feedback. Robotics is multi-disciplinary and creation
of a truly usable robot requires extensive research in several engineering fields.
Glossary

AMCL  Adaptive Monte Carlo Localization

AveP  Average Precision

CCD  Charge-Coupled Device

CNN  Convolutional Neural Network

FHD  Full High Definition

FN  False Negative

FP  False Positive

GPS  Global Positioning System

GUI  Graphical User Interface

IoU  Intersection over Union

LiDAR  Light Detection And Ranging

mA  milliampere

mAh  milliampere hour

mAP  mean Average Precision

mW  milliwatts
**OS** Operating System

**PTZ** Pan-Tilt-Zoom

**ROS** Robot Operating System

**R-CNN** Regional Convolutional Neural Network

**SLAM** Simultaneous Localization And Mapping

**SoC** System on Chip

**SONAR** Sound Navigation And Ranging

**SVM** Support Vector Machine

**TN** True Negative

**TP** True Positive

**XML** Extensible Markup Language
Bibliography


