Robot Following Ahead of the Leader and Learning Human-Relevant Navigational Cues

by

Payam Nikdel

B.Sc., Shiraz University, 2015

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY
Fall 2018

Approval

Name:	**Payam Nikdel**
Degree:	**Master of Science (Computing Science)**
Title:	**Robot Following Ahead of the Leader and Learning Human-Relevant Navigational Cues**
Examining Committee:	**Chair:** Angelica Lim Assistant Professor
	Richard Vaughan Senior Supervisor Professor
	Anoop Sarkar Supervisor Professor
	Mo Chen External Examiner Assistant Professor School of Computing Science
Date Defended:	**December 17, 2018**

Abstract

As robots become more involved in our everyday lives, we may find it useful to design new methods to interact with them. In this thesis, we design two applications to facilitate human-robot interaction.

The first application is an autonomous mobile robot that follows a walking user while staying ahead of them. Despite several useful applications for autonomous push-carts, this problem has received much less attention than the easier problem of following from behind. In contrast to previous work, we use multi-modal person detection and a human-motion model that considers obstacles to predict the future path of the user. We implement the system with a modular architecture of an obstacle mapper, a human tracker, a human motion model, a robot motion planner and a robot motion controller. We report on the performance of the robot in real world experiments. We believe that approaches to this largely overlooked problem could be useful in real industrial, domestic and entertainment applications in the near future.

The second application is a novel system for describing the navigational cues ahead of the robot. We train a Convolutional Neural Network (CNN) architecture on 2D LiDAR data and occupancy grid maps to detect navigational objects during robot movement through an indoor environment. These navigational objects include closed-rooms (room with closed doors), open-rooms (room with open doors) and intersections. On top of this network, our system uses a tracking module that improves the detection accuracy of the system by clustering and recording each detection of the model. This tracking module also enables us to describe the robot navigational cues. We evaluate the system in both simulation and the real world. We compare the combination of 2D LiDAR data and occupancy grid maps and using each of them alone.

Keywords: human-robot interaction; object detection; deep learning

Dedication

This thesis is dedicated to
my parents and my girlfriend Maryam
for their love,
endless support and
encouragement.

Acknowledgements

I would like to thank my senior supervisor Dr. Richard Vaughan for the continuous support of my research, for his patience, and immense knowledge. In addition, I would like to express my gratitude to Dr. Anoop Sarkar for many discussions we have had. His suggestions guide me to write the last part of my thesis.

I must express my very profound gratitude to my parents and my girlfriend, Maryam, for providing me support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

I am grateful to my friend Rakesh who collaborated with me during my research. I also thank my fellow lab-mates for all the great support, discussions and fun we had: Pratik, Rakesh, Jack, Faraz, Sepehr, Jeoff, Bita, Jacob and Amir.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nowadays, robots have found many applications in our daily lives. There are various kinds of assistive and service robots that can help people in different work-spaces. For example, there are service robots that can fetch items in hospitals or guide people in large stores or shopping malls [3]. These robots also can help improve the day-to-day life of users with disabilities. Consider a service robot that can guide visually impaired people using navigational instructions in a building. They could perhaps better imagine the environment [25], navigate more easily and travel to places they could not go before [11, 23].

Robots behave intelligently by sensing and reacting to their environment. They use different sensors and algorithms to map the surrounding obstacles. One of the common approaches being used is mapping the environment via Simultaneous Localization and Mapping (SLAM) [7]; this method constructs or updates a map of an unknown environment while keeping track of an agent's location using the relative observations of the robot (such as scans from a Laser Range Finder (LRF)). As we integrate more robots into our society, defining new methods of human-robot interactions may help to facilitate indoor navigational tasks.

This thesis tries to explore different approaches to make these interactions more natural. In particular, we design two mobile-robot applications. In the first application, we design a follow ahead behaviour in which a robot follows a user while staying in front of them. The second application is a system that describes the navigational options (e.g. number of paths in an intersection or positions of open and closed rooms) using 2-dimensional (2D) LiDAR data and occupancy grid maps to predict these navigational options.

In Chapter 2, we discuss our human-robot interaction for front-following. Our system uses two alternatives pipelines to detect the person of interest around the robot. Then, it tracks the person using our data association component. Next, we employ an Extended Kalman Filter (EKF) based on the position estimates of the robot and the person to obtain the person's heading. Finally, our motion model predicts the navigational goal of the robot to be a fixed "follow-ahead" distance in front of the person using the velocity estimated by

EKF and the obstacles in the environment. The results of this work have been published at the International Conference on Robotics and Automation (ICRA) 2018 [30].

In Chapter 3, we explore our novel method to describe navigational target classes (including closed-room, open-room and intersections) as the robot navigates through an indoor environment. Our system uses a combination of 2D LiDAR data and local occupancy grid maps to detect these navigational options ahead of the robot. In this study, we generate a dataset containing robot sensor readings along with the positions and labels of closed-rooms, open-rooms and beginning of intersections in each sensor data. Then, we train three alternative models to predict the positions and labels of these target classes based on the sensor data. We add our own tracking module which helps to aggregate the predictions and locate the target classes more precisely. Finally, we evaluate our system by conducting three experiments in both simulation and the real world to compute the performance of the three models.

Chapter 2

Follow Ahead

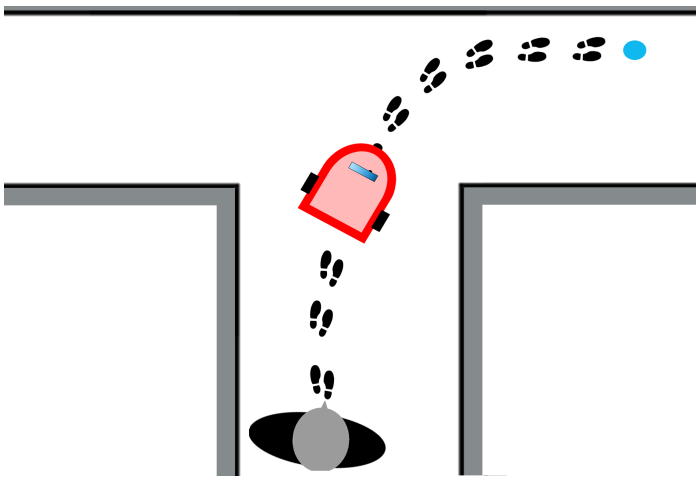


Figure 2.1: A mobile robot following-ahead of a user. The robot must predict the user's trajectory to stay in the correct position. If prediction fails, the robot must quickly recover.

Rapid technical developments will bring robots to our everyday lives. There are various applications where robots could usefully follow a human user around to assist them, for example a golf caddy or self-driving luggage. Notably, Boston Dynamics' LS3 legged robots (unpublished, derived from BigDog [44]) had a well-developed person-following capability to act as load carrying mules.

Ho et al. categorize person following into three categories: 1) following behind the leader, 2) side-by-side with the leader, and 3) ahead of the leader [17] . Following behind the leader is the most simple task as it can be implemented with a simple proportional controller that tries to keep the person in the middle of the sensor field of view and at a fixed distance. The other two tasks are significantly more challenging as they require a predictive model of the user's motion [26]. For example, for smooth following-ahead through an intersection, the robot needs to predict which direction the user will take. This problem has not been addressed well.

Following ahead has a number of useful applications. Consider the push-carts in daily use in logistics warehouses, hotels, libraries, and supermarkets; they are best placed right in front of the user for quick access. Jung et al. did an experiment in which participants were told to walk in a straight line while a robot followed them from behind [19]. They observe that participants look back to check the robot out of curiosity or fear of getting hit by the robot, imposing a cognitive load. The second advantage of following-ahead is that the user can see the cart for security: e.g. when walking through an airport with valuable luggage. In entertainment, an automatic UAV camera platform that follows ahead could capture a view of the user's face in recreational activities like skiing and mountain biking. Autonomous carts are frequently proposed and commercial devices exist but *without* follow-in-front user interaction, but using global map-based navigation instead [1].

In this chapter, we update and extend the limited previous work on this task by considering the user's path through obstacles. Using an RGB-D camera and laser scanner data, we estimate the relative position and velocity of the uninstrumented user, using an Extended Kalman Filter (EKF), and predict their future trajectory using a simple motion model interacting with a local navigability map acquired in parallel.

The contributions of this chapter are: first, we propose this problem as worthy of renewed attention due to its potential utility. Second, we provide a simple novel model for human motion in a constrained environment. Third, we provide a complete and freely available modular implementation in ROS, with the major components of obstacle mapper, human tracker, human motion model, robot motion planner and robot motion controller easily replaceable. We report on the performance of the robot in real-world experiments.

2.1 Related Work

Person tracking has been widely studied. Munaro and Menegatti propose a multiple target tracker system using Kinect RGB-D detections [29]. They use a depth based sub-clustering method to track people even near walls. Wojke et al. successfully tracked a user/operator using Bayes-optimal state estimator using RGB-D detections and image-based classification [43]. Koide and Miura developed a system that uses height and gait information to achieve person tracking and identification [20].

Takano et al. propose an approach to predict human motion using symbolic inference [38]. Ziebart et al. use a Markov Decision Process (MDP) and maximum entropy learning to obtain a probabilistic model of pedestrian trajectories assuming purposeful behavior [50]. In a similar work, Kuderer et al. remove the underlying MDP assumption and reason about continuous trajectory as opposed to discrete states [21]. Our model is simpler than the aforementioned ones as we assume that a person predominantly walks in the same direction.

[1] E.g. `http://canvas.technology`

2.1.1 Following Behind

Most of the previous work on person following has involved following a user from behind. In [22], Leigh et al. present a human-centered tracking framework which classifies laser data as human or not human. The detected person positions are tracked using a Kalman Filter, then they apply separate PID controllers to obtain the angular and linear velocities of the robot. An interesting resource-limited example is Yao et al., where the Georgia Tech Miniature Autonomous Blimp detects and follows a person using a monocular camera [45]. They use a Haar face detector and a KLT feature tracker to track the user. In a recent work, Sun et al. present a following behind the leader behavior using a social forces model [37].

2.1.2 Following Ahead

There are very few papers that address the problem of following in front of a user. Cifuentes et al. propose an approach based on a human gait model that uses a wearable Inertial Measurement Unit (IMU) for estimating orientation [5]. Ho et al. calculate human orientation using a Kalman filter with a nonholonomic human model for estimating human linear and angular velocities, while a special-purpose robot motion controller aims to align the human-robot poses such that the robot follows from the front [17]. Jung et al. present a holonomic motion model for tracking a human while staying ahead [19]. In [40], Tominaga et al. present another front-following system using simple visual servoing that tries to keep a person (marked with an AR tag) in the center of the robot's view. The heading of the person is not considered, and the robot can easily lose the person at sharp turns. Recent work by Moustris and Tzafestas describes a front-following model that uses a modified dynamic window planner without considering the current heading of the person, which is important information when predicting motion relatively far into the future [26]. Their method is extended in [27], where they assume that the user's relative heading can be estimated by how far the user is offset from the middle of the robot's field of view. This cue may often be useful, but may not predict the user's future heading changes due to interactions with walls and other obstacles.

All previous following-ahead studies, with the exception of [26, 27], assume that the environment is obstacle free. Obstacles make the problem much more challenging as the human may often be occluded from the robot's sensor, for example as the robot turns a corner ahead of the user. The focus of our work is to address this issue. Our approach assumes that the human will often be out of view, so it uses its recent estimate of the user's position combined with a local model of the world, and a model of the user's speed and direction of motion, to figure out where the human is likely to go *or has gone*, when they disappear from view. This mechanism also provides robust recovery when the robot guesses a turn incorrectly.

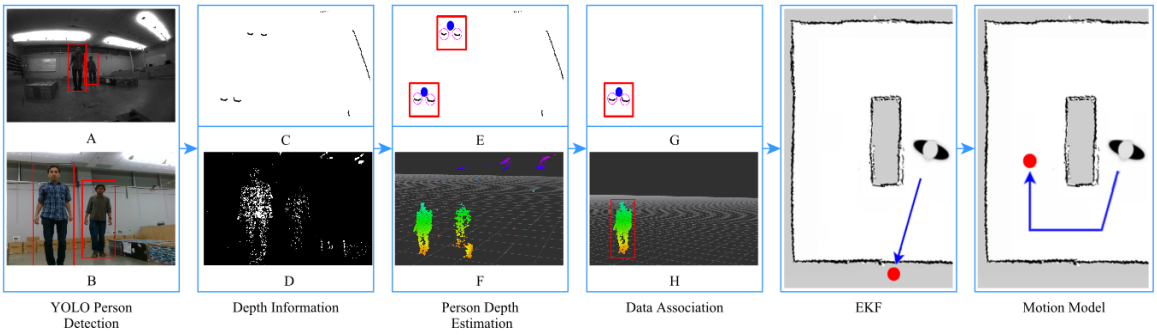


Figure 2.2: Two alternate system processing pipelines to track the user. Top row: Fisheye camera and laser rangefinder, Bottom row: RGB-D camera. [A] Fisheye image with person ROIs, [B] RGB image with person ROIs, [C] Raw laser scan, [D] Raw depth image, [E] Leg detections on laser scan segmented based on person ROIs (blue circles represent estimated positions of people), [F] Point cloud of people obtained from depth image after segmentation based on ROIs and median filter, [G] and [H] Selected person after data association for wide FOV detection mode and narrow FOV detection mode

2.2 Method

2.2.1 System Overview

The system is implemented in ROS [34] on a Pioneer P3-DX mobile robot. We implemented two alternate sensor modes for detecting humans: one with a narrow FOV and the other with a wide FOV. The former uses an RGB and depth image pair as input while the latter uses a monocular image from a fisheye camera combined with a 2D laser range scan. All the images are obtained from Intel® RealSense™ ZR300 development kit and laser scans are obtained from Hokuyo URG-04LX Laser Range Finder (LRF). Both the sensors are back-facing with respect to the robot. These positions are fed to our data association module which fuses the data to give a measurement of the position of the person to be tracked relative to the robot.

An EKF is used to estimate the speed and direction of the user. This information is used to make a naive prediction of the user's future pose without taking the environment into account. This prediction is then updated using our motion model based on the user's heading and a map of the environment.

An occupancy grid map is built incrementally during runtime as the robot navigates. The map is used in robot path planning as well as human motion prediction. The robot performs Simultaneous Localization and Mapping (SLAM) using the well-known GMap ROS implementation of the grid mapping technique [12], using odometry and range data from a front-facing Hokuyo LRF. We use the ROS navigation module[2] to navigate the robot to goal positions. The trajectory planning module is based on [9].

[2] `http://ros.org/wiki/navigation`

The processing pipeline is summarized in Fig. 2.2 and the main components are described below. The system was developed using the Stage robot simulator [41] before real-world testing. In the simulation, we abstracted the details of the person detection using Stage's color-blob finder.

2.2.2 Person Detection

We use YOLOv2 [35] on the narrow RGB or fisheye grayscale image from the RealSense device to get a Region of Interest (ROI) corresponding to the person. We use the ROIs to segment the range data from the calibrated depth image or laser scan that corresponds to people, as follows.

2.2.2.1 RGB-D camera, narrow field of view mode

Using the depth image, we compute the set of 3D points that correspond to the detected person's ROI. We reject background points inside the ROI using depth thresholding. The position of the person (in the image plane) is estimated as the center of the bounding box of the resulting point cloud while the depth is estimated as the median depth of the cloud.

2.2.2.2 Fisheye camera + LRF, Wide field of view mode

Person ROIs in fisheye images are used to filter the corresponding scans from the LRF. Human leg-detector [24] is used to find the distance and bearing to candidate humans: if a candidate is detected at the same bearing as the camera ROI, we have detected a candidate interaction partner.

Because the FOV of the of LRF (240°) is higher than that of the fisheye camera (166°), the person can still be tracked using raw laser data, as explained in Section 2.2.3. Leg detection on raw range scans without tracking leads to numerous false positives, but our data association maintains reliable tracking of the person we have previously seen with the camera.

The leg-based person detector fails to detect two valid legs for a person at large distances, often finding just a single leg which can still be used to track a previously-identified person. We use the leg-only detection only when we have already selected the person to track (Section 2.2.3).

2.2.3 Data Association

To ensure that we correctly track the person interacting with the robot, we designed a tracking state machine using Nearest Neighbor (NN) data association. This reliably tracks the user when there are multiple humans in the sensor FOV or the user goes out of the FOV, or when the detectors temporarily fail to detect the valid candidate still in the FOV.

We have four tracking states:

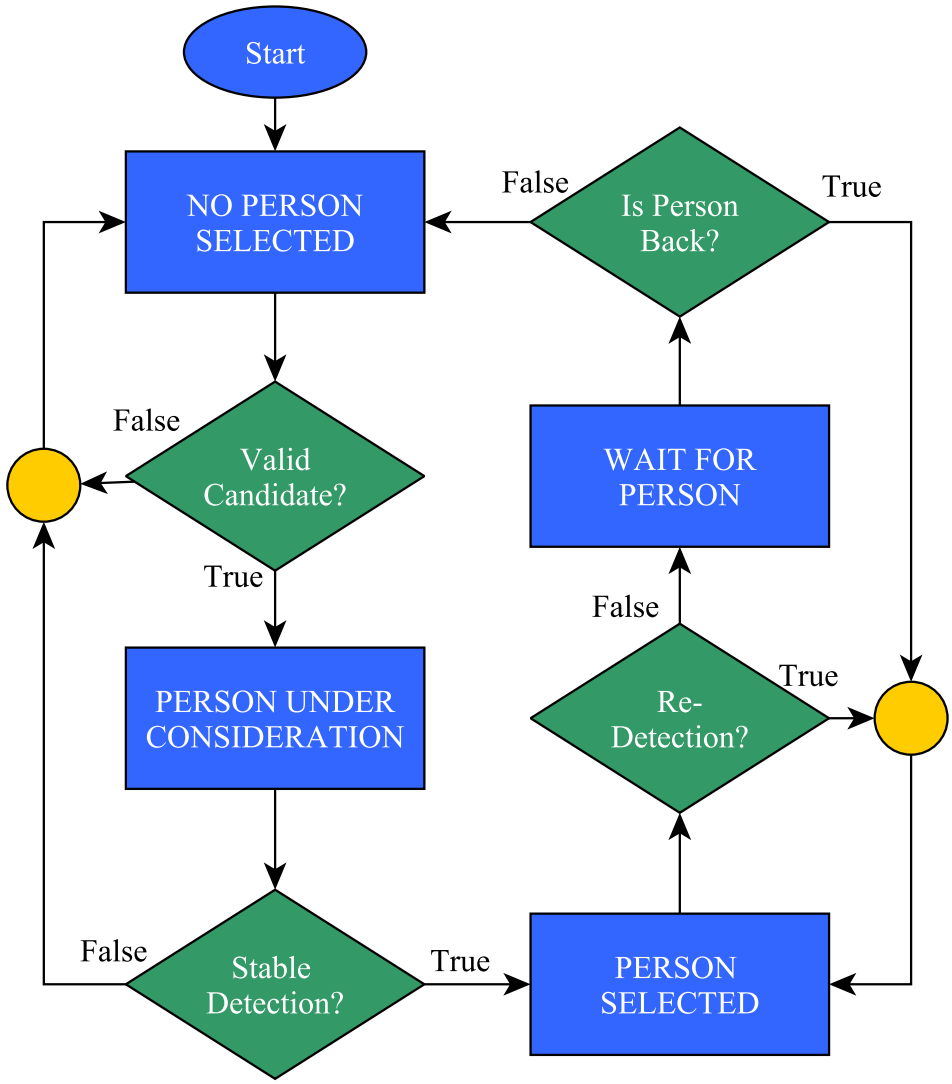


Figure 2.3: Data Association Flowchart

- *NO_PERSON_SELECTED*
- *PERSON_UNDER_CONSIDERATION*
- *PERSON_SELECTED*
- *WAITING_FOR_PERSON*

The starting state is *NO_PERSON_SELECTED*. When human(s) are detected the data association module chooses the person closest to the robot among the candidates that are closer than a fixed initiation distance. We set this distance to 2.5m for our experiments. The state is then updated to *PERSON_UNDER_CONSIDERATION*. In the following N detections, if a person is present in close proximity to the position of the person

in previous iteration the state is changed to *PERSON_SELECTED*, otherwise the state goes back to *NO_PERSON_SELECTED*. If the selected person is no longer detected, the state changes to *WAITING_FOR_PERSON* for a fixed time period before going back to *NO_PERSON_SELECTED*. Fig. 2.3 illustrates the state machine for data association.

The person position is fed to the EKF only if the tracking state is *PERSON_SELECTED*. Note that our implementation does not require the person to always be within a Human Interaction Zone [26, 5] to be tracked, but requires the person to be close enough to the robot to initiate the interaction.

2.2.4 Person State Estimation

2.2.4.1 Person State representation

Following the nonholonomic model of human motion as proposed by [2], we represent the state of the person at time t with position (x_t, y_t), walking direction θ_t and velocity along the walking direction v_t. The states are relative to the map frame of the SLAM system. Additionally, we maintain the position of the person in the previous time step $t-1$. The state transition from time t to $t+1$ is then given by:

$$\begin{aligned} x_{t+1} &= x_t + v\cos(\theta)dt \\ y_{t+1} &= y_t + v\sin(\theta)dt \\ v_{t+1} &= (1-\alpha_v)v_t + \alpha_v\|(x_t, y_t) - (x_{t-1}, y_{t-1})\|/dt \\ \theta_{t+1} &= (1-\alpha_\theta)\theta_t + \alpha_\theta atan2(y_t - y_{t-1}, x_t - x_{t-1}) \end{aligned} \tag{2.1}$$

where dt is the interval between two consecutive times.

We model velocity and direction as smoothly changing quantities by implementing updates in the form of an Infinite Impulse Response (IIR) filter[3]. The coefficients α_v and α_θ of IIR filter lie in the range $[0, 1]$ and represent how fast the values are expected to change (higher values corresponding to higher change). We found by trial and error the values around 0.5 give us the best results.

For the measurement model, we assume that the position (x_t, y_t) of the person in the global frame is observable. Our actual measurement consists of position with respect to the robot, but given that we have an estimate of the robot position and orientation from SLAM, we can find the position of the person in global frame. We propagate the uncertainty in robot pose to our measurement to make state estimation more robust, which will be discussed in Section 2.2.4.2.

[17] use a similar model in which the state of the person is computed from the robot's local frame and transform the reference frame using odometry readings at every time step.

[3]For averaging angles, we use the orientation of weighted sum of unit vectors along the angles to avoid angle wraparound discrepancy

This is likely to introduce drift in estimates over time with respect to a global frame. Although the drift in the coordinate frame itself does not adversely affect the person following behavior, using just the odometry is likely to give noisy estimates even over shorter time intervals. Our approach has the advantage of potentially better state estimates at the cost of additional sensing and localization/mapping.

We use an EKF for state estimation. Because our measurements are based on the fairly accurate absolute robot pose from SLAM and relative estimates of human position, we discount the linearization error inherent to the EKF. Improved variants of the nonlinear Kalman Filter like the Unscented Kalman Filter [42] could be substituted to improve the state estimation.

2.2.4.2 Uncertainty Propagation

Assuming a zero-mean Gaussian distribution of noise on input $\mathbf{X}$ with covariance $\mathbf{\Sigma}_X$, the noise propagated to the function $f(\mathbf{X})$ can be approximated to first order by

$$\mathbf{\Sigma}_f \approx \mathbf{J}_f \mathbf{\Sigma}_\mathbf{X} \mathbf{J}_f^T \tag{2.2}$$

where $\mathbf{J}_f$ is the Jacobian of $f(\mathbf{X})$.

Let r be the distance estimate and ϕ the bearing angle estimate of the person relative to the robot, which are the actual measurements we receive from our sensors. Let (x_R, y_R) and θ_R be the position and orientation of the robot respectively, obtained from SLAM. The measurement of our human position (x, y) is obtained as:

$$\begin{aligned} x &= -r\cos(\theta_R)\cos(\phi) - r\sin(\theta_R)\sin(\phi) + x_R \\ y &= -r\cos(\theta_R)\sin(\phi) + r\sin(\theta_R)\cos(\phi) + y_R \end{aligned} \tag{2.3}$$

Using the uncertainty propagation formulation of Eq. (2.2), we can have estimates of noise in our virtual measurement (x, y) due to independent components r, ϕ, (x_R, y_R) and θ_R. This implementation allows us to utilize the difference in the noise in measurement from the two modes of person detection we currently have and also uncertainty in the robot pose while still having a simple kinematic model.

With the estimate of walking direction, we set our naive goal to be a fixed follow-ahead distance in front of the person in that direction. To ensure that the direction estimate is reliable, we take the goal position as valid only when the estimated velocity is above a threshold and estimated variance of orientation is below a threshold.

2.2.5 Person Motion Model

In open environments without obstacles, the estimated walking direction of the person from the EKF can be used trivially to set a navigation goal for the robot. However, in bounded

environments with walls and other structures, we need a better predictor of the person's path. We propose a simple geometric model of person motion that takes into account both the current heading of the person and the environment, assuming piecewise linearity of obstacles.

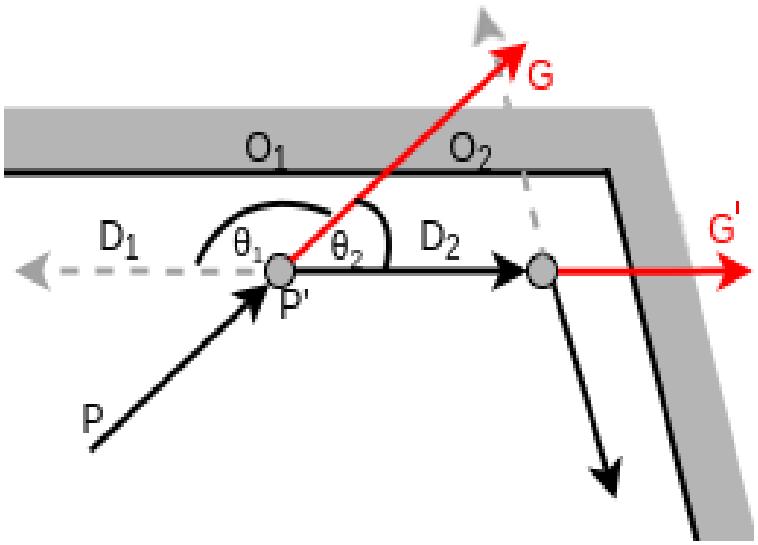


Figure 2.4: User motion model illustrated. P is the person position, with the arrow showing the heading. G is the naively predicted future position extrapolated from current velocity. The map-aware motion model predicts that the person turns at P' to be parallel to the wall (line segment (O_1, O_2) along D_2, and turns again in the corner.

Consider that a person is at point P and the naive predicted position at G, with O_1 and O_2 being endpoints of an obstacle (Fig. 2.4). Our model predicts that the person is going to change direction at point P', which is a fixed distance from the obstacle, and walks along one of two directions, $\mathbf{D_1}$ and $\mathbf{D_2}$, parallel to the obstacle. The direction is chosen based on the current heading of the person: the person will walk in the direction that minimizes the change in heading ($\mathbf{D_2}$ in this case), i.e. the smaller of the angles between vectors $\boldsymbol{\vec{PG}}$ and $\mathbf{D_1}$ and $\mathbf{D_2}$. This process is repeated until the distance covered is equal to the follow-ahead distance and the last point serves as the final goal position for the robot.

If the vector $\boldsymbol{\vec{P'G'}}$ is exactly perpendicular to the obstacle line (or approximately perpendicular in the presence of sensor noise), the situation is undecidable. This can be remedied for the updates to goal positions other than the original goal G by choosing the direction that has a smaller angle with $\boldsymbol{\vec{PG}}$ as it represents the direction that is closer to the person's heading. Also, we avoid choosing the direction that immediately leads to obstacles which is particularly helpful for sharp corners.

Our model is applied using the constantly-updating occupancy grid map from the SLAM system. We perform morphological image operations – dilation followed by erosion – to fill holes in the obstacles. Then we use ray-casting to detect obstacles between P and G. If

obstacles are detected, we do a Breadth First Search (BFS) for a limited distance starting from the known obstacle point. We find the best line that fit these points using a Hough transform.

If the person is no longer in the robot's FOV, which is the typical case in sharp turns, the robot simply goes to the last goal; before the robot gets there the person usually reappears in the sensor FOV.

2.3 Experiments

We conduct our experiments in two settings. In the first setting, the environment is an open rectangular area with flat walls. The robot has to follow in front of a person who walks around the room along the walls. The user makes two laps, turning eight corners. We refer to this as the 'Easy' setting. In our second setting, the same perimeter contains two rectangular obstacles such that there is space for walking between the obstacles and between obstacles and walls: topologically a figure eight. The user passes an intersection four times and turns at corners six times, making both right and left turns. We refer to this as the 'Difficult' environment. Our environments are depicted in Figure 2.5 along with the trajectories. The human subject for our experiments is one of the authors. The subject tries to repeat the same trajectory in all the experiments for a particular setting, guided by marks on the floor that are invisible to the robot. The trajectories of robot and human were recorded using an external Vicon motion capture system. The size of the test arena was limited by the usable Vicon area.

Our performance error metric is the percentage of total trial time the user had to wait for the robot to come in front and resume correct following-ahead behavior. A robot that predicts the user's motion perfectly (impossible in the 'Difficult' setting) and always stays ahead would score 0%. A robot that stayed still or followed behind would score 100%. These evaluations are done at the corners and turnings. We run 13 trials for each of the settings. The results (minimum, median and maximum waiting time) are summarized in Table 2.1. The evaluations were done using the wide FOV mode of detection (Section 2.2.2.2).

Setting	Minimum	Median	Maximum
Easy	2.4	5.0	12.0
Difficult	14.1	21.1	32.3

Table 2.1: User waiting time each setting, as a percentage of total trial time, over 13 repeated trials. This is an error measure of the total time a user had to stop walking and wait for the robot to resume correct following-ahead behavior after making a prediction mistake.

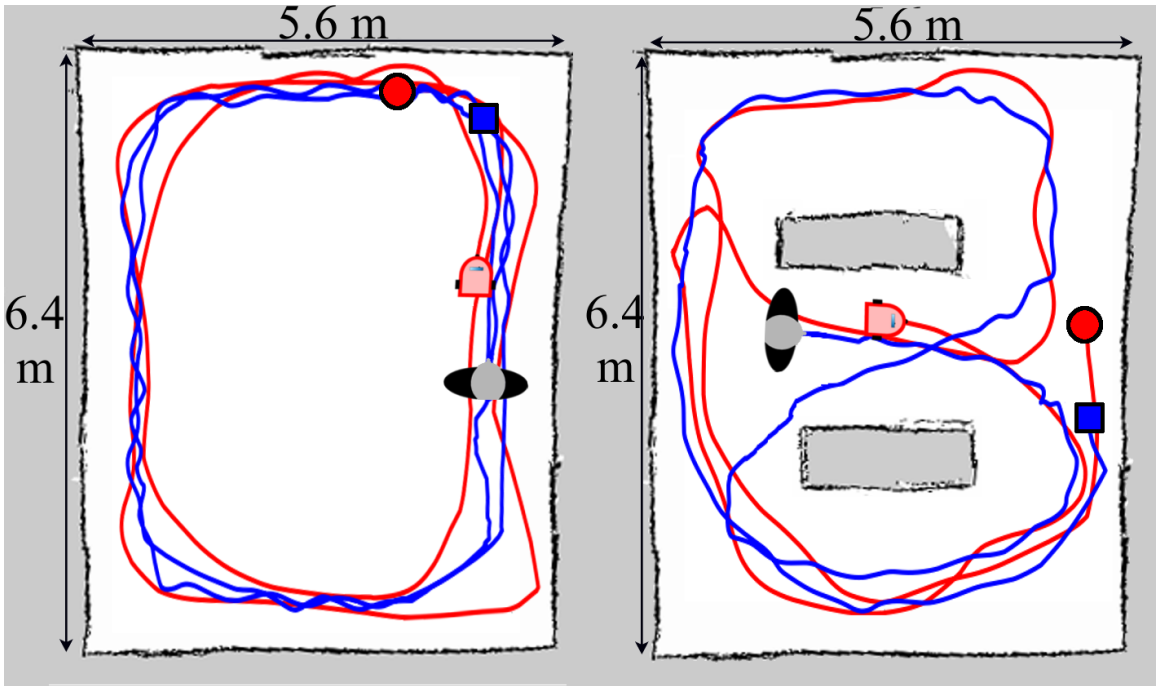


Figure 2.5: Experiment settings, Left: 'Easy', Right: 'Difficult'. The red line is the trajectory of the robot and blue that of the user. Person and robot markers represent their respective start positions. The blue square is the final position of the person and the red circle is the final position of robot

2.4 Discussion

The 'Difficult' setting has a significant number of sharp turns located close together. Either the person or the robot (or both) are turning almost all the time. When the user starts to turn down the middle passage the robot is usually already past the turning, having incorrectly predicted the user's trajectory. Once the human is seen to start turning, the robot quickly updates its user model and travels back to be in front of the person. Since the robot is limited to safe speeds close to people, the human has to wait for it to catch up. Fig. 2.8 shows this behavior. This is the cause of the long waiting times for this setting. The robot is usually able to recover after overshooting and moves ahead of the user again. There were 2 cases in the 'Easy' setting and 4 cases in the 'Difficult' setting (not included in our analysis) where the robot made an erroneous prediction (causing it to turn in the wrong direction), and the user then falls out of the robot's FOV. As a result, the robot strayed away from the user's intended path. Additional error checks on the predicted goal can help remedy this problem which will be explored in future works.

Fig. 2.6 and 2.7 visualizes the robot's performance in one trial for each of the settings. The top plot shows samples of the absolute positions of the person and robot at evenly-spaced time intervals, joined by a line segment (only one loop is shown for clarity). The middle plot shows the distance in meters between the robot and person while the bottom

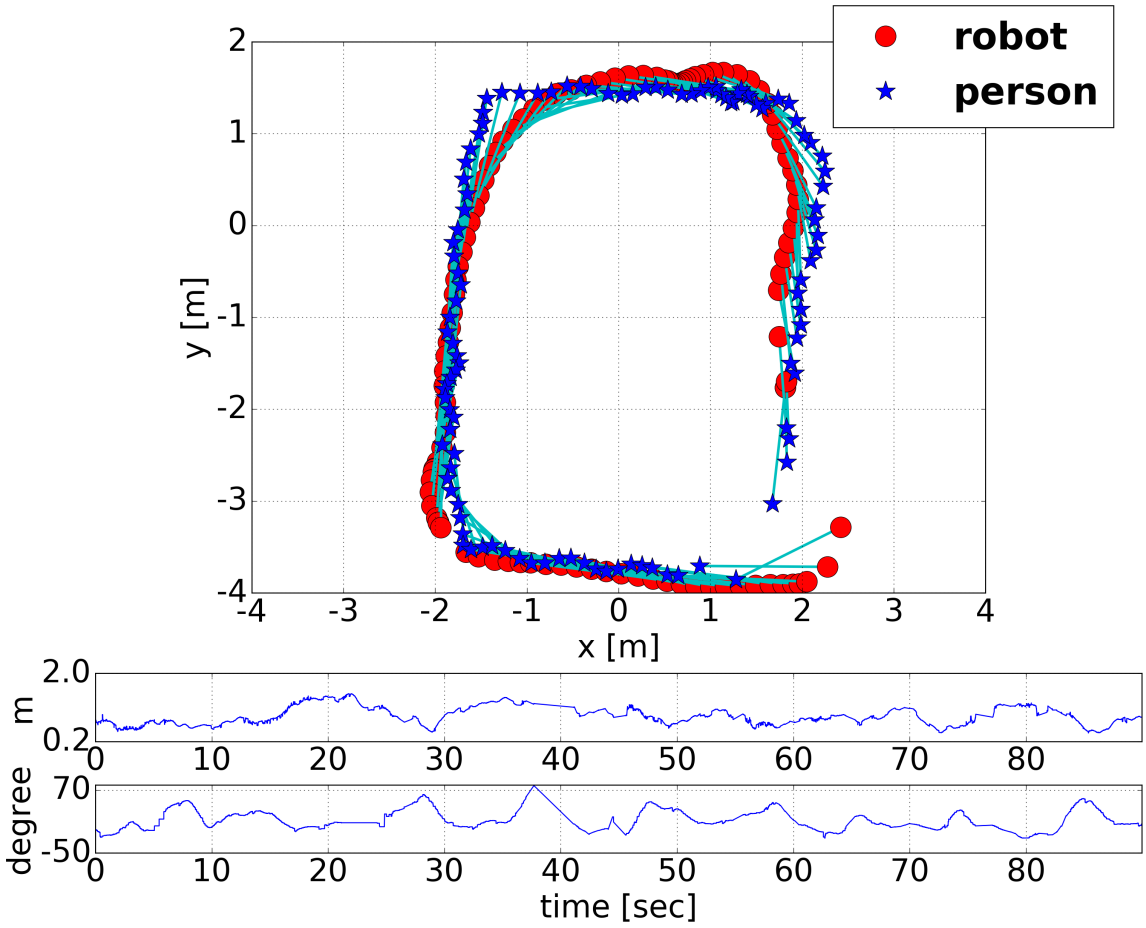


Figure 2.6: Evaluation of 'Easy' setting. Top: cyan lines show the corresponding simultaneous positions of robot (red dots) and person (blue stars) over time. Middle: Distance between the robot and person over time. Bottom: Bearing angle of the person with respect to the robot over time.

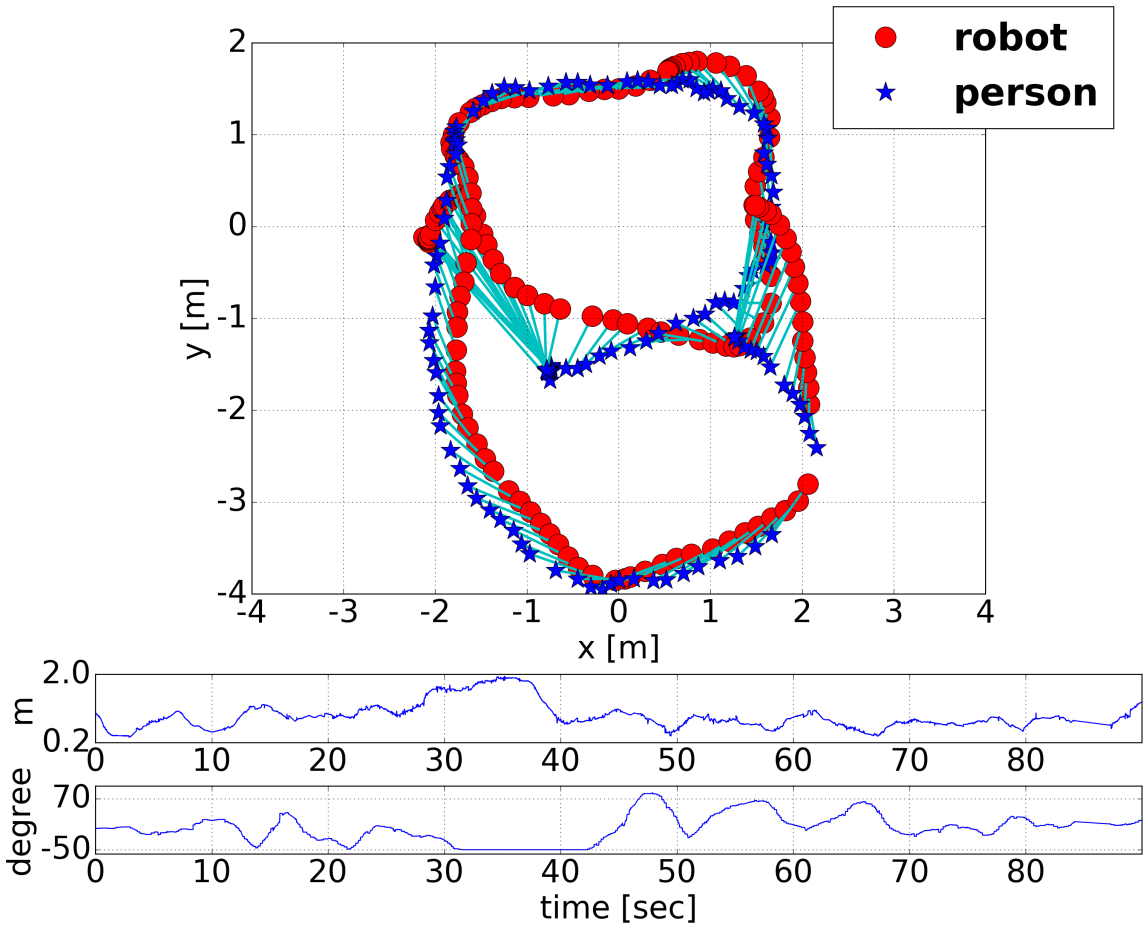


Figure 2.7: Evaluation of 'Difficult' setting. Top: cyan lines show the corresponding simultaneous positions of robot (red dots) and person (blue stars) over time. Middle: Distance between the robot and person over time. Bottom: Bearing angle of the person with respect to the robot over time (measurement between 30 to 40 seconds is missing data due to the user being out of the robot's FOV).

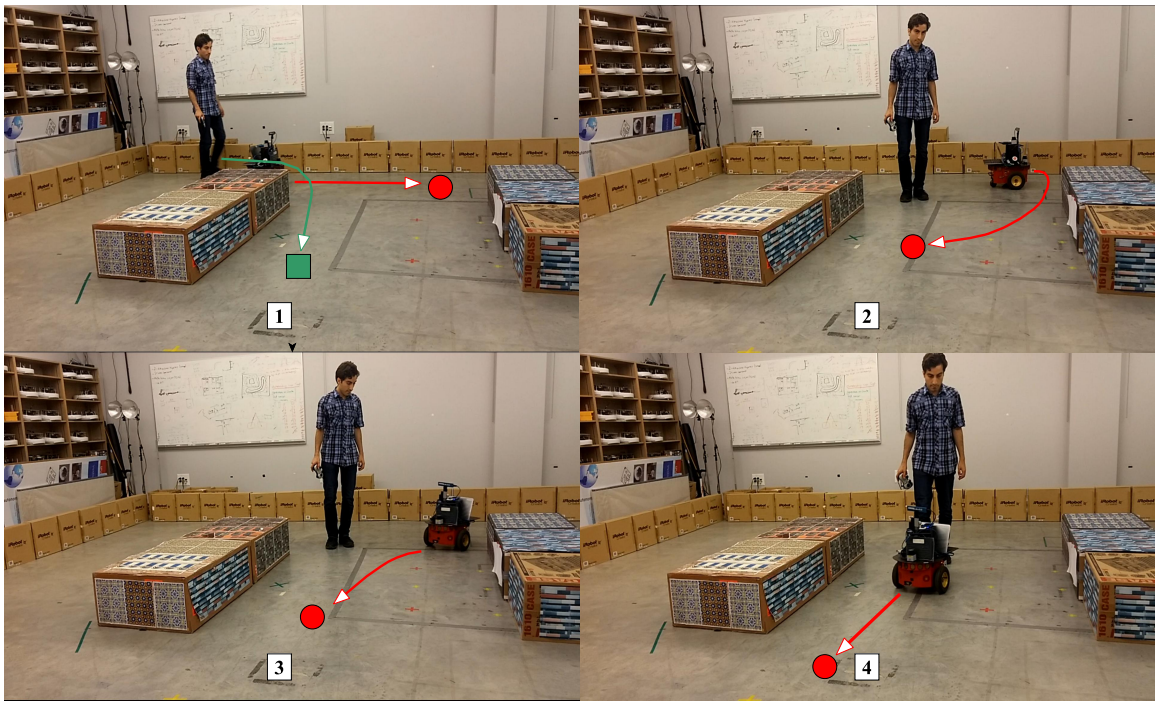


Figure 2.8: User waiting due to incorrect motion prediction and recovery (Red circle: goal point estimated by motion model. Green square: actual goal position of the user.) [1] Robot is following-ahead just before the human starts to turn to his right. [2] Robot perceives updated goal position along the new walking direction of the person [3] Subject pauses briefly, waiting for the robot to get in front [4] Front-following resumes.

plot shows the bearing angle of the person with respect to the robot. The trajectory and distances were obtained from a Vicon motion capture system while the bearing information was obtained from the estimated relative positions.

The system has consistent performance except in cases where the off-the-shelf path planner fails to navigate to a valid goal position. We conducted additional experiments in a real building hallway with two dogleg intersections using both wide and narrow FOV modes of detection. No ground truth data from a motion capture system were available, but the trials can be seen in supplementary videos[4].

2.5 Conclusion and Future Work

In this chapter, we designed a follow-ahead system. This system improves the previous work by using a motion model that considers the heading of the user and obstacles in the environment. We evaluated our follow ahead system in easy and hard settings.

In future work, we aim to integrate gesture recognition and other cues to resolve ambiguity in user intention which can happen when the walking direction is normal to an upcoming obstacle surface. There is also room for improvement in our motion model. For instance, we could remove the piecewise linearity of obstacle constraint. The user's heading could possibly be anticipated from the gaze direction like in the previous study by Zhang and Vaughan, (2016) [48].

[4]http://autonomylab.github.io/following-ahead

Chapter 3

Learning to Describe Local Occupancy Maps

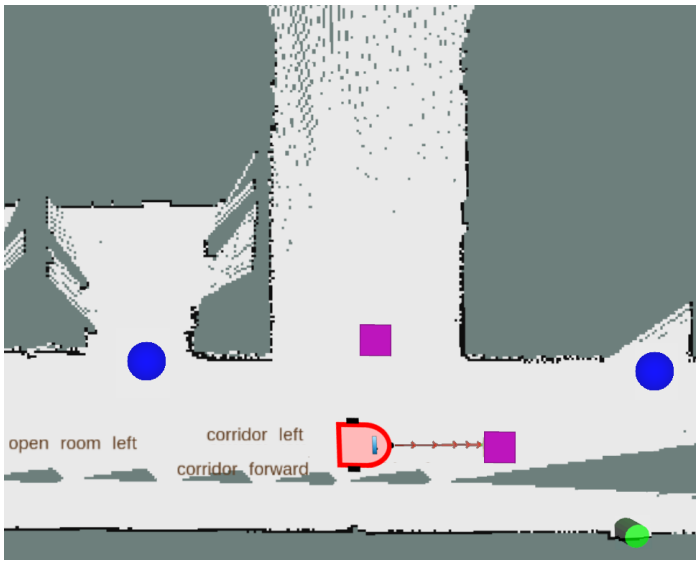


Figure 3.1: An example of a mobile robot describing the navigational options as it traverses through an indoor environment. Navigational options include the positions of open-rooms, closed-rooms or paths in an intersection.

Usage of language to communicate between human and robot may facilitate human-robot interaction. For example, in the previously discussed front following system, if the robot conveys its decision while turning at an intersection, it can potentially get feedback from the user. This feedback would help the robot avoid making wrong decisions and wasting time recovering from them [30]. Additionally, the robot can inform the user about risks involved in their intended path or propose an alternative route. This guidance may be especially useful in designing robotic systems for visually impaired people to navigate in an unknown environment.

A variety of methods can be used to localize the robot in a known map using LRF and odometry sensors, e.g. Adaptive Monte Carlo Localization [15]. Using the location of the robot in an annotated prior map containing the locations of hazards, doors and intersections, the robot can warn the user not to enter dangerous places. Such robots can also be useful to guide visually impaired users toward their destination or help them better understand the path by providing the navigational options such as the positions of doors or intersections.

However, if the robot is located in an unknown unmapped environment, we can exploit machine learning methods to recognize different environmental features. For instance, Capi analyzed 2-dimensional (2D) LiDAR data using the clustering technique to detect obstacles, steps and stairs and make it possible to warn the user about collisions or hazards [4].

In this chapter, we use CNN to detect open-rooms, closed-rooms and intersections around the robot in an unknown environment. We build a system to describe the navigational options ahead of the robot using a combination of 2D LiDAR data and occupancy grid maps. To do so we generated a dataset of robot's sensor readings where each observation is labelled with multiple potential target objects and their positions. We trained three neural network models to classify and locate the target classes around the robot. The model's predictions were tracked to locate the target classes more precisely which enables us to describe the navigational options for the user (e.g. open-room on the left). We experimentally evaluated our system in the Stage robot simulator [41] and in the real world.

3.1 Related Work

In the first part of this section, we survey the studies which facilitate navigation for visually impaired people. Then we review studies in which a robot gives environmental cues to the user. Finally, we review the research that focuses on object detection using LiDAR data.

3.1.1 Assistive Robot for Visually Impaired People

Several studies have proposed robotic systems to facilitate the indoor navigation of visually impaired people [3, 4, 31]. In a study done by Capi, a robot equipped with two LRFs and a camera mounted on a trolley walker, helps visually impaired people navigate in an unknown environment [4]. In this work, LRFs scan the environment ahead of the user in both horizontal and vertical planes to detect hazards such as stairs, obstacles and steps. In our work, we are trying to design a similar application that can detect and describe navigational cues, such as intersections, open-rooms, and closed-rooms, using LRF and occupancy map data.

3.1.2 Robot Describing the Path

While there has been a lot of research on robots which can understand natural human commands [39, 46, 49], few studies have focused on robots which can translate their world

observations to descriptions or instructions. Such descriptions can open new trails to analyze the robot's observations and its strategies. It can also be useful in the context of human-robot interaction.

Skubic et al. investigated spatial semantic models for human-robot dialogue. The robot describes the spatial relation of objects with respect to itself [36]. Daniele et al. proposed a navigational guide system able to generate instructions for navigating from point A to point B given a known map [6]. They first define a formal language named Compound Action Specification (CAS) to formulate available information in the map. They then exploit a sequence-to-sequence Recurrent Neural Network (RNN) to generate instructions close to natural language. They compared instructions produced by their system with human-generated instructions.

In the context of multi-agent systems where agents collaborate towards a goal, communication plays a significant role. Andreas et al. formulated the problem of interpreting the policy of agents as translating their messages to human language. They introduce criteria for good translations and build a translation model that facilitates collaboration between trained agents and the user [1].

3.1.3 Object detection with LiDAR scanner

Object detection using images has been greatly studied in the field of computer vision. Among the current object classification systems, YOLO 9000 introduced by Redmon and Farhadi had considerable impact due to its real-time responsiveness [35]. Our aim in this study is to do object detection and classification using 2D LiDAR data, rather than images. Although LRFs are more expensive than cameras, they have several benefits. First, LRFs may be a better choice when users are concerned with their privacy. For instance, an autonomous vacuum cleaner with a camera would concern more people than the one with an LRF because of privacy concerns. Second, in many autonomous robot applications LRF is already being used for the robot's navigation.

Various studies have used 2D [4, 10, 33] and 3D [13, 14] LiDAR scanners for object detection. Among these studies, 3D LiDAR scanners on mobile vehicles are able to collect more accurate and efficient 3D information about the surrounding environment. This type of mobile LiDAR scanning has gained popularity in road mapping studies. In a review, Guan et al. report recent studies which use LRF data to detect and extract road surfaces, on-road structures and pole-like objects [14]. Also, 3D LiDAR classification is being used in ecological analysis. For instance, Guan et al. proposed a tree classification method to classify different species of trees using mobile LiDAR data [13].

On the other hand, 2D LiDAR data is usually being used to classify locations (such as corridors, doorways or different rooms [10, 33]) or detect big environmental objects (such as stairs, steps or obstacles [4]). Goeddel and Olson used CNN models to classify objects in the environment taking 2D LiDAR data as the model's input [10]. Likewise, in another

study, Pronobis and Rao propose a probabilistic framework using Sum-Product Networks (SPNs) and deep learning to classify locations in an autonomous robot application [33]. Although these two studies are similar to the research we present in this chapter, they are mapping each 2D LiDAR scan to only one target class. In contrast, we detect multiple target classes from every data frame (including LiDAR and Gmap local-map) using our fully convolutional network. We also apply our tracking module to improve the accuracy and position estimates of detected target classes as the robot moves through corridors. In addition, we compare the usage of either or both of the 2D LiDAR scans and occupancy grid local-maps.

3.2 Approach

Given a sequence of a robot's sensor data (2D LiDAR data, IMU and odometry), we create and annotate a map of the environment while describing the navigational options for the user. Our system can be applied when designing a robot which can guide a user to navigate in an unknown indoor environment. This system could provide feedback about positions of closed-rooms, open-rooms and paths in an intersection. For instance, the robot can interact with the user by sentences similar to "Open-room on the right" or "corridors on the left and right" (for a three-way intersection with two navigational options). To achieve this, as the robot navigates through indoor maps, it should detect and track the positions of doors and intersections.

3.2.1 System Overview

Our system is implemented in Robot Operating System [34] (ROS) and tested in both simulation (Stage [41] robot simulator) and the real world (using a Husky robot[1]).

The laser scans are obtained from a Sick LMS-111 LiDAR with 270° field of view. The raw laser array contains range and bearing for each laser from LRF, which are converted to a 2D local-map (Section 3.2.3) and used as the first input to our neural network.

As the robot navigates in an unknown environment, it uses the ROS implementation of grid mapping (GMapping) for SLAM [12] to build an occupancy grid map of the environment. From now on, we will call the occupancy grid map that is created by performing SLAM "GMap". This occupancy grid map is created incrementally using the odometry data, the inertial measurement unit (IMU) data and 2D LiDAR data. A local-map from this occupancy grid map (by cropping the 16-by-16 meter map around the robot) is used as the second input to our neural network. The robot also uses this occupancy grid map to track and improve model detections. To navigate in the environment, the robot uses the

[1] `https://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/`

ROS navigation stack[2] along with a trajectory planning module based on Dynamic Window Approach [8].

We use a combination of convolutional, Batch Normalization and ReLU layers with residual connections as our network architecture (see Section 3.2.4 for more details). The inputs of our model are the laser local-map (a local-map from laser data) and the GMap local-map (a local-map from GMap).

On top of this network, our system uses a tracking module which aggregates predictions from individual frames and locates the target classes more precisely. The tracked predictions are then passed to our describe module which mentions the navigational options to the user.

3.2.2 Dataset

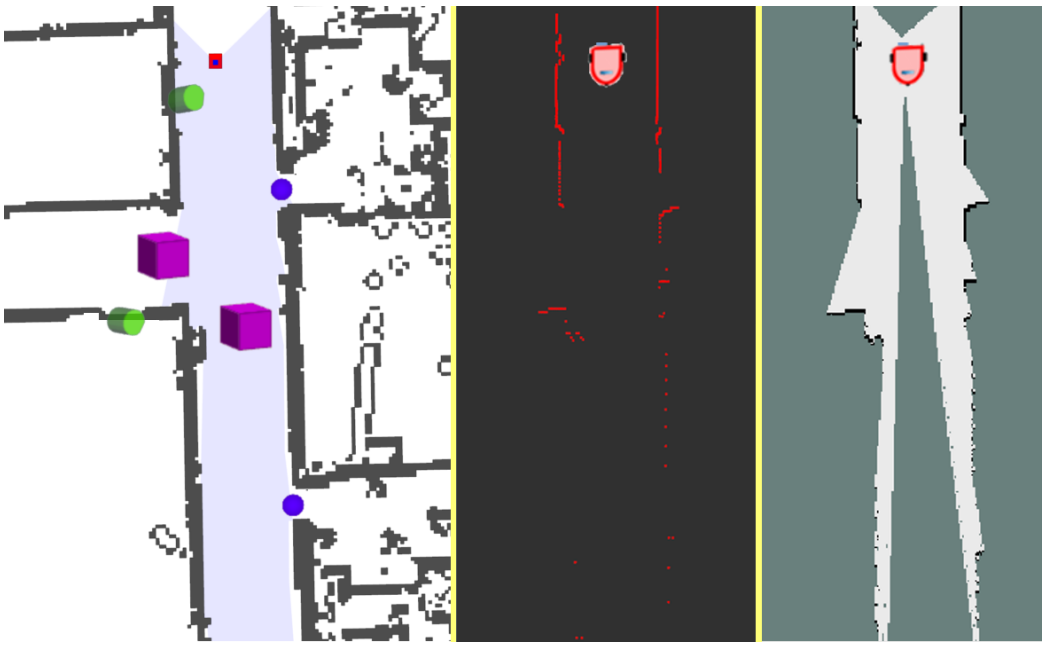


Figure 3.2: Visualization of one frame of data during dataset generation. Left: visualization from the Stage simulator containing ground-truth annotations of closed-rooms (green cylinders), open-rooms (blue spheres) and the start of each corridor in each intersection (purple cubes). Center: visualization of 2D LiDAR data in RViz. Right: GMap local-map.

A dataset suitable for our specific problem was not available, thus we generate our own dataset. This dataset contains the raw laser data (an array containing angle of each hit and its distance from the LRF), current local-map (a 16-by-16 meter map around the robot) from the GMap and also positions and labels of target classes (navigational objects). To gather the data, we use the Stage robot simulator [41] loaded with real-world occupancy grid maps. We include six preexisting occupancy grid maps [18, 28] (with small modifications)

[2] `http://ros.org/wiki/navigation`

and two new ones which were created by performing SLAM in the Applied Science building at Simon Fraser University. Figure 3.2 visualizes one frame of the dataset generation using the Stage robot simulator and RViz[3] (a 3D visualization tool for ROS). We convert the raw laser data to a 2D representation of obstacles around the robot before feeding it to our neural network (Section 3.2.3). For each map, we annotate the open-rooms, closed-rooms and the beginning of each corridor at all the intersections. The total number of target labels in these eight maps is shown in Table 3.1. To generate paths, we divide the maps into train and test. Two of our maps are used only for training, one map only for testing and the other five divided into two parts. Then, while we move the robot through a train or test map, we record the coordinates of the robot until it explores every corridor in that map at least two times. These trajectories are later used to generate the train and test data.

Table 3.1: Total number of annotations for each one of our target classes.

Closed Room	Open Room	Corridor	Total
232	121	95	448

For accurate data gathering, we use the ground truth position of the robot inside the preloaded occupancy grid maps for navigation and finding the positions of annotated classes around the robot. While the robot is navigating in one of these recorded trajectories, it simultaneously performs SLAM (using ROS GMapping package [12]) to create an occupancy grid map of the environment. GMapping is used to save a realistic local-map of the environment similar to the real world.

Having this setting for each map, we begin our dataset generation by randomly picking a short or long trajectory and spawned the robot at the beginning of that trajectory. Then we randomly choose a point in a radius of 0.5 meter of the next point in the trajectory which is not inside an obstacle (to have a different result each time the robot gets to the same point). Afterward, we use the ROS navigation module to traverse to the selected point. Whenever the map is updated by GMapping, we save the local-map (16-by-16 meter map around the robot in the GMap's occupancy grid), current laser data and target labels withing the local-map. We include corridor class labels in the predictions only if they refer to a corridor other than the one the robot is currently in. In other words, we find all the navigational options ahead of the robot. For instance, if the robot is at a three-way intersection it should only detect two paths. To achieve this, in the data annotation, we save the angle of the line connecting the beginning of each corridor to the center of that intersection. By comparing this corridor angle with the robot's orientation, we can eliminate the current corridor.

Whenever the robot approaches the current navigational goal, the planner provides the next point in the trajectory. When the robot reaches the last point in the current trajectory, we reset the GMap and start the whole process again by choosing a new trajectory. The

[3]http://wiki.ros.org/rviz

Table 3.2: Number of labels per target classes in the train, validation and test portion of the dataset.

	Closed Room (%)	Open Room (%)	Corridor (%)	Total
Train	29	46	25	33316
Validation	29	46	25	6663
Test	28	39	33	15610

robot is made to traverse these trajectories multiple times because every time the robot gets to the same map coordinates the GMap can be different. Moreover, since we choose these points from the radius of 0.5 meter of the original trajectory, the robot will be at different coordinates even if it tries to follow the same trajectory. During this data generation, we save the map in front of the robot, where the robot is always in the center of the map with the same rotation (in 90° orientation).

3.2.3 Data Augmentation

To make the model more robust to unseen data, we apply data augmentation which includes rotating, translating and re-sizing the local-map images (both the laser and the GMap local-maps). We also add different orientations of the door for the open-room class.

3.2.3.1 Rotation, Translation or Re-size

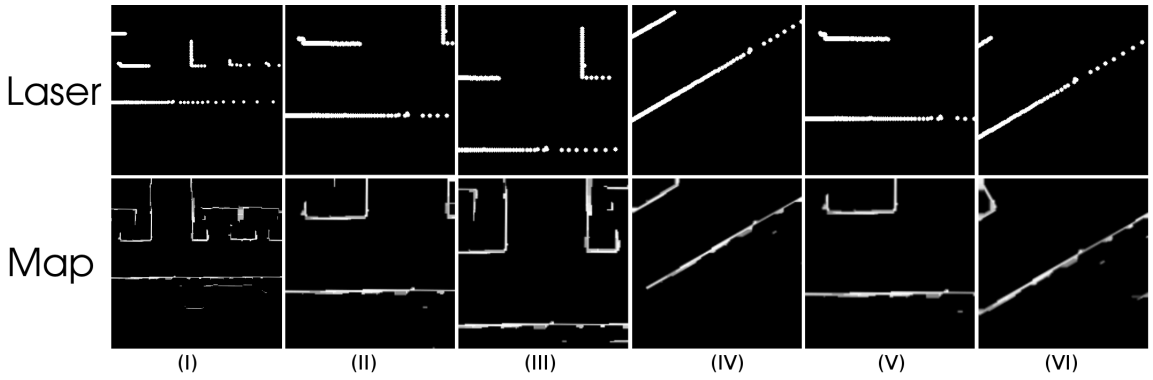


Figure 3.3: Data augmentation process. Top: the laser local-maps, Bottom: the GMap local-maps. (I) the original 16-by-16 meter local-maps, (II) the cropped 8-by-8 meter local-maps centered at the robot, (III) translation of +1.6 meter in both x and y directions, (IV) rotation by -30°, (V) resize by 1.2 times and (VI) the obtained augmentation by applying these operation in order. (III) to (VI) are cropped 8-by-8 meter local-maps.

During training, whenever we fetch a GMap local-map or a 2D LiDAR data from the dataset, we randomly rotate, translate or re-size them. Figure 3.3 shows the possible data augmentation operations. The resulting images are fed into our neural network.

The primary GMap local-map is saved with dimensions twice as big as the images the network needs (the original image dimensions is a 16-by-16 meter which is cropped to an

8-by-8 meter map for our network) which helps to do the data augmentation by preventing unknown pixels in the final map. In the end, we crop (8 meters ahead of the robot in the x direction and ±4 meters in the y direction relative to the robot's orientation) and re-size the map to the desired network input size (244x244 pixel). The same operations (rotation, translation and re-size) are also applied to the laser scan data by first converting the 1D laser array to 2D local-map coordinates. To do this, we perform the following operations:

We calculate an angle "α" and distance "d" for each element of the laser array (LA) based on the FOV of the laser, max range of the laser (MR) and element index "i" in LA.

$$Steps = \frac{FOV}{len(LA)} \tag{3.1}$$

$$\alpha = (\frac{len(LA)}{2} - i) * Steps, \;\; d = LA[i] \tag{3.2}$$

Converting the laser coordinates(d, α) to the local-map (X, Y) coordinates(MSM is desired local laser Map Size in Meter, MSP is desired Map Size in Pixel and TMC stands for "To Map Coordinates"):

$$TMC = \frac{MR}{MSM} \tag{3.3}$$

$$X = d * \cos(\alpha) * MR, \;\; Y = d * \sin(\alpha) * MR \tag{3.4}$$

$$X = X * TMC, \;\; Y = (-Y + \frac{MSM}{2}) * TMC \tag{3.5}$$

Resizing the map coordinates(X, Y) by the resize factor (RF):

$$X = X * RF \;\; , Y = Y * RF + (1 - R) * 0.5 * MSP \tag{3.6}$$

Rotating it by the angle β (Rotate the X,Y coordinates with origin x=0 and y=MSP/2 in the image frame coordinate):

$$X = \cos(\beta) * X - \sin(\beta) * (Y - \frac{MSP}{2}) \tag{3.7}$$

$$Y = \frac{MSP}{2} + \sin(\beta) * X + \cos(\beta) * (Y - \frac{MSP}{2}) \tag{3.8}$$

Translating it by Tx, Ty:

$$X = X + Tx * MSP, \;\; Y = Y + Ty * MSP \tag{3.9}$$

3.2.3.2 Adding New Doors' Orientations

In this step, we create new maps by modifying the previous maps. First, we annotate the position and width of each door panel in open-rooms; then remove the door panel from

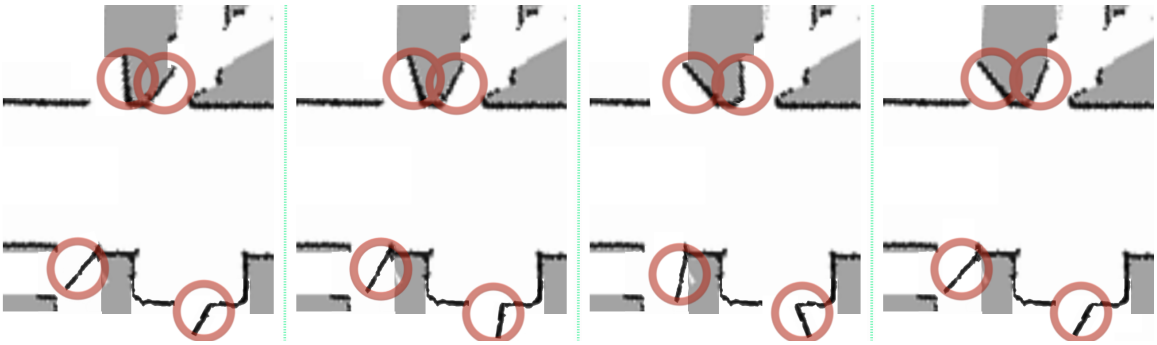


Figure 3.4: Four different orientations of four doors. Doors are annotated by red circles. During data augmentation, we add different orientations for the door panels to avoid overfitting. If we call the angle between door panel and door frame α; for open-rooms: $30 < \alpha < 100$.

occupancy grid maps. Next, we generate new maps by randomly adding open-rooms. For open-rooms, we assume that the angle between the door frame and door panel is between 30° to 100°. An example for four different doors' orientations illustrated in Figure 3.4.

3.2.4 Architecture

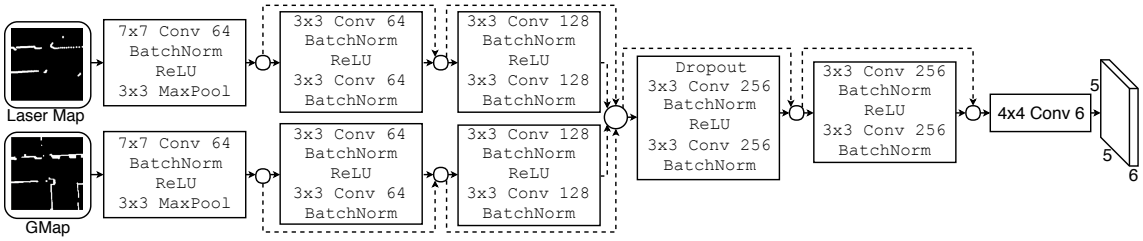


Figure 3.5: Our network architecture is a fully convolutional model with residual connections. The inputs are two local-map images, 244×244, one generated from a 2D LiDAR and the other one from the GMap (the local-map generated by performing SLAM) during run-time. The output is a $5 \times 5 \times 6$ tensor which is the grid representation of predictions. Each grid cell contains the confidence score, coordinates and probability of target classes for that grid cell.

In this section, we describe three models we use to predict the position of the target classes around the robot. All of these three models use a similar network architecture but with different input data. We define these three models as follows:

- **Laser model**, which uses only the laser local-map
- **Map model**, which uses only the GMap local-map
- **Combined model**, which uses both GMap local-map and laser local-map

The network architecture is presented in Figure 3.5. The first part of the network consists of two parallel networks where one extracts features from the GMap local-map and the other from the laser local-map. Each part of the parallel network consists of three ResNet blocks with residual connections. Each ResNet block contains convolutional, batch normalization and ReLU layers. In the next stage, the concatenation of these two parallel network forms the input of the next two ResNet blocks followed by a 4×4 convolutional layer.

3.2.5 Implementation details

Our architecture is inspired by ResNet34 [16] and YOLO9000 [35]. The inputs of our network are two 244×244 pixel images of the surrounding environment, belonging to a laser local-map or a GMap local-map. Inspired by YOLO methodology, we divide each input image into a 5 by 5 grid (referred to as g). For each grid cell we define several properties including confidence score, coordinates and the probability of target classes. Therefore, the model predicts a $5 \times 5 \times 6$ tensor, where the numbers along the 3rd axis correspond to predicted properties of each grid cell. Each vector $v_{i,j,*}$ is responsible for detecting the object which resides in the grid cell of $g(i, j)$ with six properties: confidence score, (x, y) coordinates and conditional class probabilities for the three target classes. The confidence score is the estimate of model certainty about whether the center of an object resides in this grid cell. The (x, y) coordinates are the positions of objects' center relative to the top-left corner of the grid cell in the input image ($0 < x, y < 1$). The last three conditional class probabilities $Pr(Class_i|Object), i = 0, 1, 2$ are the probabilities of each target class if an object of class i exists in this grid cell. In our case, object classes are open-room, closed-room and the beginning of corridors. To calculate the final probability of each target class, we multiply the conditional class probabilities of each target class by the cell confidence score:

$$Pr(Class_i) = Pr(Class_i|Object) * Pr(Object) \tag{3.10}$$

To train our network to predict the $5 \times 5 \times 6$ tensor, we use the combination of Cross Entropy loss and L2 loss (mean squared error). The Cross Entropy loss (X_{ent}) is used to learn the conditional class probabilities and the L2 loss used to learn both the coordinates position (ℓ_2^p) and confidence score (ℓ_2^c) of each target classes. Our final loss function is a weighted sum of these losses:

$$\mathcal{L} = \alpha_1 * X_{ent} + \alpha_2 * \ell_2^p + \alpha_3 * \ell_2^c \tag{3.11}$$

where,

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \tag{3.12}$$

In these equations, α_1, α_2 and α_3 are weights of the loss functions. We set $\alpha_1 = 0.61$, $\alpha_2 = 0.14$ and $\alpha_3 = 0.25$, although we tested other values empirically and observed minor

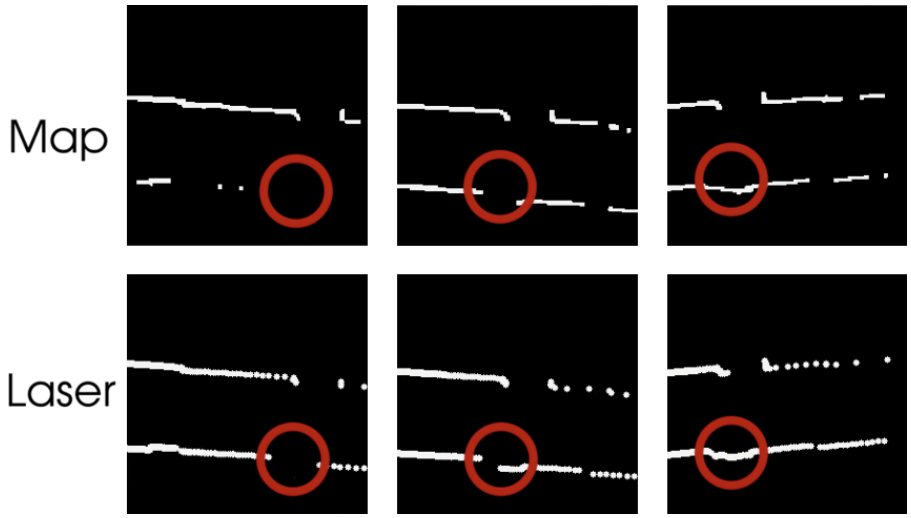


Figure 3.6: An example of local-maps when the robot approaches a closed-room. The closed-room is annotated by a red circle. The closed-room becomes clearer as the robot comes closer to the closed-room. At the beginning it looks like an open-room but later (on the second frame for the laser local-map and the third frame for the GMap local-map) it can be distinguished as a closed-room.

variation in performance. All three models are trained using the Adadelta optimizer [47] with the batch size of 60 and randomly initialized weights on the training dataset. During training, we evaluate the model on each epoch on the validation dataset and keep the weights of the model which has the minimum total loss. These models are implemented with PyTorch library [32] and are trained on an Nvidia GeForce GTX 1080 Ti GPU.

3.2.6 Tracking Module

In our application, a robot should be able to describe the navigational options for the user at the appropriate moments. Consequently, our system should be able to track the surrounding objects as the robot approaches them and mention the labels at the right time. If the system does not use this tracking module, it may detect an object in one frame and not be able to detect it in another one as the local-map changes.

When the robot is moving along a path and approaches a room or an intersection, the target object would be in the field of view of the robot for a few frames. In each frame, the robot sees the object with a slightly different shape. The reasons for these differences are the movement of robot, the incomplete map created by SLAM (which is being updated incrementally as the robot gets closer to the object) and noise in the robot's sensors. For example, a closed-room in some cases (especially at distance) would look like an open-room (see Figure 3.6). Open-room and corridor also look similar in many cases. Thus, the network may change its prediction when the robot approaches an object. Moreover, when the shape of the object changes, the network finds slightly different coordinates for the labels. In other

words, the system should not change the coordinates or class of an object based on a single network prediction. Instead, it should consider all the prediction labels and positions for that object.

Our tracking module works by first, saving and calculating the position of the target labels on the GMap global map (all the conversion of different coordinate system frames is done using the ROS TF package[4]). Next, we cluster the nearest target points using the k-Nearest Neighbors algorithm (with a maximum distance of one meter). Then for every target class in each cluster, we average over the final probability (obtained from Equation 3.10) and assign the most probable target to the cluster. Finally, the coordinates of each cluster is calculated by averaging the coordinates of the most probable target class for that cluster.

The tracking module relies on the global coordinate frame that is updated by the GMapping algorithm. Although the SLAM algorithm usually builds a reliable map, sometimes, it shifts the global coordinate frame or part of the occupancy grid maps. The reason for this is the cumulative errors in the odometry of the robot or delays in the sensor's data. These shifts add cumulative errors to the previous predictions of our tracking module. To reduce these cumulative errors, each point in our tracking module has a lifetime of 30 seconds after its detection.

Ultimately, using the most probable target label and position, the describe module explains the environment for the user with the following procedure. First, it finds the coordinates of the detected targets around the robot's position within five meters. Next, it calculates the angle Q between the robot-object line and the robot orientation. At last, the describe module says the target class name followed by int position around the robot (e.g. closed-room left). The position of the object with respect to the robot is defined as follows:

- left side, if -120°$< Q <$-50°
- front, if -30°$< Q <$-30°
- right side, if 50°$< Q <$120°

3.3 Results and Discussion

We evaluate our methodology by conducting three experiments. The results of each experiment are compared using three models described in Section 3.2.4. These models are trained on the training dataset (see Section 3.2.2), and the best performing model is obtained by evaluating on the validation dataset. For all three experiments, we use the same model weights. On each experiment, we report the recall, precision and F1 score which are three standard metrics for classification.

[4]http://wiki.ros.org/tf

3.3.1 Experiment One

The first experiment is conducted to evaluate the accuracy of our network without the use of the tracking module. We run each one of three models on testing dataset. Table 3.3 shows the recall, precision and F1 score of each class for all three models in the training and the testing datasets. These results show the highest recall is for open-rooms and the lowest recall is for the corridors. Open-rooms are probably easier to detect for several reasons. First, we train our network with different orientations of doors (for details see Subsection 3.2.3), which makes it more robust to open-rooms. Secondly, door frames normally have similar dimensions between different buildings. Lastly, in the case that the network has learned the relation between free spaces, it may detect an open-room as a narrow space that connects two wide open-areas which is much easier to detect compared to closed-room, which is a small recess inside the wall. Another reason behind the poor recall of the corridors is probably the variety of intersections types. For example, the size of intersection corridors can vary widely. In addition, the orientations in which corridors are connected to the intersections can differ. In some cases, the three-way intersections are very similar to a corridor with an open room.

Based on these precision results, the models are less precise at detecting closed-rooms compared to other classes. The model might falsely identify some of the recesses in the wall as a closed room, resulting in a lower precision.

Table 3.3: Recall, precision and F1 score of three models running on the training and testing dataset. The numbers indicate difficulties of detecting corridors (in the beginning of each intersection) compared to closed-rooms and open-rooms.

	Train Recall				Test Recall		
	Closed Room	Open Room	Corridor		Closed Room	Open Room	Corridor
Laser	0.84	0.91	0.77	Laser	0.72	0.85	0.5
Map	0.82	0.89	0.77	Map	0.68	0.68	0.46
Combined	0.9	0.95	0.89	Combined	0.78	0.86	0.48

	Train Precision				Test Precision		
	Closed Room	Open Room	Corridor		Closed Room	Open Room	Corridor
Laser	0.83	0.99	0.98	Laser	0.82	0.99	0.96
Map	0.74	0.99	0.97	Map	0.65	0.98	0.95
Combined	0.84	0.98	0.99	Combined	0.78	0.99	0.96

	Train F1				Test F1		
	Closed Room	Open Room	Corridor		Closed Room	Open Room	Corridor
Laser	0.84	0.95	0.86	Laser	0.77	0.91	0.66
Map	0.78	0.94	0.86	Map	0.67	0.80	0.62
Combined	0.87	0.96	0.94	Combined	0.78	0.92	0.64

We also observe that the F1 score of the Map model is lower than other two models. After investigating these results, we find that the GMap local-maps have less details compared to local-map from LiDAR data. In addition, when the robot is turning into an unknown area

of the map, the occupancy grid map will only be updated a few frames later, because SLAM requires multiple measurements for each one of the obstacles in the LiDAR view. Although GMap local-map may have less details compared to laser local-map, a combination of them in the Combined model slightly enhances the F1 score in most cases.

We measure the run-time speed of our network as it is important for a robotic application to have a fast network that can be run in real-time. Our network can run as fast as 140 Hz (around 7 milliseconds for each run) with a system equipped with a GeForce GTX 1080 Ti or 12 HZ (around 83 milliseconds for each run) using an Intel i7-7700 CPU, due to our fully convolutional architecture.

3.3.2 Experiment Two

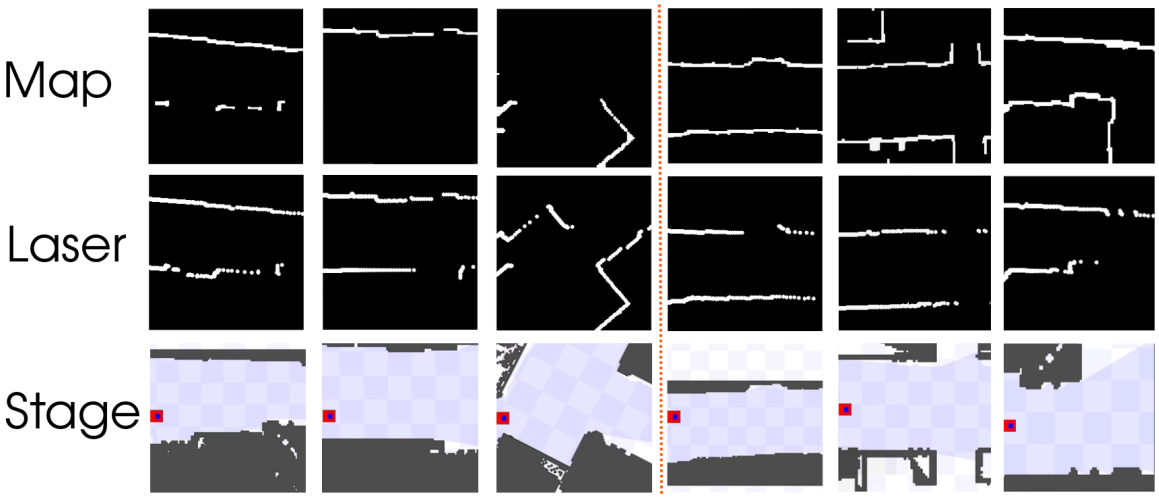


Figure 3.7: Comparison between 2D LiDAR local-maps and GMap local-maps. The top row shows the GMap local-map; the middle row, shows the laser local-maps and the bottom row is the screenshot of robot in the Stage simulator. On the left side, there are three cases in which laser local-maps are more detailed; this normally happens in an unknown map when robot changes its orientation. The right side, shows three cases that GMap data is more detailed. These cases normally happen when the robots is in a previously explored place.

In the second experiment we evaluate the combination of our three models' predictions and the tracking module. We also investigate the performance of our system in an unknown environment versus a previously explored environment. We annotate new trajectories in the previous occupancy grid maps (the maps we used for generating our dataset). Train and test trajectories are chosen from the train and test part of dataset maps. These trajectories are chosen in a way to visit the whole map with minimal crossings (See Figures 3.10 to 3.18 for the train and test trajectories). As a result, we can assume that the robot visits each part of the map for the first time (it is seeing an unknown environment).

In the unexplored experiment, the robot is initialized at the beginning of each trajectory for each map. We then choose the next point in the trajectory as the goal and the robot

navigates toward the goal using the ROS navigation module[5]. When the robot gets close to the goal, the next point would be chosen as the new goal. We repeat this process until the robot gets to the end of the trajectory. At this point, the whole map of the environment is created by SLAM GMapping. In the explored experiment, we navigate the robot to the beginning of the trajectory. The same experiment is then conducted using this known occupancy grid map.

As the robot navigates in the environment, for each update of GMapping, we use three models to predict the target classes around the robot. The tracking module then uses these predictions to update the accuracy and position of the current predictions.

[5] `http://ros.org/wiki/navigation`

Table 3.4: Recall, precision and F1 Score for all three models for experiment two in the unexplored (unknown maps) and the explored maps.

(a) Recall

	Unexplored Train				Unexplored Test		
	Closed Room	Open Room	Corridor		Closed Room	Open Room	Corridor
Laser	0.88	0.89	0.76	Laser	0.75	0.83	0.74
Map	0.78	0.86	0.69	Map	0.63	0.79	0.61
Combined	0.82	0.89	0.77	Combined	0.79	0.84	0.69

	Explored Train				Explored Test		
	Closed Room	Open Room	Corridor		Closed Room	Open Room	Corridor
Laser	0.91	0.91	0.8	Laser	0.81	0.85	0.75
Map	0.82	0.87	0.7	Map	0.72	0.83	0.69
Combined	0.9	0.93	0.79	Combined	0.85	0.9	0.75

(b) Precision

	Unexplored Train				Unexplored Test		
	Closed Room	Open Room	Corridor		Closed Room	Open Room	Corridor
Laser	0.79	0.90	0.97	Laser	0.83	0.94	0.95
Map	0.62	0.82	0.92	Map	0.73	0.80	0.87
Combined	0.86	0.94	0.98	Combined	0.80	0.92	0.92

	Explored Train				Explored Test		
	Closed Room	Open Room	Corridor		Closed Room	Open Room	Corridor
Laser	0.85	0.90	0.91	Laser	0.87	0.95	0.95
Map	0.77	0.95	0.94	Map	0.82	0.87	0.98
Combined	0.87	0.94	0.98	Combined	0.86	0.93	0.93

(c) F1 Score

	Unexplored Train				Unexplored Test		
	Closed Room	Open Room	Corridor		Closed Room	Open Room	Corridor
Laser	0.83	0.90	0.85	Laser	0.79	0.88	0.83
Map	0.69	0.84	0.79	Map	0.68	0.80	0.72
Combined	0.84	0.91	0.86	Combined	0.80	0.88	0.79

	Explored Train				Explored Test		
	Closed Room	Open Room	Corridor		Closed Room	Open Room	Corridor
Laser	0.88	0.91	0.85	Laser	0.84	0.90	0.84
Map	0.80	0.91	0.80	Map	0.77	0.85	0.81
Combined	0.88	0.94	0.88	Combined	0.86	0.91	0.83

We evaluate our system by finding the closest ground-truth point to each one of the predictions using the tracking module with a maximum distance of 0.5 meter. Because we use LiDAR and GMap local data, details of the obstacles in the local-maps improves as the robot gets closer to them. For example, when the robot is far away from a closed-room, the model may classify it as an open-room, but as it gets closer, the model changes it's prediction to closed-room. As mentioned in Section 3.2.6, the tracking module updates the probability of target classes using all the previous predictions on that region. As a result, we calculate the metrics when the points get out of the view of the robot. If the robot follows the same trajectory multiple times, the constructed local-maps (either GMap or

laser local-map) might be slightly different for the same region. To address this difference, for each setting (the explored or unexplored), we repeat the experiment 30 times and report the averaged results.

Some examples of the predictions of our system for a few maps is shown in Figures 3.10 to 3.18. In these figures, we also annotated the false-positives and false-negative predictions of our system.

The recall, precision and F1 score for each target class for unexplored (unknown) maps and explored maps is shown in the Table 3.4. The results in this table shows an increase in the F1 score of all three models using explored maps. Part of this increase is because of our tracking module. The tracking module uses GMap global frame to track the target classes. As a result, if we have a more stable GMap (as SLAM GMapping will shift less when it has the whole map), the accuracy of tracking increases. In explored maps, this increase is more pronounced in the Map model which is due to a more detailed GMap local-map. Figure 3.7 shows a few examples comparing the laser and GMap local-map in the explored and unknown maps.

Furthermore, Figure 3.8 and 3.9 depicts the histogram of the distances that the target labels predicted correctly (in the robot coordinate frame). These histograms are plotted for all three models in the unknown and explored maps. In these histograms, higher distances indicate anticipation of farther objects. Although these results do not show a considerable difference between the models, we observe that the laser model can predict a little farther in the unknown maps compared to the Map and Combined model. On the other hand, the Combined and Map models detect farther objects in the explored experiment due to usage of a complete map.

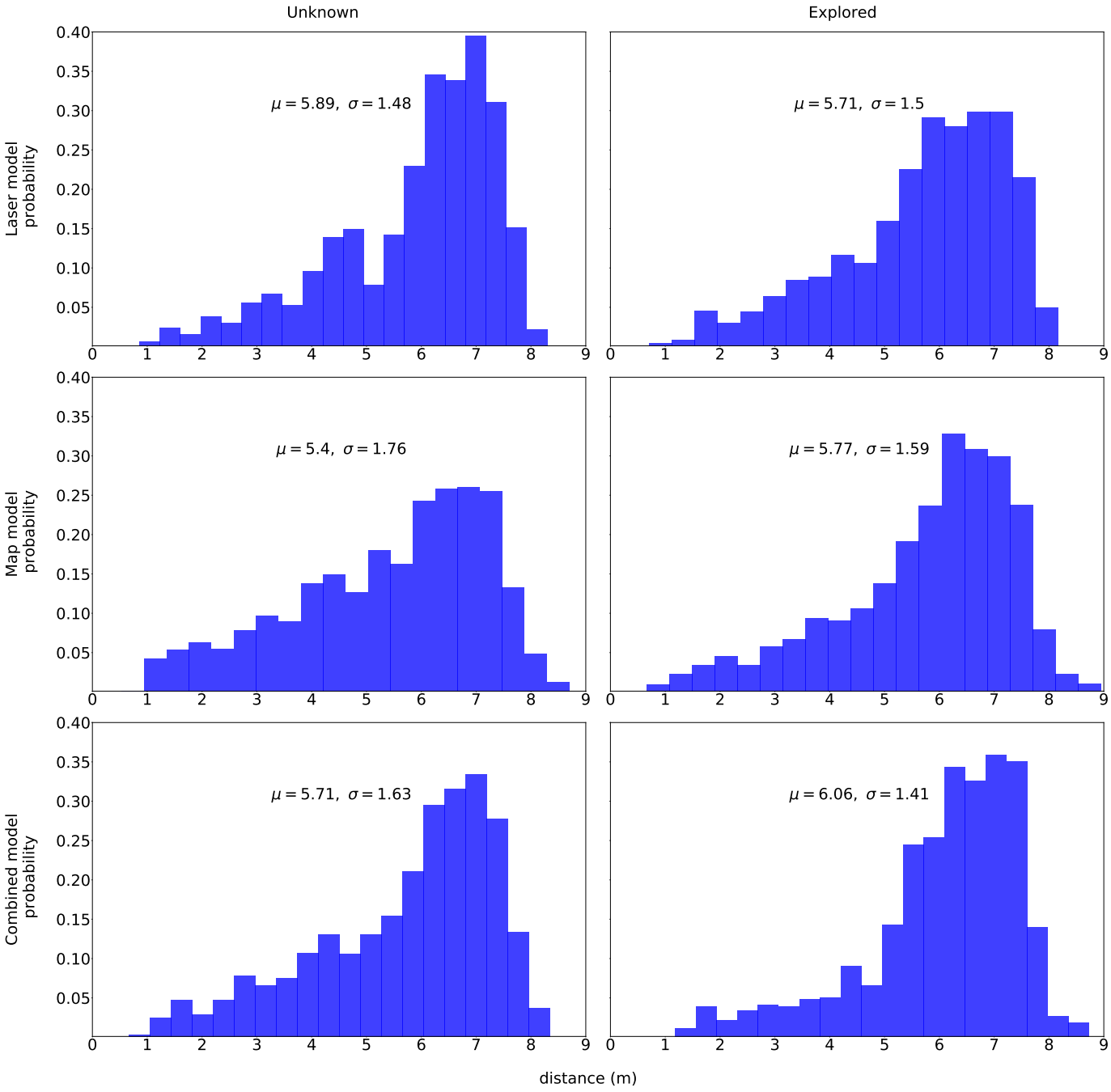


Figure 3.8: Histogram of the distances that the target labels predicted correctly on the **train** portion of maps. The left histograms are the distances for the unknown maps and the right histograms are the distances for the explored maps using all three models in experiment two. The mean and sigma of the distribution are denoted in each histogram separately.

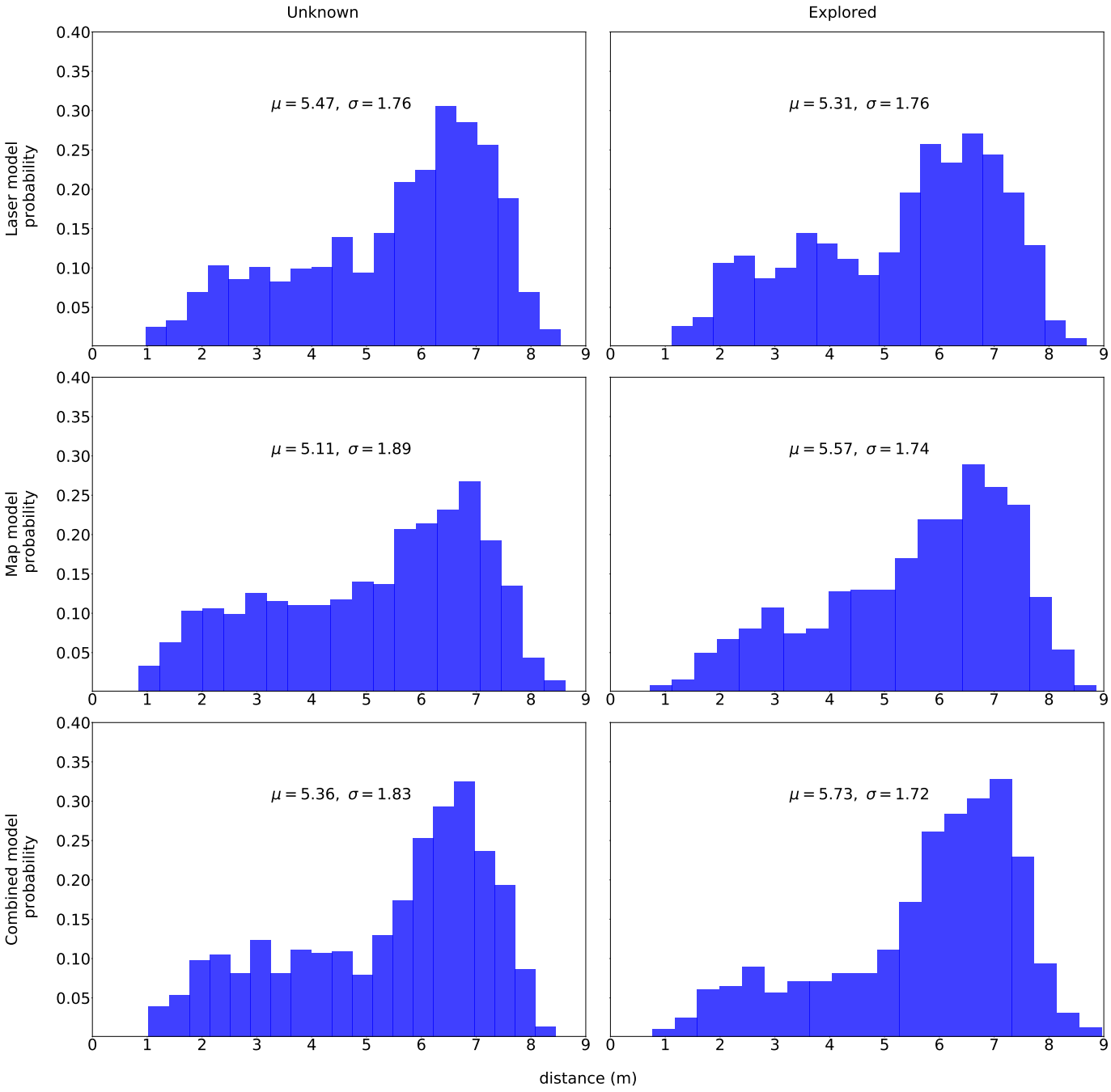


Figure 3.9: Histogram of the distances that the target labels predicted correctly on the **test** portion of maps. The left histograms are the distances for the unknown maps and the right histograms are the distances for the explored maps using all three models in the experiment two. The mean and sigma of the distribution are denoted in each histogram separately.

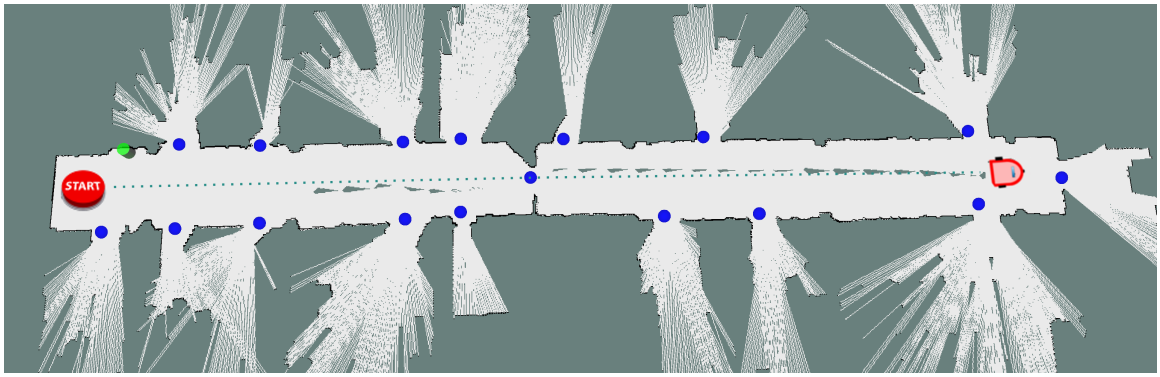


Figure 3.10: Prediction of our system for FR79 building map. This map is only used in the **train** part of the dataset. Closed-rooms predictions are annotated by green cylinders and open-rooms by blue spheres.

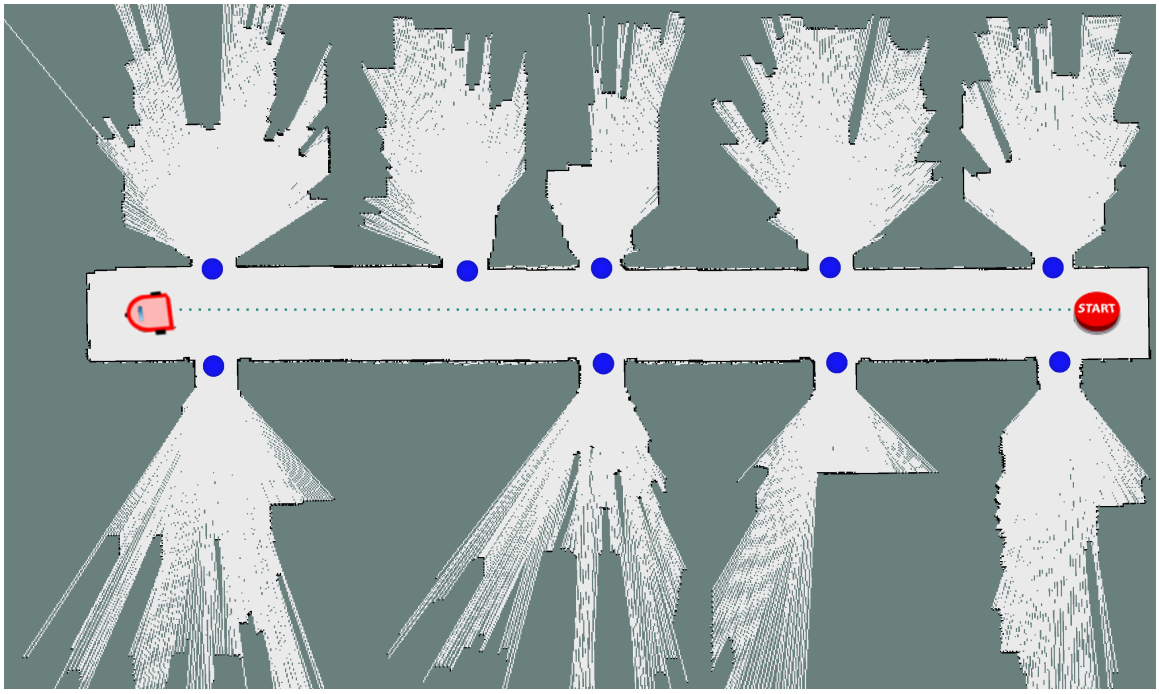


Figure 3.11: Prediction of our system for FR52 building map. This map is only used in the **test** portion of the dataset. Open-rooms predictions are annotate by blue spheres.

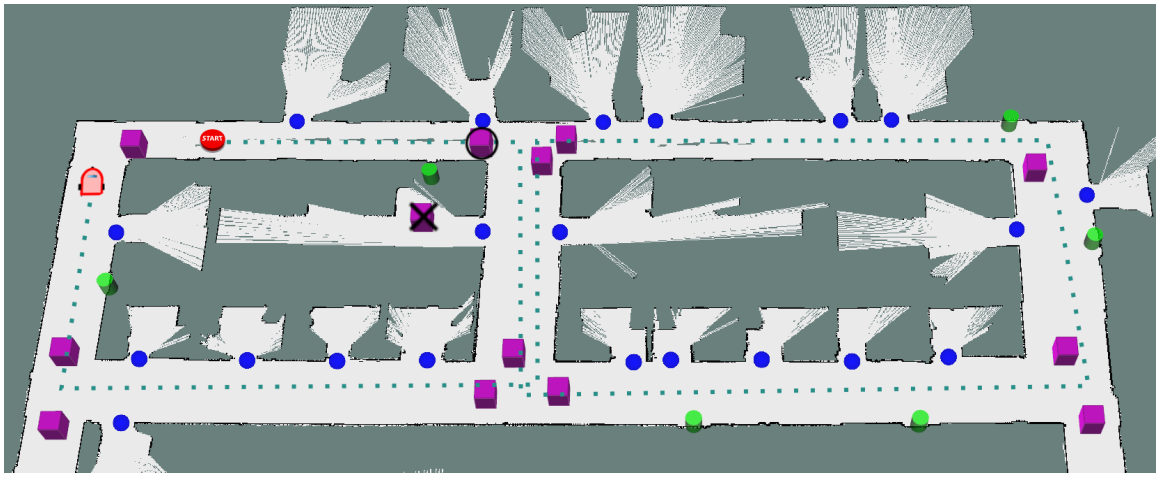


Figure 3.12: Prediction of our system for the SAIC (train portion) building map. This part of the map is used in the **train** part of the dataset. Closed-rooms predictions are annotated by green cylinders, open-rooms by blue spheres and the start of each corridor in the intersections by purple cubes. The false-positives are annotated by $\times$ and false-negatives by $\circ$ over predictions.

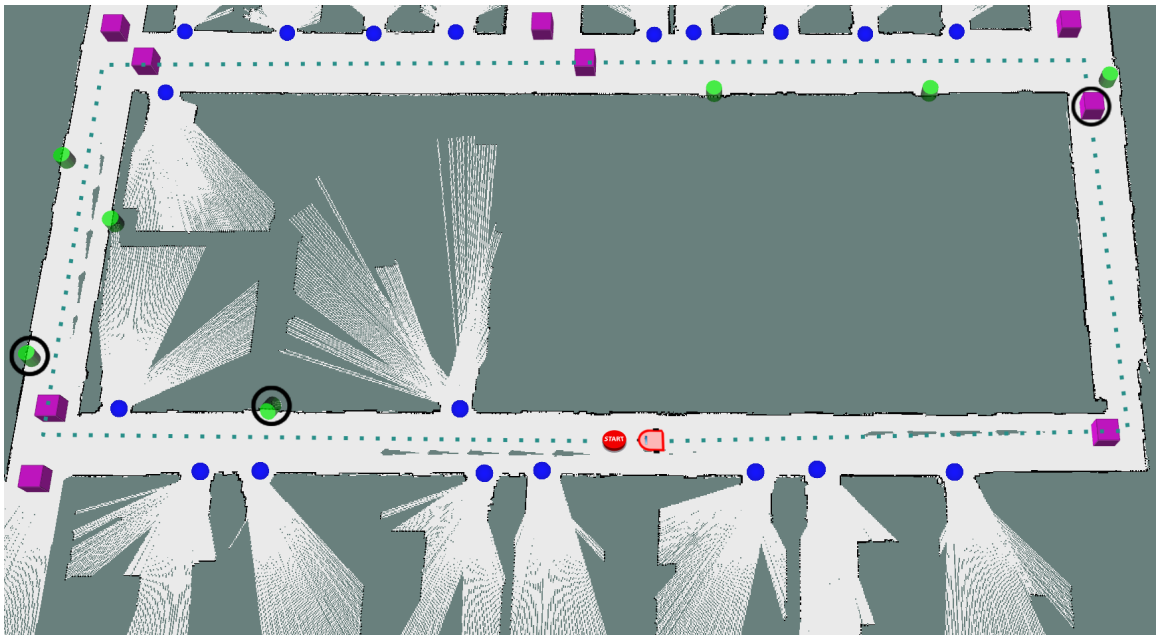


Figure 3.13: Prediction of our system for the SAIC (test portion) building map. This part of the map is used in the **test** part of the dataset. Closed-rooms predictions are annotated by green cylinders, open-rooms by blue spheres and the start of each corridor in the intersections by purple cubes. The false-positives are annotated by $\times$ and false-negatives by $\circ$ over predictions.

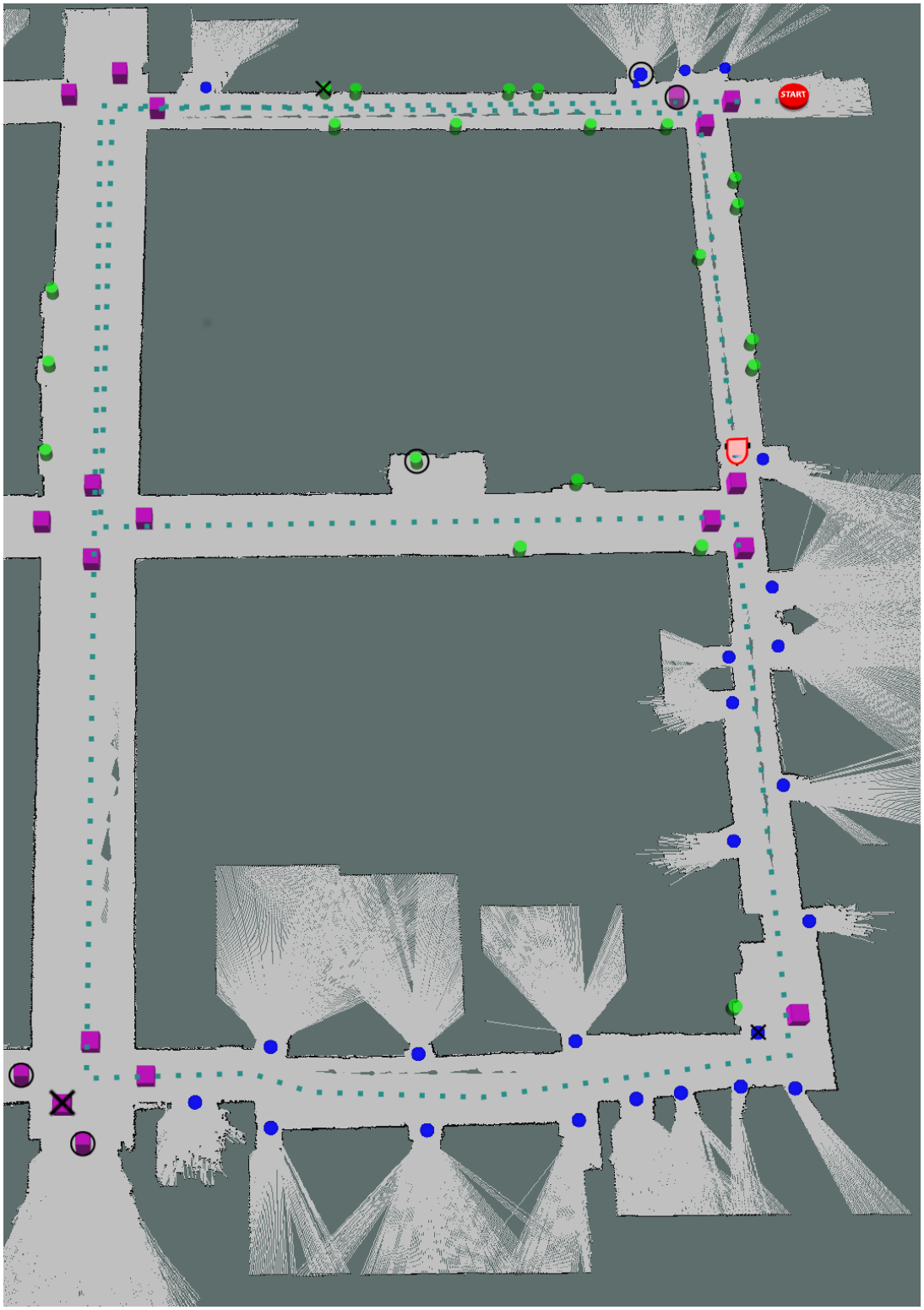


Figure 3.14: Prediction of our system for the ACES3, Austin (train portion) building map. This part of the map is used in the **train** part of the dataset. Closed-rooms predictions are annotated by green cylinders, open-rooms by blue spheres and the start of each corridor in the intersections by purple cubes. The false-positives are annotated by $\times$ and false-negatives by $\circ$ over predictions.

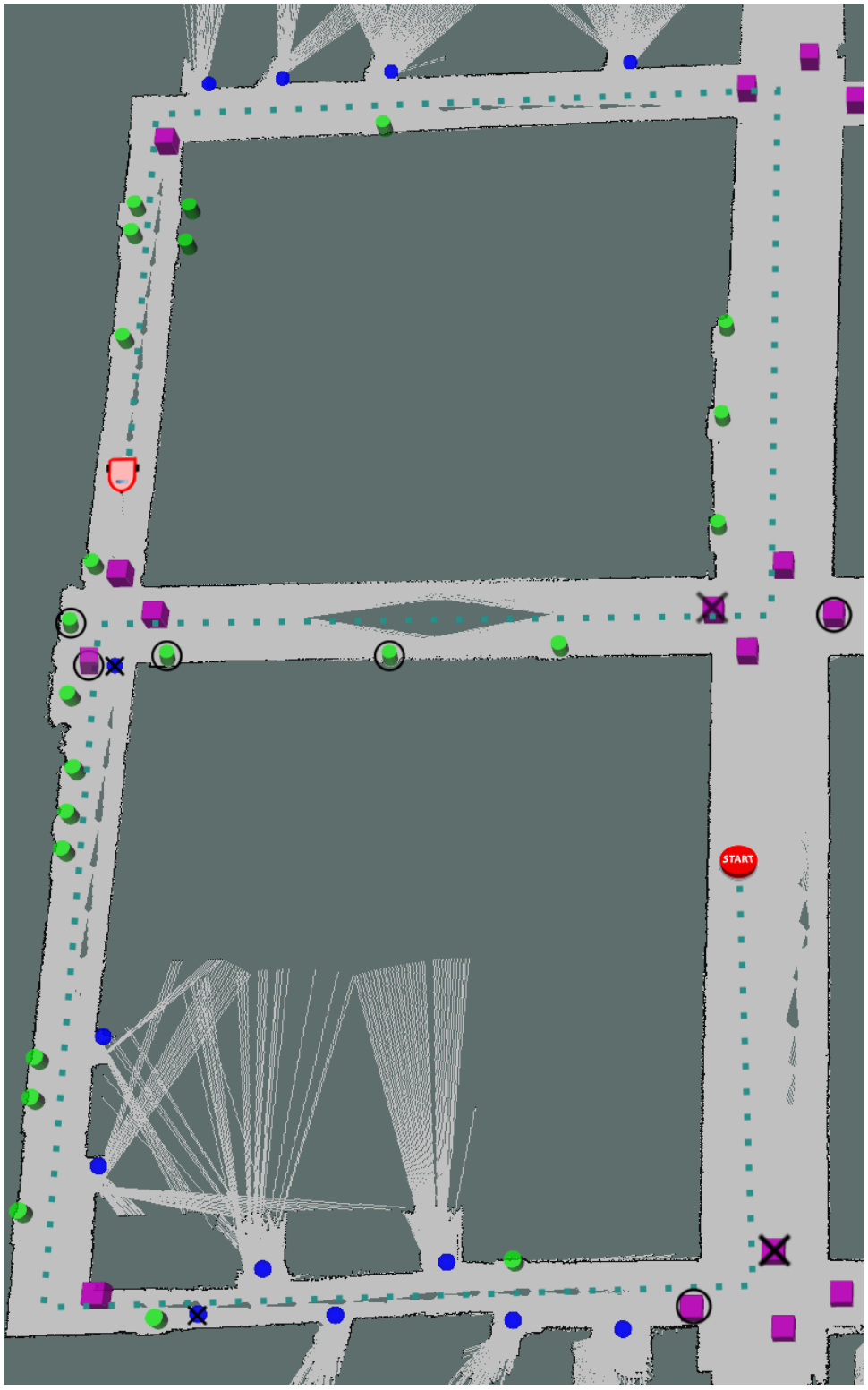


Figure 3.15: Prediction of our system for the ACES3, Austin (test portion) building map. This part of the map is used in the **test** part of the dataset. Closed-rooms predictions are annotated by green cylinders, open-rooms by blue spheres and the start of each corridor in the intersections by purple cubes. The false-positives are annotated by $\times$ and false-negatives by $\circ$ over predictions.

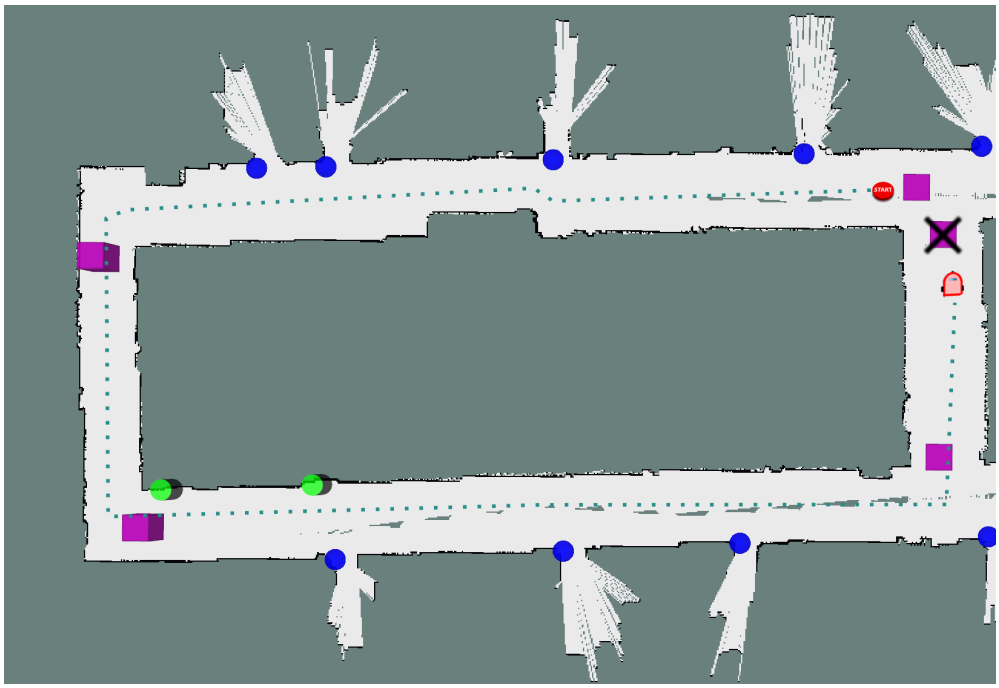


Figure 3.16: Prediction of our system for the real1 (train portion) building map. This part of the map is used in the **train** part of the dataset. Closed-rooms predictions are annotated by green cylinders, open-rooms by blue spheres and the start of each corridor by purple cubes

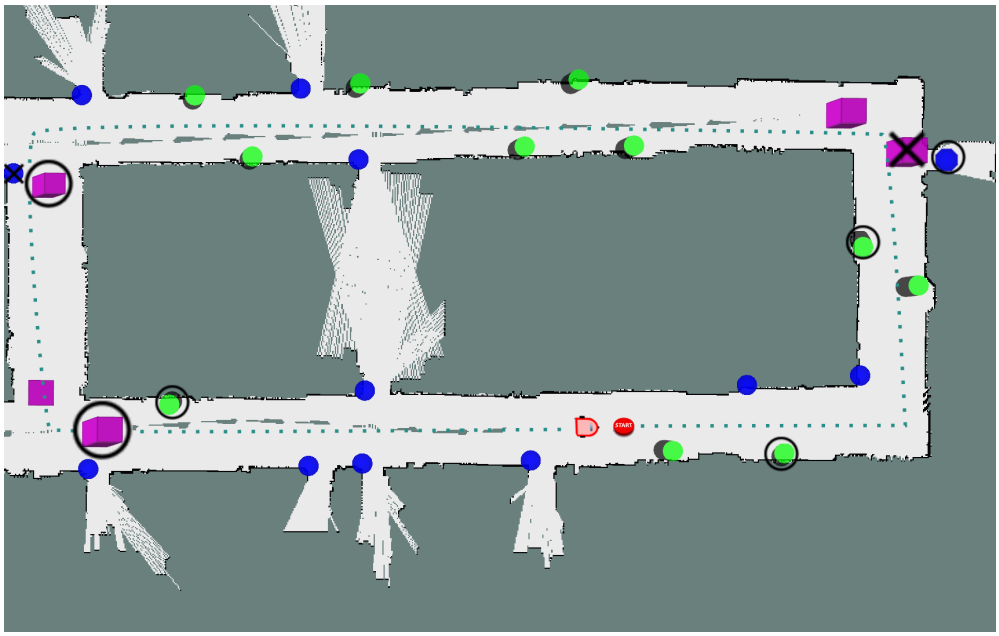


Figure 3.17: Prediction of our system for the real1 (test portion) building map. This part of the map is used in the **test** part of the dataset. Closed-rooms predictions are annotated by green cylinders, open-rooms by blue spheres and the start of each corridor by purple cubes. The false-positives are annotated by $\times$ and false-negatives by $\circ$ over predictions.

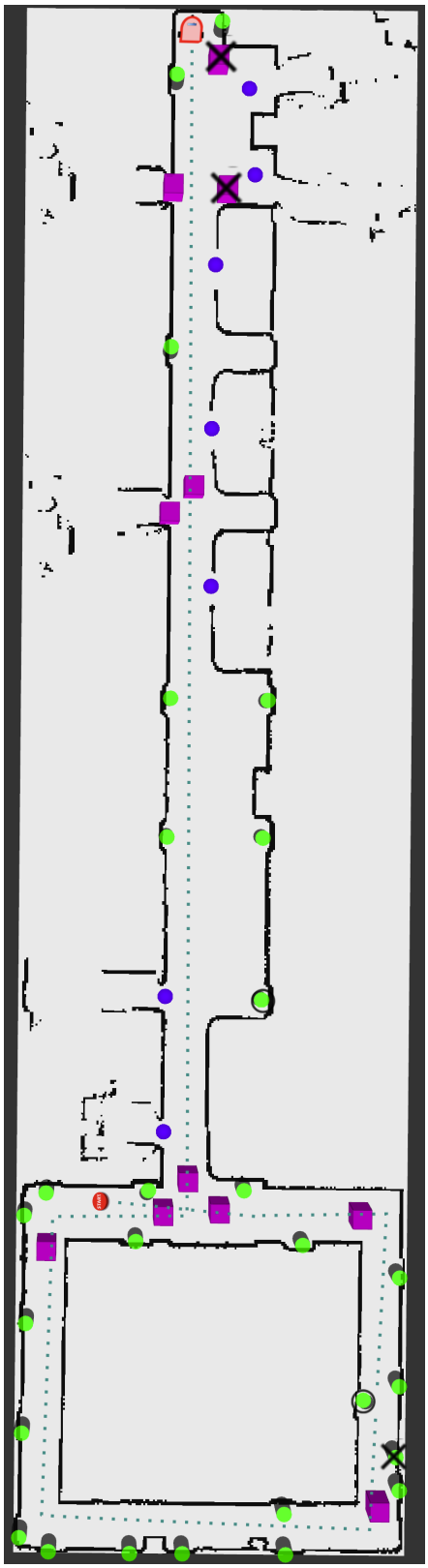
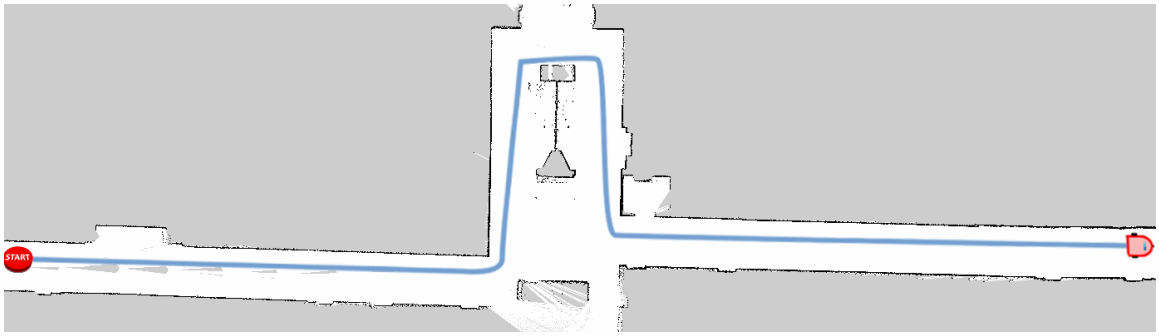
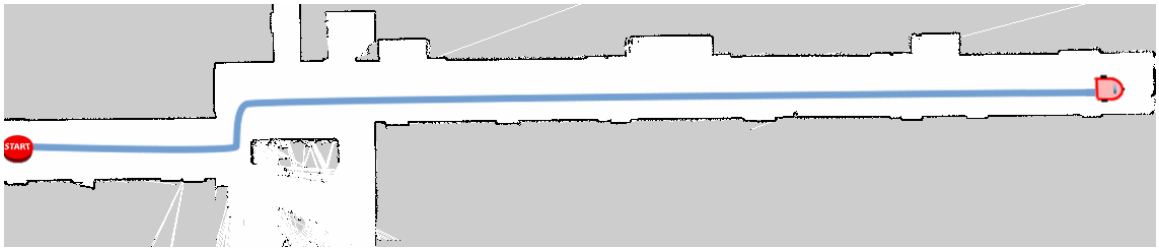


Figure 3.18: Prediction of our system for SeattleUW building map. This map is only used in the **train** part of the dataset. Closed-rooms predictions are annotated by green cylinders an open-rooms by blue spheres

3.3.3 Experiment Three



(a) First part of the experiment



(b) Second part of the experiment

Figure 3.19: Maps for the real world experiment (experiment three). The trajectory of the experiment is annotated with a blue line. These maps are created by running SLAM GMapping in the TASC1 corridors of Simon Fraser University

The third experiment is conducted to evaluate our system in the real-world. In this experiment, we use a ClearPath Husky robot equipped with a Sick LMS-111 LiDAR and an IMU sensor. The experiment is done in two corridors of the TASC1 building at Simon Fraser University. We moved the robot in these two corridors using a joystick while recording the robot's sensors data. The trajectory and map of the environment are shown in Figure 3.19. These maps include a total of 26 closed-rooms, ten corridors (corridors are paths in each intersection) and three open-rooms (Table 3.5). We tested all three models using these two sensors' data.

Table 3.5: Total number of each target class in the real-world experiment.

Closed Room	Open Room	Corridor
26	3	10

Table 3.6: Precision for each of our target classes in the experiment three.

	Closed Room	Open Room	Corridor
Laser	0.96	0.27	1
Map	0.81	0.11	0.57
Combined	0.91	0.33	0.67

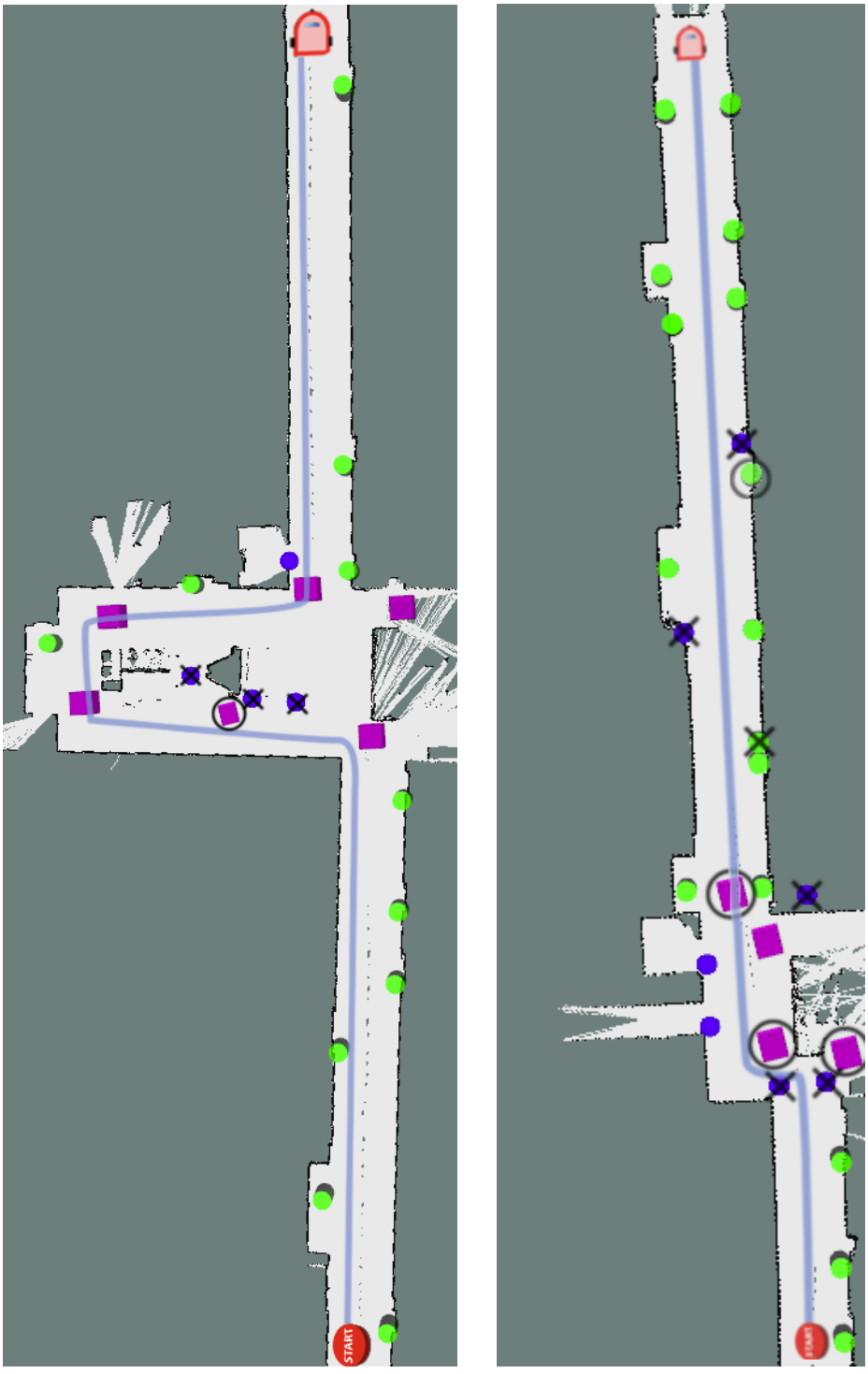
Table 3.7: Recall for each of our target classes in the experiment three.

	Closed Room	Open Room	Corridor
Laser	0.96	1	0.6
Map	0.81	0.67	0.4
Combined	0.77	1	0.2

Table 3.8: F1 score for each one of our target classes on the experiment three.

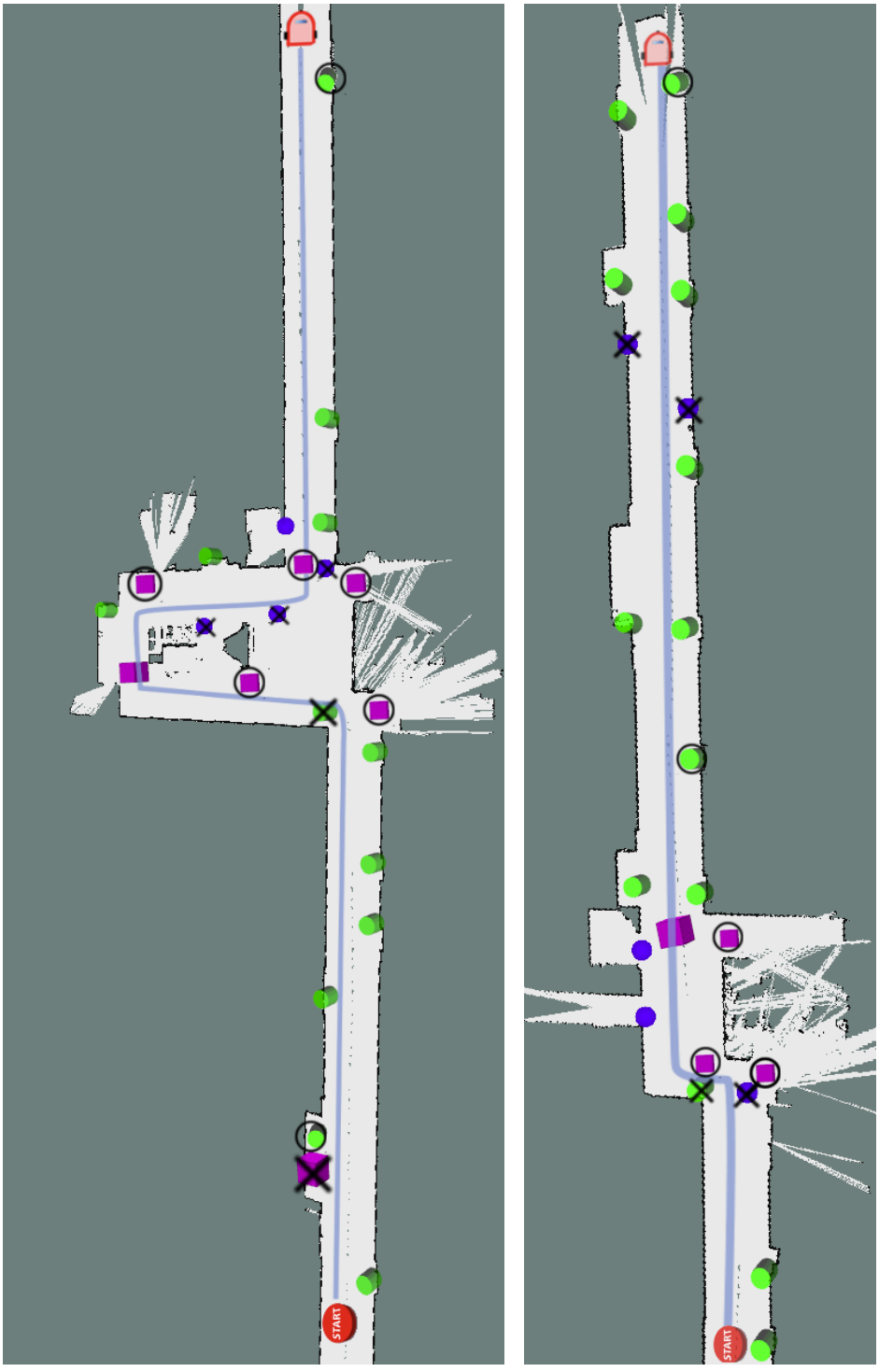
	Closed Room	Open Room	Corridor
Laser	0.96	0.43	0.71
Map	0.81	0.19	0.47
Combined	0.83	0.50	0.31

Tables 3.6, 3.7 and 3.8 show the precision, recall and F1 score of each target class using our three models. Based on these tables, the laser model gives a slightly better F1 score for closed-rooms and corridors compared to the other two models. This is probably due to the increased delays and the accumulating sensor noise while generating the GMap in the real-world settings. We also observe that the models have some difficulty in classifying corridors. In some cases, corridors are classified as open-rooms. Moreover, although recall for open-rooms is high, precision is much lower. It is due to many false-positive predictions from the network. These false-positives increased compared with simulation which may be because of detecting sensor noise as an open-room. The robot trajectories and our system's predictions for all three models are depicted in Figures 3.20 to 3.22.



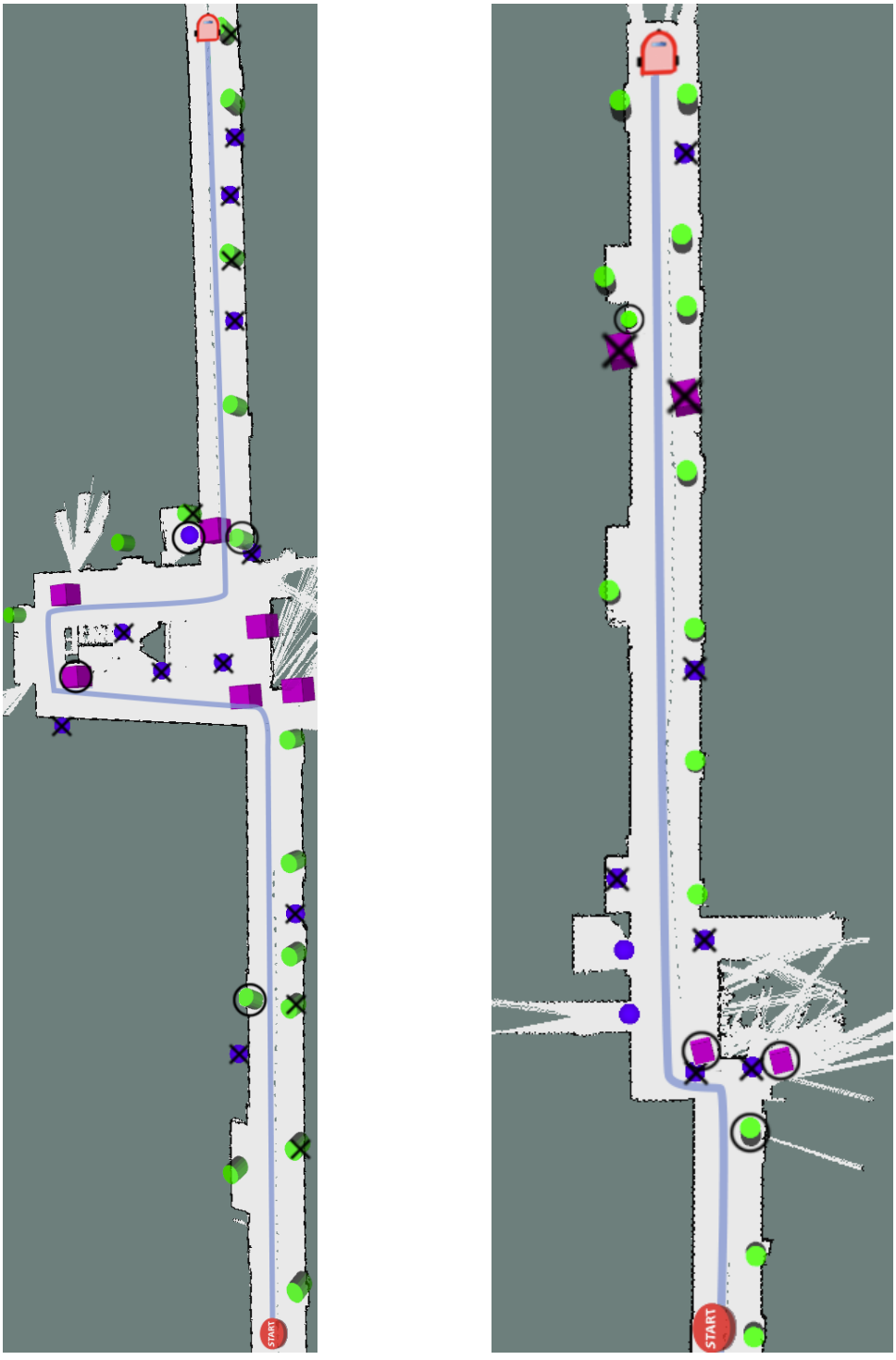
(a) First part of experiment (b) Second part of experiment

Figure 3.20: Prediction of our system for the real-world (TASC-1 building) experiment using the **Laser model**. Closed-rooms predictions are annotated by green cylinders, open-rooms by blue spheres and the start of each corridor by purple cubes. The false-positives are annotated by $\times$ and false-negatives by $\circ$ over predictions.



(a) First part of experiment (b) Second part of experiment

Figure 3.21: Prediction of our system for the real-world (TASC-1 building) experiment using the **Combined model**. Closed-rooms predictions are annotated by green cylinders, open-rooms by blue spheres and the start of each corridor by purple cubes. The false-positives are annotated by $\times$ and false-negatives by $\circ$ over predictions.



(a) First part of experiment (b) Second part of experiment

Figure 3.22: Prediction of our system for the real-world (TASC-1 building) experiment using the **Map model**. Closed-rooms predictions are annotated by green cylinders, open-rooms by blue spheres and the start of each corridor by purple cubes. The false-positives are annotated by $\times$ and false-negatives by $\circ$ over predictions.

3.4 Conclusion and Future Work

In this chapter, we presented a novel system to describe the navigational cues around a robot using a combination of 2D LiDAR data and occupancy grid maps. We trained a CNN to predict the closed-rooms, open-rooms and intersections around the robot. On top of this a tracking module aggregated the predictions to locate and classify the navigational cues more accurately. In addition, results of our tracking module enabled us to describe the navigational cues around the robot using class names and their approximate position relative to the robot. We evaluated our system in different settings in both simulation and the real-world. For all the experiments, we compare the result of our system using three models: (i) Laser model (uses only 2D LiDAR local-map) (ii) Map model (uses only GMap local-map) (iii) Combined model (Uses both of the 2d LiDAR and GMap local-map). Based on our results in the first experiment, the Map model performed with the lowest F1 score while the Laser model and Combined model similarly achieved higher F1 scores. Adding a tracking module in the second experiment, enhanced all the models performance in the explored maps compared to unexplored ones. In the real-world experiment, the laser model achieved the highest performance. Perhaps this is due to increase of delay and sensor noise in the real-world, which generates a less accurate GMap compared to the simulation.

For this work, we created our own dataset using eight occupancy grid maps. In the future, this study can be improved in terms of accuracy and robustness using a dataset that contains a variety of environments. Also, environmental feedback can be extended to include more useful guidance. For instance, the system can be improved to detect steps or stairs and notify the user about them. Moreover, this work can be combined with our follow-ahead system to improve the following behaviour. In particular, the robot can use our detection system to slow down near the intersections and get the user's reactions to choose which path to follow.

Chapter 4

Conclusion

In this thesis, we explored and demonstrated two little-studied capabilities of autonomous robots to facilitate human-robot interactions. First, we defined a follow ahead system whereby the robot follows a person while staying ahead of them. Second, we designed a module to describe the navigational objects around the robot using occupancy grid maps and 2D LiDAR data.

In our first study, explained in Chapter 2, we proposed a “following in front of the leader" robot behavior. Our implementation improves on previous work by featuring a motion model which predicts the trajectory of the user by reasoning about walking direction in the context of the local navigable surroundings. We used state-of-the-art CNN-based object detection, and all our code is freely available online as ROS modules. We proposed a simple error metric for this behavior and evaluated our system in easy and harder settings. The results are qualitatively good, especially for the easy setting. A user study would be required to make a formal claim, but informally we believe our following behavior feels natural and easy in our experiments.

In Chapter 3, we designed a system to describe navigational objects using 2D LiDAR data and occupancy grid maps. Our system consisted of three parts: (i) A neural network that detects objects around the robot using both 2D LiDAR data and local occupancy grid map. (ii) A tracking module which improves the location accuracy of the target objects by aggregating the neural network detections. (iii) A describe module to explain the objects around the robot by naming object class and position relative to the robot. This system was evaluated in different settings using three models in both simulation and the real-world.

Bibliography

[1] J. Andreas, A. D. Dragan, and D. Klein. Translating Neuralese. *Computing Research Repository* , abs/1704.06960, 2017. URL `http://arxiv.org/abs/1704.06960`.

[2] G. Arechavaleta, J.-P. Laumond, H. Hicheur, and A. Berthoz. On the nonholonomic nature of human locomotion. *Autonomous Robots*, 25(1):25–35, Aug 2008. ISSN 1573-7527. URL `https://doi.org/10.1007/s10514-007-9075-2`.

[3] S. Azenkot, C. Feng, and M. Cakmak. Enabling building service robots to guide blind people a participatory design approach. In *Human-Robot Interaction (HRI), 2016 11th ACM/IEEE International Conference on*, pages 3–10. IEEE, 2016.

[4] G. Capi. Assisting and guiding visually impaired in indoor environments. *International Journal of Mechanical Engineering and Mechatronics*, 1929:2724, 2012.

[5] C. A. Cifuentes, A. Frizera, R. Carelli, and T. Bastos. Human–robot interaction based on wearable IMU sensor and laser range finder. *Robotics and Autonomous Systems*, 62 (10):1425–1439, 2014.

[6] A. F. Daniele, M. Bansal, and M. R. Walter. Navigational Instruction Generation As Inverse Reinforcement Learning with Neural Machine Translation. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '17, pages 109–118, New York, NY, USA, 2017. ACM. URL `http://doi.acm.org/10.1145/2909824.3020241`.

[7] Dissanayake, MWM Gamini and Newman, Paul and Clark, Steve and Durrant-Whyte, Hugh F and Csorba, Michael. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on robotics and automation*, 17(3): 229–241, 2001.

[8] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, March 1997. ISSN 1070-9932.

[9] B. P. Gerkey and K. Konolige. Planning and control in unstructured terrain. In *In Workshop on Path Planning on Costmaps, Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2008.

[10] R. Goeddel and E. Olson. Learning semantic place labels from occupancy grids using cnns. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3999–4004. IEEE, 2016.

[11] A. R. Golding and N. Lesh. Indoor navigation using a diverse set of cheap, wearable sensors. In *Wearable Computers, 1999. Digest of Papers. The Third International Symposium on*, pages 29–36. IEEE, 1999.

[12] G. Grisetti, C. Stachniss, and W. Burgard. Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 23(1):34–46, Feb 2007. ISSN 1552-3098.

[13] H. Guan, Y. Yu, Z. Ji, J. Li, and Q. Zhang. Deep learning-based tree classification using mobile LiDAR data. *Remote Sensing Letters*, 6(11):864–873, 2015.

[14] H. Guan, J. Li, S. Cao, and Y. Yu. Use of mobile LiDAR in road information inventory: A review. *International Journal of Image and Data Fusion*, 7(3):219–242, 2016. ISSN 19479824. URL `http://dx.doi.org/10.1080/19479832.2016.1188860`.

[15] J. . Gutmann and D. Fox. An experimental comparison of localization methods continued. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 454–459 vol.1, Sept 2002.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] D. M. Ho, J. S. Hu, and J. J. Wang. Behavior control of the mobile robot for accompanying in front of a human. In *Advanced Intelligent Mechatronics (AIM), 2012 IEEE/ASME Int. Conf.*, pages 377–382. IEEE, July 2012.

[18] A. Howard and N. Roy. The Robotics Data Set Repository (Radish), 2003. URL `http://radish.sourceforge.net/`.

[19] E. J. Jung, B. J. Yi, and S. Yuta. Control algorithms for a mobile robot tracking a human in front. In *Intelligent Robots and Systems, 2012 IEEE/RSJ Int. Conf.*, pages 2411–2416. IEEE, Oct 2012.

[20] K. Koide and J. Miura. Identification of a specific person using color, height, and gait features for a person following robot. *Robotics and Autonomous Systems*, 84:76 – 87, 2016. ISSN 0921-8890. URL `http://www.sciencedirect.com/science/article/pii/S0921889015303225`.

[21] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard. Feature-Based Prediction of Trajectories for Socially Compliant Navigation. In *Proceedings of Robotics: Science and Systems (RSS)*, Sydney, Australia, July 2012.

[22] A. Leigh, J. Pineau, N. Olmedo, and H. Zhang. Person tracking and following with 2D laser scanners. In *Robotics and Automation (ICRA), 2015 IEEE Int. Conf.*, pages 726–733. IEEE, 2015.

[23] B. Li, J. P. Munoz, X. Rong, Q. Chen, J. Xiao, Y. Tian, A. Arditi, and M. Yousuf. Vision-based Mobile Indoor Assistive Navigation Aid for Blind People. *IEEE Transactions on Mobile Computing*, 2018.

[24] D. V. Lu and W. D. Smart. Towards more efficient navigation for robots and humans. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ Int. Conf. On*, pages 1707–1713. IEEE, 2013.

[25] K. C. A. S. J. S. Lynne Grewe, Archana Kashyap. iSight: computer vision based system to assist low vision. *Proc.SPIE*, 10646:10646 – 10646 – 8, 2018. URL `https://doi.org/10.1117/12.2305233`.

[26] G. P. Moustris and C. S. Tzafestas. Assistive front-following control of an intelligent robotic rollator based on a modified dynamic window planner. In *Biomedical Robotics and Biomechatronics (BioRob), 2016 6th IEEE Int. Conf.*, pages 588–593. IEEE, June 2016.

[27] G. P. Moustris and C. S. Tzafestas. Intention-based front-following control for an intelligent robotic rollator in indoor environments. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7, Dec 2016.

[28] O. M. Mozos, C. Stachniss, and W. Burgard. Supervised learning of places from range data using adaboost. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1730–1735. IEEE, 2005.

[29] M. Munaro and E. Menegatti. Fast RGB-D people tracking for service robots. *Autonomous Robots*, 37(3):227–242, 2014.

[30] P. Nikdel, R. Shrestha, and R. Vaughan. The Hands-Free Push-Cart: Autonomous Following in Front by Predicting User Trajectory Around Obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[31] H. Ogawa, K. Tobita, K. Sagayama, and M. Tomizuka. In *2014 IEEE Symposium on Computational Intelligence in Robotic Rehabilitation and Assistive Technologies (CIR2AT), title=A guidance robot for the visually impaired: System description and velocity reference generation*, pages 9–15, Dec 2014.

[32] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

[33] A. Pronobis and R. P. Rao. Learning deep generative spatial models for mobile robots. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 755–762. IEEE, 2017.

[34] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009.

[35] J. Redmon and A. Farhadi. YOLO9000: Better, Faster, Stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, July 2017.

[36] M. Skubic, D. Perzanowski, S. Blisard, A. Schultz, W. Adams, M. Bugajska, and D. Brock. Spatial language for human-robot dialogs. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 34(2):154–167, 2004.

[37] Y. Sun, L. Sun, and J. Liu. Human comfort following behavior for service robots. In *Robotics and Biomimetics (ROBIO), 2016 IEEE Int. Conf.*, pages 649–654. IEEE, 2016.

[38] W. Takano, H. Imagawa, and Y. Nakamura. Prediction of human behaviors in the future through symbolic inference. In *Robotics and Automation (ICRA), 2011 IEEE Int. Conf.*, pages 1970–1975. IEEE, 2011.

[39] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy. Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation. In *AAAI*, 2011.

[40] J. Tominaga, K. Kawauchi, and J. Rekimoto. Around me: a system with an escort robot providing a sports player's self-images. In *Proceedings of the 5th Augmented Human Int. Conf.*, page 43. ACM, 2014.

[41] R. Vaughan. Massively Multi-Robot Simulations in Stage. *Swarm Intelligence*, 2(2-4): 189–208, December 2008.

[42] E. A. Wan and R. V. D. Merwe. The unscented Kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pages 153–158, 2000.

[43] N. Wojke, R. Memmesheimer, and D. Paulus. Joint operator detection and tracking for person following from mobile platforms. In *Information Fusion (Fusion), 2017 20th Int. Conf.*, pages 1–8. IEEE, 2017.

[44] D. Wooden, M. Malchano, K. Blankespoor, A. Howardy, A. A. Rizzi, and M. Raibert. Autonomous navigation for BigDog. In *Robotics and Automation, 2010 IEEE Int. Conf.*, pages 4736–4741. IEEE, May 2010.

[45] N. Yao, E. Anaya, Q. Tao, S. Cho, H. Zheng, and F. Zhang. Monocular vision-based human following on miniature robotic blimp. In *Robotics and Automation (ICRA), 2017 IEEE Int. Conf.*, pages 3244–3249. IEEE, 2017.

[46] D. Yi, T. M. Howard, M. A. Goodrich, and K. D. Seppi. Expressing homotopic requirements for mobile robot navigation through natural language instructions. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 1462–1468. IEEE, 2016.

[47] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[48] L. Zhang and R. Vaughan. Optimal robot selection by gaze direction in multi-human multi-robot interaction. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ Int. Conf.*, pages 5077–5083. IEEE, 2016.

[49] Y. Zheng, Y. Liu, and J. H. Hansen. Navigation-orientated natural spoken language understanding for intelligent vehicle dialogue. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 559–564. IEEE, 2017.

[50] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa. Planning-based prediction for pedestrians. In *Intelligent Robots and Systems (IROS), Int. Conf.*, pages 3931–3936. IEEE, 2009.