EU Horizon 2020 Research & Innovation Program
Advanced Robot Capabilities & Take-Up
ICT-25-2016-2017
Grant Agreement No. 779942

CROWDBOT

Safe Robot Navigation in Dense Crowds

http://crowdbot.eu/project/

Technical Report

D 3.1: 1st Release of Localization, Mapping & Local Motion Planning

Work Package 3 (WP 3)
Navigation

Task Lead: Swiss Federal Institute of Technology (ETHZ), Switzerland
WP Lead: Swiss Federal Institute of Technology (ETHZ), Switzerland

Table of Contents

EXECUTIVE SUMMARY 4

1. INTRODUCTION 5

2. RELATED WORK 6

2.1 Active SLAM in static environments 6

2.2 SLAM in dynamic environments 6

3. MAPPING 8

3.1 SLAM framework 8

3.1.1. Front-end 8

3.1.2. Back-end 8

3.1.3. Occupancy grid mapping 9

3.2 Autonomous exploration for initial map building 11

3.2.1. Frontier exploration 11

3.2.2. Navigation 11

3.2.3. Path utility 11

3.3 Detection and tracking of moving people in 2D laser scans 15

3.3.1. Classifying dynamic scan points 15

3.3.2. Adaptive person ellipse 17

3.4 Mapping results in simulation 17

3.4.1. Simulation environment 18

3.4.2. People/dynamic object simulator 18

3.4.3. Metrics for map quality 19

3.4.4. Mapping performance 19

3.5 Mapping demonstration on the Pepper robot 23

3.6 Outlook for CROWDBOT mapping 25

4. LOCALISATION 27

4.1 Branch and bound map matching 27

4.2 Localisation performance 29

4.2.1. Results in simulation environment 29

4.2.2. Results in real world office environments 33

4.3 Outlook for CROWDBOT localisation 37

5. LOCAL MOTION PLANNING 38

5.1 Global motion planning 38

5.2 Local, low-latency planning 40

5.3 Outlook for CROWDBOT local motion planning 42

5.4 Dynamical systems-based control for avoiding concave obstacles 42

5.4.1. Method 43

5.4.2. Experimental results *44*

REFERENCES **45**

Executive Summary

This report details the mapping, localisation and path planning solutions developed for the CROWDBOT project between months M1 to M20. Each of the three technical components have been designed with the explicit goal of achieving robot navigation in *crowded* environments, where many existing methods struggle due to the high degree of dynamic motion around the robot. In particular, the three key challenges that we address are:

- Generating clean and coherent maps of the static environment despite the presence of dynamic obstacles during mapping;
- Achieving fast, and accurate localisation when prior information on the robot pose is unavailable;
- Executing low-latency local motion planning that balances the collision avoidance routines of the robot with making progress along the path.

Mapping via active SLAM

Our proposed mapping strategy employs an active SLAM framework. Whereas most mapping frameworks conduct pure exploration to maximise coverage alone, active SLAM also considers the robot's ability to maintain accurate localisation within the available map during the mapping process. Active SLAM accounts for the exploration-exploitation trade-off when deciding where the robot should sense next to either observe new parts of the environment (explore) or reduce its pose uncertainty by re-observing known locations in the map (exploit). We capture this trade-off in a Rényi entropy-based utility function, which combines the map and pose uncertainties, two fundamentally different quantities, in a mathematically sound framework without the need for hand-tuned scaling parameters.

To deal with dynamic obstacles in the sensor field of view, i.e. a 360º 2D LIDAR scan in our case, we conduct a pre-filtering step over the incoming scans to remove points that are deemed to be returns from dynamic obstacles rather than from the static environment. An additional benefit of the pre-filtering is that we can perform basic LIDAR person tracking by clustering the dynamic LIDAR points and running a Kalman Filter over each cluster to estimate the target's velocity. Further details, including experimental results on the CROWDBOT Pepper robot mapping in a crowd of humans, are provided in Section 3. Our code is available open source at: https://github.com/ethz-asl/crowdbot_active_slam.

Fast, prior-free accurate localisation

Existing localisation solutions, such as adaptive Monte Carlo localisation, rely heavily on having a prior estimate of the robot's pose to initialise the search. These methods are shown to perform very poorly when prior pose information is unavailable or under equivalent circumstances such as when the robot is "kidnapped". Unfortunately, when operating in a dense crowd, the reality is that the robot may often be unable to sense the static environment around it due to severe occlusions from the crowd. Thus, raising the chances for experiencing kidnapped robot scenarios.

Our goal for CROWDBOT localisation is to achieve a prior-free solution that is fast to generate an accurate estimate of the robot pose at initialisation and is also robust to partial or temporary sensor occlusions. Thus, our proposed approach is based on map-matching, as opposed to scan-matching. Furthermore, we use a branch-and-bound search to improve the computational complexity of the matching routine and enable real-time operation. Further details and comparisons are provided in Section 4. Our code is available open source at: https://github.com/danieldugas/map_matcher.

Low-latency local motion planning

One of the main aims of CROWDBOT is to overcome the "freezing robot" problem, i.e. when there are many dynamic obstacles around the robot and it is unable to find a collision-free path to its goal. Our approach is to first plan a global path through the map of the static environment. This allows the robot to ignore the presence of moving objects in its initial planning solution but shifts the responsibility of collision avoidance onto the local motion planner. Thus, our local motion planning routine is designed to be highly responsive, while at the same time balancing the safety considerations with managing progress along its planned path. This results in a local motion planner that yields and gains space in an intuitive and highly reactive manner. Analysis of our current implementation as well as directions for future development are discussed in Section 5.

1. Introduction

One of the aims of CROWDBOT is to develop robust localisation and mapping techniques that are able to cope with highly dynamic environments. Compared to static environments, where all detected landmarks can be taken into account for localisation purposes, the use case of densely occupied dynamic environments entails additional criteria for the localisation approach. The localisation needs to be robust and precise even though the majority of the robot's field of view may be occluded by other traffic participants, i.e. other pedestrians.

In this report, we detail the first release of the CROWDBOT mapping, localisation and local motion planning packages. Each of the three components are designed to allow a CROWDBOT robot to navigate safely within the broad spectrum of diverse situations that occur in a pedestrian environment. In this first release, we focus on 2D LIDAR-based solutions. This decision comes from the recommendations of CROWDBOT D6.2 "Robot Design Recommendations", which highlighted the need for 360° horizontal planar sensing for 2D navigation.

The algorithms described in this report have been tested on the CROWDBOT Pepper robot, shown in Figure 1.1, as well as in simulation on other robot form factors with similar sensing capabilities.

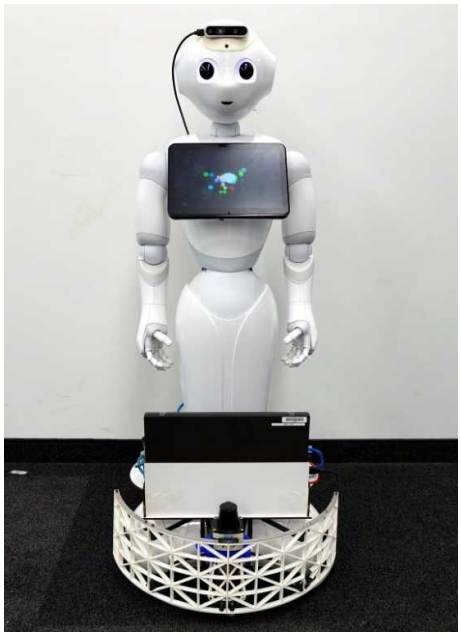


Figure 1.1: The CROWDBOT Pepper robot has two 2D SICK LIDARS mounted on the front and back of the base of the robot to provide high resolution 360° planar sensing of the environment. These are the primary sensors used in the first release of our mapping, localisation and local motion planning packages.

In the following section we describe the state of the art in SLAM, from which we develop our crowd-specific mapping and localisation solutions. Section 3 details our active SLAM algorithm for mapping in crowded environments. Our main contribution here is a framework that can generate clean and coherent maps despite the presence of many dynamic obstacles in the environment. In Section 4 we describe our localisation method. Our approach relies on a branch-and-bound map-matching routine, which provides fast and robust localisation solutions without the need for a prior estimate on the robot pose. Finally, Section 5 provides details on our global and local motion planning strategies. Our general paradigm is to construct the global path in the map of the static environment and have the local motion planning handle the lower-level interactions with other crowd participants. Each section is concluded with an outlook towards future iterations of the CROWDBOT mapping, localisation and local motion planning packages.

2. Related work

Most of the research in the area of SLAM has be conducted under the assumption of a static environment where the robot is steered by human actions. Because of this it is hard to find a work that tries to solve the active SLAM problem for dynamic environments. There are a few active SLAM algorithms for static environments, which are summarised in Section 2.1. There are also some approaches on solving the SLAM problem for dynamic environments, which are discussed in Section 2.2.

One approach to solve the active SLAM problem for dynamic environments is described in [2]. The authors developed their own active SLAM algorithm for dealing with dynamic problems. The main idea was to use a monocular camera pointing towards the ceiling. The camera is used to detect ceiling features and to estimate the pose of the robot using an iterative closest point (ICP) algorithm with the detected features. The usage of a camera, which is pointing to the ceiling, prevents moving objects in the lateral plane, such as humans, from disturbing the robot's sensors. But in our opinion this approach has some general disadvantages, like the need for additional sensors for navigation, a higher computation usage and the need for structures on the ceiling. In this thesis we show that active SLAM is possible in crowded environments by only using 2D laser scans.

2.1 Active SLAM in static environments

As the name suggests, active SLAM seeks to find the control actions that lead to sensor observations which simultaneously minimise both the map and robot localisation uncertainty. Active SLAM has largely been explored for static environments. In most approaches the active part can be used for different SLAM frameworks. In general, the goal is to decrease the pose uncertainty of the robot and the map uncertainties simultaneously and in an optimal way. A good survey on the state-of-the-art in active SLAM and SLAM in general can be found in [3]. The authors describe active SLAM as a decision-making problem which encompasses an exploration-exploitation trade-off. To solve this problem there are several frameworks that can be used. One of these is the Theory of Optimal Experimental Design (TOED) [4]. An example applied to active SLAM can be found in [5]. It allows selecting future robot actions based on the predicted pose and map uncertainties.

Another framework is the application of information theory [6] to the decision problem. The decision making here is in general guided by the notion of information gain. An example application is [1] where a utility function based on Rényi's general theory of entropy [7] is used. One main advantage of this utility function is that it automatically trades off between exploration and exploitation. Because of this advantage, we have also incorporated this utility function into our active SLAM solution. Another example is the work of Stachniss et al. [8] where a Rao-Blackwellized particle filter has been used for solving the SLAM problem using occupancy grid maps as map representation. Map and pose uncertainties are described using the Shannon entropy. Carrillo et al. [1] showed in their work that utility functions that only use Shannon's entropy, neglect the fact that the entropies for map and pose uncertainties are two completely different quantities, and thus, finding a comparable scale between the two can be problematic. Nevertheless, utility functions that directly combine pose and map uncertainty have been used in several other active SLAM frameworks, for example, when combined with Model Predictive Control [9].

One of the first approaches of active SLAM to Pose SLAM is described in [10]. The advantage in the computation of the utility is that with Pose SLAM only one map exists instead of multiple maps, as generated for example using a particle filter SLAM system. An extension of this work is described in [11] where a growing RRT* tree provides a set of candidate paths, which are then used for the computation of the utility functions. A further work, which only used the Shannon entropy in the utility computation, is [12]. This approach can be used for any graph-based SLAM algorithm. The work proposes the use of a Topological Feature Graph (TFG) as map representation, where landmarks are connected together if they belong to the same object.

2.2 SLAM in dynamic environments

Until now, active SLAM for crowded environments has not been a focus in research. Therefore, in this section, we will focus on applications for SLAM in dynamic environments. Usually, the developed approaches can be

easily included or adapted for active SLAM. Dynamic objects are mostly handled in two ways. Either one tries to detect them and treats them as outliers or tracks each dynamic object separately, using for example multi-target tracking. An overview on multi-target tracking can be found in the work of Luo et al. [13]. One of the first approaches combining SLAM with detection and tracking of moving objects (DATMO) in dynamic outdoor environments is described in [14]. The detection of moving vehicles is based on the inconsistencies between the observed free space and occupied space in the local grid map, which has been built so far. Detected moving objects are then tracked by a multiple hypothesis tracker (MHT) coupled with an adaptive interacting multiple model filter. Bahraini et al. [15] propose to use a multilevel-RANSAC algorithm for classifying detected objects into stationary and moving objects. The detected objects are extracted from 2D laser scans using an adaptive breakpoint detector [16]. The objects classified as moving are then integrated into an EKF SLAM algorithm. Another approach called SLAMIDE [17] includes the detected dynamic objects into the least-square formulation of SLAM. To include dynamic objects into SLAM, they combine sliding window optimisation and least-squares together with generalised expectation maximisation to do reversible model selection and data association.

Explicitly accounting for dynamic objects in SLAM helps with localisation in highly dynamic environments. Another approach, called dynamic pose graph SLAM [18], focuses more on the long-term mapping for low dynamic environments and uses iSAM [19] as the underlying SLAM state estimation engine. Laser scans for the same area of an environment at different times are compared to perform change detection. If a change has occurred, old poses and scans that no longer match the current state are removed from the pose graph.

Mertz et al. [20] describe a DATMO system for 2D and 3D laser scanners where the core algorithm of DATMO is described in [21] and [22]. Objects in the scan data are clustered into segments using a basic region growing algorithm. They then extract features (lines and corners) from the segments, such that the tracking using Kalman Filters is simplified. Yin et al. [23] include the detection of dynamic objects into the scan matching process for graph-based SLAM. The scan matching consists of three phases. The first phase executes a Hough Transform-based segmentation to extract and group line features, the second phase is an occupancy-analysis-based moving object detection, which detects and discards moving objects. The third phase performs a linear regression-based feature matching to estimate the roto-translation parameters. This approach is based on line features in indoor environments and thus would not suitable for moving people.

A more exotic approach to detecting moving objects is based on deep learning methods [24]. In fact, this approach is not restricted to moving objects as the geometric structure of the laser scan points is learned and can be used to detect objects independent of the moving state. A Convolutional Neural Network (CNN) is trained to detect wheelchairs and walkers. An extension of this work includes the detection of people in the 2D range data [25]. Such methods have the potential to be combined with existing active SLAM approaches to improve map and localisation quality.

3. Mapping

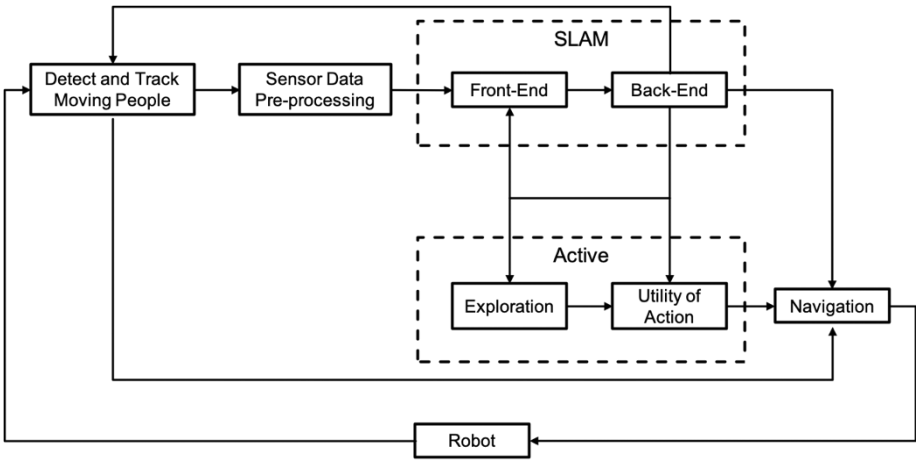


Figure 3.1: Overview of the active SLAM framework developed for CROWDBOT. The boxes show the main modules necessary for active SLAM in crowded environments. Arrows show the flow of data between the modules.

3.1 SLAM framework

The focus on the development of our SLAM framework was to build a state-of- the-art system for active SLAM in crowded environments. For programming reasons the usage of a complete SLAM system, such as gmapping[1] [26][27] or ORB-SLAM [28][29], was not desired, as active SLAM needs access to internal information, which is not readily exposed by the out-of-the-box frameworks. In addition, developing a custom framework gives extra flexibility in customising its characteristics to the problem at hand. Further, in prospect of the development of a solution for crowded environments, the system should be kept simple, such as only relying on 2D laser scans.

3.1.1. Front-end

For the SLAM front-end, we use a laser scan matcher for the generation of laser odometry information. The scan matcher is an iterative closest point (ICP) variant using a point-to-line metric [30]. The implementation in this thesis is based on the implementation in the ROS package laser_scan_matcher[2], which uses this ICP variant for consecutive laser scans. A keyframe-based approach is used to prevent pose drift. The ICP variant, also called canonical scan matcher (CSM) is a very fast scan matcher. Much of the speed in the CSM comes from a smart correspondence-search procedure. As an alternative, the libpointmatcher developed from Pomerleau et al. [31] has also been tested as a potential front-end, but results showed it to be slower. Further, an extensive parameter tuning of scan filters is necessary to achieve good performance. For these reasons, the CSM has been chosen as it is already highly optimised for our application. The CSM has also an accurate closed-form solution implemented for the estimation of ICP's covariance [32].

3.1.2. Back-end

For the SLAM back-end, an incremental graph-based SLAM algorithm called iSAM2 [33] is used. The graph is based on a factor graph implementation. See Figure 3.2 for an example of a factor graph for pose SLAM. The graph is built using factors, which describe the relationship between the nodes. The nodes correspond to the position estimates in the world. Factors can be built from consecutive poses through laser odometry and through loop closures. iSAM2 is based on further improvements of iSAM [19] and Square Root SAM [34]. A nice tutorial on factor graphs and GTSAM[3] can be found in [35]. GTSAM is a smoothing and mapping (SAM)

[1] http://wiki.ros.org/gmapping, (Accessed on 20.12.2018)

[2] http://wiki.ros.org/laser_scan_matcher, (Accessed on 20.12.2018)

[3] https://borg.cc.gatech.edu, (Accessed on 20.12.2018)

library from Georgia Tech, which has implemented all above-mentioned algorithms. iSAM2 has been chosen, as it is, for the nature of iterative algorithms, very fast and currently one of the state-of-the-art SLAM back-ends. iSAM2 allows us to easily access the marginal covariances of the current pose graph, which will be necessary for the computation of the utility function of our active SLAM framework, as presented in Section 3.2.3.

The pose graph is built online while the robot is moving. In the beginning, we initialise the graph with a prior factor on the starting pose. In simulation we use simulated localisation information to initialise the SLAM map frame to the same frame as the ground truth map frame, allowing us to easily perform comparisons. In a real environment, where an initial localisation is unavailable, the map frame is arbitrarily set to having its origin at the robot's initial position. After the robot moves for a certain distance, which can be estimated using the laser odometry data, a new node is added to the graph. A new factor is defined between the previous and current node pose. These factors also contain pose uncertainty information, which can be retrieved by our laser scan matcher as mentioned in Section 3.1.1. When adding new nodes, we save the corresponding laser scans for map building and loop closure purposes. After every newly added node, the system searches for loop closures. Loop closures are found by iterating through the pose graph and by checking if there exist poses in a range around the current pose that do not exceed a certain threshold. For each loop closure that is found, we use the CSM to find the relative pose between the two laser scans of the loop closure poses. For initialisation, the relative pose between them in the current graph is used. The resulting relative pose is used to build a new factor for the factor graph. After the graph has been searched for loop closures, the pose graph is optimised. For computational reasons we check after each scan matching step if enough time is left for further loop closure computations. After a scan match step, we skip a certain number of nodes in the graph, with the idea that we only compute one loop closure per graph intersection area.

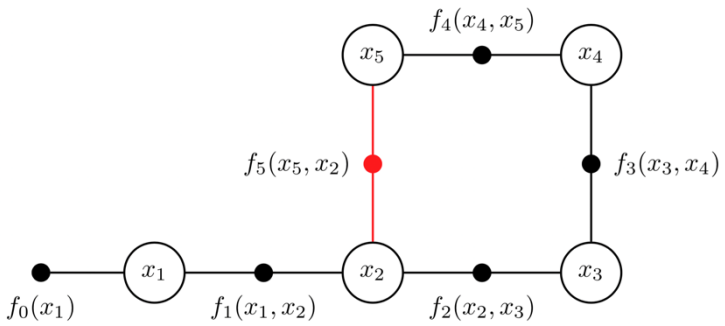


Figure 3.2: An example of a factor graph for pose SLAM, where x_i corresponds to the nodes of the factor graph and f_i to the factors. Here, $f_0(x_1)$ is a prior factor on x_1. $f_5(x_5, x_2)$ is a factor between x_5 and x_2, which describes a loop closure. Figure referenced from [35].

3.1.3. Occupancy grid mapping

The map is represented by a 2D occupancy grid map and is computed using the occupancy grid mapping algorithm [36]. The general update equation for an observed map cell can be defined using the log odds notation as:

$$l(m_i|z_{1:t}, x_{1:t}) = l(m_i|z_t, x_t) + l(m_i|z_{1:t-1}, x_{1:t-1}) + l(m_i), \tag{3.1}$$

where the three terms on the right hand side of the equation represent the inverse sensor model, recursive term and prior, respectively. m_i is the i-th discrete random variable of the occupancy grid map, which describes the occupation of a cell. z and x are time-dependent measurement and state variables, respectively. The prior is the log odds value of the prior probability $P(m_i)$ at map initialisation. In our case we initialise all m_i as $P(m_i) = 0.5$ (i.e. unknown), which results in a value of zero for the prior. The recursive term is simply the log odds value in the previous time step. The inverse sensor model term handles the update of the log odds value depending on the latest measurement z_t. In Table 3.1 the chosen probabilities for our sensor model are shown. The value of $P(z = 1|m = 1)$ has been chosen such that there exists the possibility

that a cell could be occupied by a window or a glass door, which is difficult to detect using only 2D LIDARs. Laser beams can reflect off glass or pass through completely and induce wrong measurements.

Table 3.1: Model used to describe our sensor setup, i.e. the probabilities $P(z|m)$.

	$z = 1$	$z = 0$
$m = 1$	0.8	0.2
$m = 0$	0.01	0.99

The corresponding inverse sensor model is shown in Table 3.2.

Table 3.2: Inverse sensor model, i.e. the probabilities $P(m|z)$.

	$z = 1$	$z = 0$
$m = 1$	0.998	0.168
$m = 0$	0.002	0.832

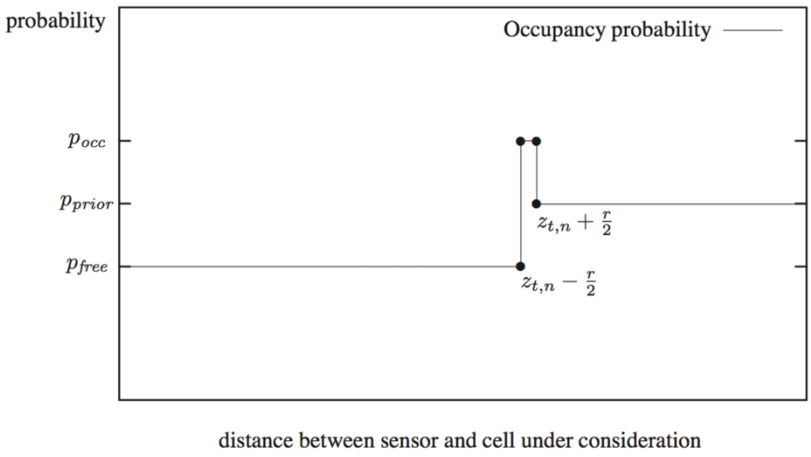


Figure 3.3: The inverse sensor model for a laser beam hitting a wall at distance $z_{t,n}$, where r can be interpreted as the cell size of the occupancy grid map.[4]

The inverse sensor model is shown in Figure 3.3. Laser measurements are not noise free and this noise can have an impact on which cell the measurement is deemed to lie in. Thus, one can define a region with the size r, in our case r corresponds to the map resolution, where the probability is updated as an occupied cell. In other words, this means that if a sensor measurement is not exactly in the centre of a map cell, the next closest cell is also updated as an occupied cell. This results in occupancy grid maps that tend to have thicker walls.

As the pose graph estimates can change after each optimisation, i.e. after adding a new node, the whole map needs to be recomputed. This is intractable for growing pose graphs. For this reason, the map is only updated incrementally after a new node is added. As this SLAM system is intended to be used in an active SLAM approach, recomputing the whole map can be done after a goal position of the robot has been reached and before the possible future goals and the corresponding utilities of the action paths are computed. If it is necessary to only use the SLAM system, the re-computation can be done after a fixed number of nodes are added to the graph.

[4] Figure source: http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam11-gridmaps.pdf, (Accessed on 05.01.2019)

3.2 Autonomous exploration for initial map building

To autonomously build a consistent map of its environment, the robot must explore new areas of the space while maintaining good localisation with respect to the map that it is updating online. This leads to an exploration-exploitation trade-off, where good localisation solutions can be found when revisiting previously observed regions, yet conversely, this does not allow the robot to explore new regions to complete its mapping task. In our active SLAM framework, we incorporate this trade-off in the design of the utility function, which is used to evaluate potential frontier waypoints in the current map. The utility function accounts for the path the robot will take to reach the frontier waypoint, specifically the sensor observations that the robot would receive along the path as well as any potential loop closures that could be achieved.

3.2.1. Frontier exploration

Frontier detection [37] can be used to generate possible future goals for a mobile robot attempting to map a previously unknown environment. Frontiers are defined as regions on the border of explored and unexplored areas. In the case of occupancy grid maps, these frontiers are the regions between free and unknown cells. Our implementation is based on the ROS frontier exploration package[5]. The frontier cells are first detected by searching for free cells that are only adjacent to unknown cells. These are then clustered together. A cluster is only classified as frontier if its size is greater than a certain threshold. This threshold is a design parameter and is also dependent on the map resolution and the environment in which the robot operates. In our case the threshold is set to 8 cells, which corresponds to a range of up to 0.4m when using a map resolution of 0.05 m. This represents a length scale that is comparable to the robot's footprint.

In principle, frontier exploration does not take into consideration the possibility of revisiting already observed regions to decrease robot pose uncertainties. During exploration, the paths from the current robot pose to the frontiers can pass through previously observed regions but are not specifically required to. As described in Section 3.2.3, we explicitly consider possible future loop closures in the computation of the utility of a path.

3.2.2. Navigation

Since global motion planning is not a main focus of CROWDBOT, we base our global path planning routines on the existing ROS *move_base*[6] package. The *move_base* package also includes a built in local motion planner, which we used in initial studies of our mapping and localisation algorithms. However, this will be replaced with our custom local motion planner described in Section 5.

For the computation of the utility function described in the next section, the paths from the current robot pose to the frontiers must first be computed. We use the *move_base* global planner, *navfn*[7], for the computation of these paths.

3.2.3. Path utility

Our utility function is based on a Shannon and Rényi entropy formulation developed by Carrillo et al. [1]. Compared to other active SLAM approaches, most of them only use the Shannon entropy. But this is in general not a good choice since the map and pose entropies are fundamentally different quantities. The entropy of the pose of the robot, which is a continuous random variable, can take any real value while the entropy of a discrete random variable, such as the grid cells in our map, takes only strictly non-negative values. As shown by Carrillo et al. [1], these two entropies can have vastly different numerical values in an autonomous exploration task.

Rényi's entropy can be seen as the generalised Shannon's definition of entropy and is defined by:

$$H_\alpha[P(x)] = \frac{1}{1-\alpha}\log_2\left(\sum_{i=1}^{n} p_i^\alpha\right) \tag{3.2}$$

[5] http://wiki.ros.org/frontier_exploration, (Accessed on 20.12.2018)

[6] http://wiki.ros.org/move_base, (Accessed on 21.12.2018)

[7] http://wiki.ros.org/navfn, (Accessed on 21.12.2018)

Shannon's entropy is a special case of Rényi's entropy in the limit of $\alpha \to 1$. Jumarie [40] presents a definition of mutual information as the difference between the Shannon entropy of the probability distribution $P(x)$ and Rényi's entropy of the same distribution with $\alpha = c$

$$I_c[P(x)] = H_{\alpha=1}[P(x)] - H_{\alpha=c}[P(x)], \tag{3.3}$$

where I_c is called the mutual information. The first term on the right hand side of the equation is Shannon's entropy and the second term is Rényi's entropy. The parameter α can be seen as a gain coefficient, which measures the efficiency of an observer who is considering the distribution $P(x)$. Based on the mutual information Carrillo et al. [1] defined a new utility function as:

$$\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}} I_{c(\boldsymbol{a})}[P(\boldsymbol{m}|\boldsymbol{x}, \boldsymbol{d})], \tag{3.4}$$

where $\boldsymbol{a}$ are the possible future actions and $P(\boldsymbol{m}|\boldsymbol{x}, \boldsymbol{d})$ is the current distribution over possible maps. The value of $\alpha = c(\boldsymbol{a})$ depends on the possible future actions $\boldsymbol{a}$. $\boldsymbol{d}$ is the history of data, i.e. control inputs and sensor measurements. Note that computing Equation 3.4 does not require updating the map using possible future measurement, which would require propagating sensor and localisation uncertainties forward. Additionally, the utility function is computed only over the regions of the map that will be visited during the action $\boldsymbol{a}$ as was similarly proposed by Stachniss et al. [8]. This leads to the formulation:

$$\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}} \sum_{m \in \boldsymbol{m}(\boldsymbol{a})} H[P(m|\boldsymbol{x}, \boldsymbol{d})] - H_{\alpha(\boldsymbol{a})}[P(m|\boldsymbol{x}, \boldsymbol{d})], \tag{3.5}$$

where $\boldsymbol{m}(\boldsymbol{a})$ is the subset of the current map that may be visible to the robot when following the action plan. $\boldsymbol{m}(\boldsymbol{a})$ can be computed with standard ray tracing techniques, here we use Bresenham's line algorithm [41][8]. For the computation of the Shannon entropy in Equation 3.5, we used the same equation for the Shannon entropy of the map distribution as given by Stachniss et al. [8]:

$$H[P(m|\boldsymbol{x}, \boldsymbol{d})] = -\big(P(m)\log_2\big(P(m)\big) + \big(1 - P(m)\big)\log_2\big(1 - P(m)\big)\big). \tag{3.6}$$

In general, the utility function (see Equation 3.5) will choose actions that will most improve the map estimate of the partially explored areas of the map, i.e. where $P(m) \neq \{0, 0.5, 1\}$. These situations arise when the robot has poor localisation during its pass through an area. For further details on properties of the utility function, the reader is referred to [1].

The parameter $\alpha(\boldsymbol{a})$ is related to the predicted uncertainty in the pose of the robot after taking action $\boldsymbol{a}$, such that it will decrease the information gain when the pose uncertainty is high. When the robot has minimal uncertainty on the pose estimate then the information gain should be maximal. Similarly, when the uncertainty is high the information gain should be zero, which causes the robot to perform exploitation over the current map in order to decrease its pose uncertainty by choosing the appropriate path. In the case of our utility function we want $\alpha \to 1$ when the uncertainty becomes infinite, since the two entropies cancel out. Conversely, we want $\alpha \to \infty$ when the pose uncertainty approaches zero since this minimises Rényi's entropy. A simple candidate for α is:

$$\alpha = 1 + \frac{1}{\sigma}, \tag{3.7}$$

where σ is a scalar representation of the predicted pose uncertainty. Carrillo et al. [1] propose to compute the uncertainty scalar σ by optimality criteria from the Theory of Optimal Experimental Design (TOED) [4]. They propose to use one of the three most commonly used criteria: A-optimality, D-optimality or E-optimality. The criteria can be applied to either the full covariance matrix or to the marginal covariance matrices from each node in the action pose graph. The action pose graph is explained further bellow. We use the D-optimality as it minimises the volume of the covariance matrix ellipsoid and is defined as:

$$\sigma = \det(\boldsymbol{\Sigma})^{\frac{1}{n}} = \exp\Big(\frac{1}{n}\sum_{k=1}^{n}\log(\lambda_k)\Big), \tag{3.8}$$

where n is the dimension of the covariance matrix and λ_k is the k-th eigenvalue of the covariance matrix.

[8] Implementation is based on algorithm found on https://csustan.csustan.edu/~tom/ Lecture-Notes/Graphics/Bresenham-Line/Bresenham-Line.pdf, (Accessed on 21.12.2018)

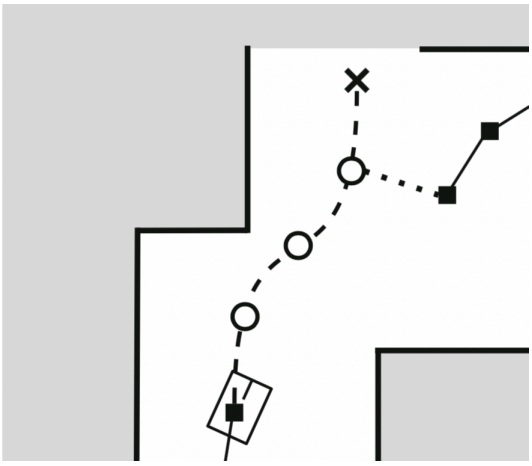
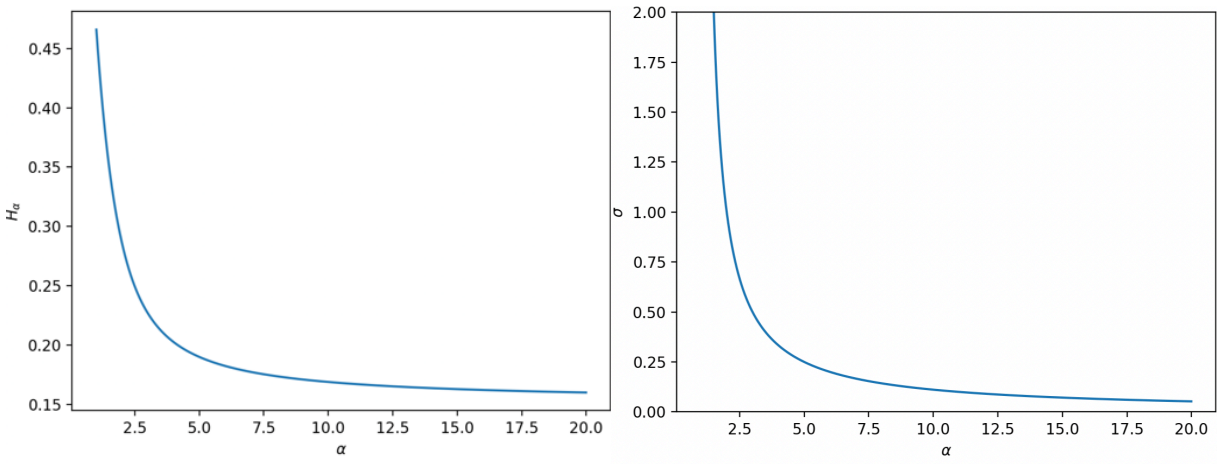


Figure 3.4: The action pose graph is a miniature graph, which represents the action plan. The black square boxes are nodes from the previously built factor graph during exploration. The cross describes the goal of the current action. The circle corresponds to future nodes added to the action pose graph along the action plan. Figure from [1].

For the computation of the path utilities, we need to compute the uncertainty prediction of the future paths. As proposed by Carrillo et al. [1] we use a miniature pose graph representing the action plan, which is also called the action pose graph. We initialise the graph with an initial factor at the robot's current estimated location with the covariance matrix taken from the most recent node in the graph. The marginal covariances can be extracted using GTSAM, which uses the method presented by Kaess and Dellaert [42]. To build our action pose graph, we interpolate the action plan, which we get from our global planner, with some fixed step size and add pose nodes along the length of the path. As odometry constraints between the nodes, we use fixed covariance values such that longer actions will lead to larger increases in uncertainty. The fixed covariance values are set according to average values resulting from our laser scan matcher. If the action plan takes the robot close to other nodes in the existing graph, the robot adds additional factors if the current distance in the estimated locations is smaller than a certain threshold, as has been already done in our SLAM framework. In [1], the authors use the number of FLIRT features in the laser scans to decide whether or not to add a new factor. To get the covariance matrices of the action pose graph, we use the iSAM2 library to optimise the action pose graph. Now we can compute the utilities and the best action a to execute with Equation 3.5. For the computation of the D-optimality criteria we use the marginal covariance matrices as different nodes may have different α values. If the full covariance matrix is used, only a single α value would exist for the entire path. Additionally, for the computation of the map subset $\boldsymbol{m}(\boldsymbol{a})$, we need to find the last node j in the action pose graph from which the cell was visible and then use that α_j to compute the information gain in that cell.



(a) Rényi's entropy (b) α function

Figure 3.5: (a) shows the plot of Rényi's entropy and (b) shows the plot of the α function.

Analysis of the α function

A full justification and analysis of the original α function was not provided in Carrillo et al. [1]. Instead the function was chosen largely due to the constraints of the utility function. However, to ensure that the α function acts as desired, we conducted a deeper analysis into the sensitivity of the system with respect to its formulation.

Assume we have a map cell with the probability $P(m) = 0.1$ of being occupied and a pose uncertainty with a standard deviation of the size of the map resolution, e.g. the standard deviation in x is equal to the map resolution and is zero for y and θ. This pose uncertainty results in $\sigma = 0.0025$. Figure 3.5 shows two plots, one for Rényi's entropy and the other for the α function using our numerical example values. For both functions we can see that for higher α values the graph gets flatter. If we now try to find the corresponding α value for our σ value, we notice that it will be far outside of the plot range. If we compute the value we get $\alpha = 401$. If we now try to find the corresponding value for the Rényi entropy, we can see that if we have α values in this range, the Rényi entropy will be almost constant. This means that it will have almost no influence on the utility computation. We will only see a bigger influence if we would have a lot of map cells with high uncertainties. But this should not be the case as we normally just plan actions in the sub-part of the environment we want to explore. If we would need to define a region for α where its effect on the utility has an influence, we could define this between $\alpha = 1$ and $\alpha = 20$.

Figure 3.6: The geometric relationship for the orientation error φ with respect to the map resolution (map_res). l corresponds to the distance from the robot at which the error should be equal to the map resolution.

Normalised α function

We propose to use an α function that scales depending on the map resolution. Our proposed function uses a normalisation value to enforce the influence on the utility depending on how the normalisation is chosen. The function is defined as:

$$\alpha = 1 + \frac{\sigma_{norm}}{\sigma}, \tag{3.9}$$

where

$$\sigma_{norm} = \exp\left(\frac{1}{n}\left(2\log(map\_res^2) + \log\left(\tan^{-1}\left(\frac{map_{res}}{l}\right)^2\right)\right)\right). \tag{3.10}$$

The inverse tan term equals φ in Figure 3.6. In Equation 3.10 we use the D-optimality (Equation 3.8) for the case where we would have a pose uncertainty with standard deviations of the map resolution in x, y and θ. For θ we defined an orientation error φ, which defines a distance l from the robot at which the error is equal to the map resolution. Figure 3.6 shows an illustration of the mathematical relation. For the following experiments, we used $n = 3$, $map\_res = 0.05$ and $l = 10m$. Our α function causes α to have a value of 2, if we have a pose uncertainty in the range of the map resolution as defined for σ_{norm}.

3.3 Detection and tracking of moving people in 2D laser scans

The robustness and precision of our mapping and localisation solutions rely on the ability to detect and remove scan points that correspond to dynamic components of the environment, that is, returns generated by other pedestrians in the crowd. The final goal of the CROWDBOT project is to integrate the pedestrian tracker supplied by RWTH to aid in this task. However, as a first implementation, we have first developed a method that does not rely on high fidelity people tracking with RGBD sensing. The method described below only uses the same 2D laser scan data needed for the raw mapping task and is relatively lightweight in terms of computation.

3.3.1. Classifying dynamic scan points

To simplify the task of finding scan points in the laser data that correspond to moving people, we first remove scan points that correspond to static objects from the raw laser data. We can do this by using the current occupancy grid map we have built so far during exploration. Figure 3.7 shows this idea visually. This simplification assumes that our current occupancy grid map is correct.

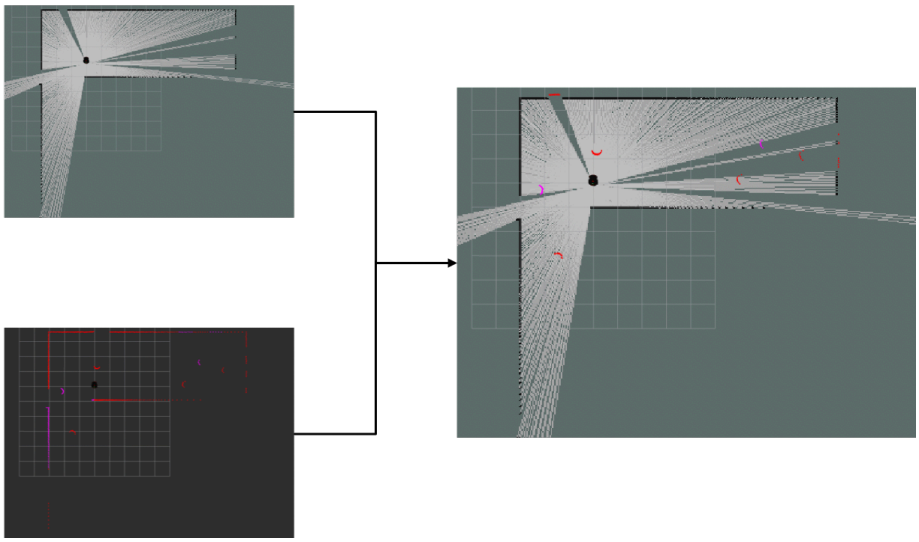


Figure 3.7: Static objects in the current occupancy grid map (left upper image) are removed from the current laser scan (left lower image). The right image shows the current occupancy grid map and the overlaid adapted laser scan.

This method relies on the availability of an existing portion of the occupancy grid map. Thus, at initialisation, when no occupancy grid map is available, we first record the laser scans over a specified period of time. During this time, we list the range data corresponding to each scan angle. We then search for the median range values of each angle and check if at least 50% of the data is in a range around the median value. If this occurs, we assume that the median value corresponds to a static object and we send these points in the laser scan data to our SLAM framework.

As soon as we have a candidate scan (with removed static scan points), we can cluster the data using an adaptive breakpoint detector [16]. Figure 3.8 shows an example of how the clustering could look. We additionally save the information of whether an object is occluded or not during clustering. This is done by verifying if on both sides of the clusters the following scan point is closer or further away from the robot. The raw laser data is used for this task, as the candidate scan might not have this information anymore after removing the static scan points. The occlusion information is essential to later distinguish if an object can be dynamic. For example, when a clustered object is occluded, the computed centroid of the cluster could move over time, even though it is static. This can be caused by objects moving in front of the occluded object or other occlusions that occur during exploration.

We then compute the centroids of the clusters and use this information for tracking. We use individual Kalman Filters [43] for each tracked object, which encode a constant acceleration model. By using a Kalman

Filter we can extract velocity estimates of the tracks during exploration. This is important to distinguish if a track is assigned as dynamic or not. The data association during the update step in the Kalman Filter is currently solved by searching for the closest measurement around a track. To further improve the association, the Mahalanobis distance [44] could be used instead. If no measurements are available for updating a track, the algorithm waits for the next iteration. Tracks are only removed if no measurements are received over a longer time.

To distinguish if a track is dynamic or static we defined a process where the tracks need to pass some checks before they are assigned as either. Figure 3.9 shows the main process of these checks. In the beginning when a new track is initialised it is assigned as an *unknown* track. Once a track is assigned as *static* or *dynamic*, the assignment is fixed. The first check is a counter and checks for how long a track is already unknown. If the track is unknown long enough, it is assigned as static.

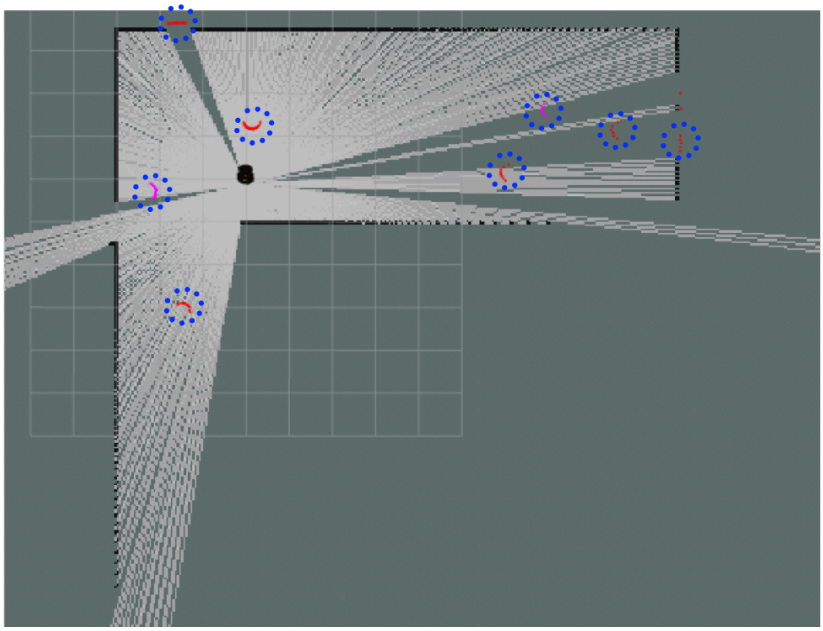


Figure 3.8: Example of how the adaptive breakpoint detector clusters the laser data, where the encircled laser points correspond to one clustered object.

The next check inspects the current cluster size of a track. If the size is bigger than the expected size of a person it is assigned as static. If only a few scan points are available, the track is kept as unknown. This is done to avoid having false dynamic detections caused by noisy measurements. If a track is not occluded and passes all size checks, the velocity estimates are used to detect if a track is dynamic. But to avoid noisy measurements resulting in high velocity estimates, the velocity is averaged over a period of time before it is compared to a threshold value.

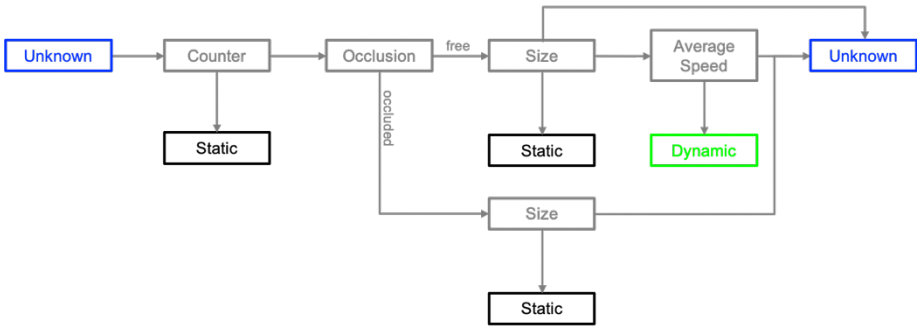


Figure 3.9: Flow chart showing the main process an identified track must pass to be assigned as a dynamic or static track. If the assignment fails, the track remains labelled as an "unknown" track and the process repeats at the next scan iteration.

3.3.2. Adaptive person ellipse

Following the recommendations from CROWDBOT D62 Robot Design Recommendations, the LIDARs are mounted close to the robot base (0.219m and 0.289m above the ground) to provide proximity data at the most likely point of contact. Thus, the LIDAR scans return measurements at roughly leg/ankle-height. The standard walking gait of a human makes it difficult to maintain a stable track of each leg since the stop-and-go movements of the legs make it challenging for the local planner to include the dynamic obstacles into the planning. To solve this issue, it is necessary to know which *clusters* correspond to an individual person. To do this, we use an adaptive ellipse, which changes its form depending on the current estimated speed and movement direction of the tracked person. Figure 3.10 shows the ellipse around a dynamic track.

Given a set of dynamic tracks, all clusters within an ellipse are merged together and the merged centroid is then used to update the track. The formula for the ellipse can be defined as:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{3.11}$$

where

$$a = c\left(1 + \frac{v}{v_{norm}}\right),\ b = c\left(1 + d\frac{v}{v_{norm}}\right) \tag{3.12}$$

describe the adaptive axes and where v corresponds the norm of the velocity. The parameters c, d and v_{norm} depend on the proportions of a human. We found that $c = 0.35m$, $d = 0.1$ and $v_{norm} = 0.8m/s$ were good approximations for an average-sized person.

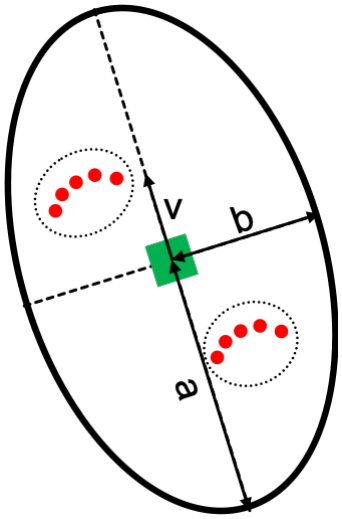


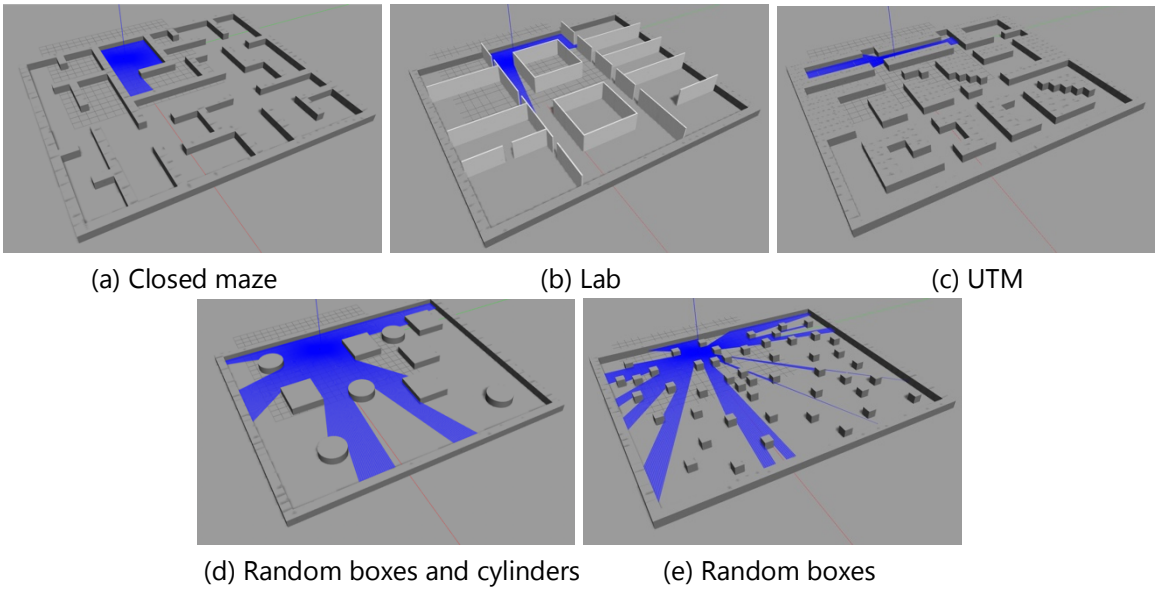
Figure 3.10: The adaptive person ellipse merges leg clusters together. The green block represents the current dynamic track around which the adaptive ellipse is used. The red points show laser scan points corresponding to the legs.

3.4 Mapping results in simulation

We now present test results obtained in our simulation environments. We primarily compare our active SLAM framework with a shortest frontier strategy, which selects the next global waypoint as the nearest waypoint to the robot's current location (ignoring obstacles). Section 3.4.1 shows the different simulation environments that we test in. Section 3.4.2 describes how dynamic obstacles are introduced into the simulation environment. Section 3.4.3 details the various metrics we use to measure the resulting map quality. Section 3.4.4 presents the results and accompanying analysis.

3.4.1. Simulation environment

For algorithm development and testing purposes, we created a set of Gazebo[9] environments in which to run a basic Pioneer 3-DX robot with a mounted 360° 2D LIDAR[10], see Figure 3.11. The LIDAR setup on the simulated Pioneer robot is similar to that used on the CROWDBOT Pepper robot. However, on Pepper, we use two 2D LIDARS, each with 270° coverage, mounted at the front and back of the robot base to ensure 360° planar coverage. In effect, the two robots have access to the same LIDAR information needed for our active SLAM algorithm. The simulated Pioneer 3-DX robot includes an additional challenge in the fact that, unlike Pepper, it is a differential drive robot rather than a holonomic robot. A nonholonomic robot is unable to drive freely in any direction and can increase the difficulty of motion planning. We expect that if our active SLAM framework is successfully deployed on a nonholonomic robot, then the results will translate to a holonomic robot platform.



(a) Closed maze (b) Lab (c) UTM

(d) Random boxes and cylinders (e) Random boxes

Figure 3.11: The worlds used for testing and algorithm development as shown in the Gazebo simulator. These environments have been designed such that most scenarios the robot will see are covered. (a) is a closed maze world where bigger loop closures happen less often than in other worlds. (b) shows an environment similar to the floor plan of the Autonomous Systems Lab at ETH Zürich. (c) shows a suburban street map used for UAV traffic management (UTM). (d) and (e) are random maps built from boxes and cylinders placed at random positions. The blue areas in the worlds indicate the LIDAR rays of the Pioneer 3-DX.

3.4.2. People/dynamic object simulator

To simulate a crowded environment, a people simulator has been developed. It is based on a simulation used in the work of Pfeiffer et al. [45], which uses a social forces model for pedestrian dynamics [46]. In particular, the interaction between robot and people has been added and for simplicity the same social forces model has been used. For testing purposes, we can define how many people are spawned in the world and test how our framework behaves. People are moving randomly in the world by following a random goal. In Figure 3.12 an example of the people simulator is shown.

[9] http://gazebosim.org, (Accessed on 04.01.2019)

[10] Implementation is based on https://github.com/JenJenChung/pioneer_description, (Accessed on 04.01.2019)

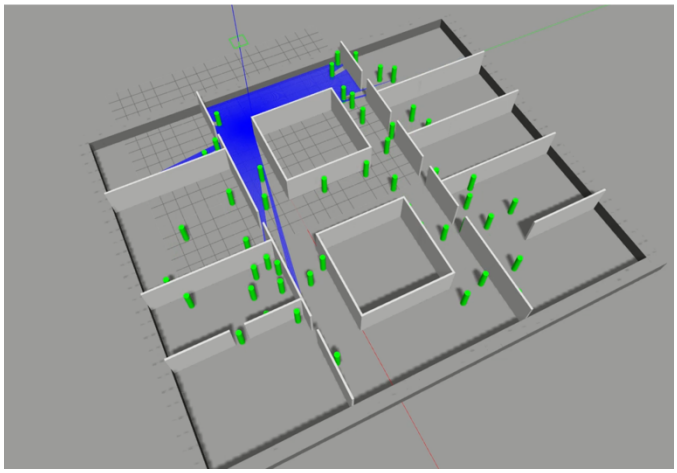


Figure 3.12: 50 simulated pedestrians/dynamic obstacles are shown as green cylinders in the Lab world.

3.4.3. Metrics for map quality

As map metric, we defined three different metrics, which compare the generated map with the ground truth map. For all three metrics, a lower score indicates a map that is closer to the ground truth map. The first metric is a simple sum of squared errors of the occupancy probability in the corresponding map cells, which we call the map score [47] metric:

$$map\_score = \sum_{i,j} \left(P(m_{i,j} - P(n_{i,j}) \right)^2, \qquad (3.13)$$

where $P(m_{i,j})$ and $P(n_{i,j})$ are the occupancy probabilities of the maps m and n at indices i and j. The problem with this score is that it penalises the mapping behaviour. As the mapping algorithm tends to build maps with thicker walls, when the same region is visited more often, the score increases. See Section 3.1.3 for further information on the mapping behaviour. As the mapping behaviour has no influence on the map quality, a better metric is necessary to describe the map quality. To handle errors induced by the mapping behaviour, we defined a signed distance field (SDF) score, which should decrease the impact of wall thickness on the score. The SDF score is defined as:

$$SDF\_score = \sum_{i,j} (s_{i,j} - t_{i,j})^2, \qquad (3.14)$$

where s and t correspond to the distance values in the SDF maps. Before we can compute the SDF score, we need to compute the SDF maps of the estimated occupancy grid map and the ground truth map. Each distance value corresponding to a cell describes in our case the distance in number of cells to the closest wall cell. If a cell corresponds to a wall cell, the negative distance to the closest wall cell, which is adjacent to a free cell, is computed. For simplicity, unknown cells are treated as wall cells. To further enforce the score on navigationally important areas, a relaxed SDF score has been defined. The relaxed SDF score uses two relaxations. The first relaxation handles errors caused by wall shifts as induced for example by our mapping behaviour, while the second focuses map regions that are important for navigation. The score is formulated as:

$$rel\_SDF\_score = \sum_{i,j \in IJ_{nav}} \begin{cases} \left(|s_{i,j} - t_{i,j}| - 1 \right)^2, & if\ |s_{i,j} - t_{i,j}| > 1 \\ 0, & otherwise. \end{cases} \qquad (3.15)$$

where $s_{i,j}$ and $t_{i,j}$ are the distances at indices i and j of the SDF maps. In case of $|s_{i,j} - t_{i,j}| > 1$ we subtract one cell number from the current distance error to relax the score. The set IJ_{nav} corresponds to relaxed map region indices, which focus on navigationally important areas. These areas correspond to regions, which are not unknown in the ground truth map. We consider wall cells, which are not adjacent to a free cell, as unknown cells and are thus not considered in the score.

3.4.4. Mapping performance

Static environments

For the evaluation of map quality and pose uncertainty performance, five test runs for each world and each strategy have been conducted. In Figure 3.13 resulting maps in the lab world are shown as example. It can be observed that the utility strategy revisits previously observed regions more often. But as can be observed, it is hard to find any map differences in the images. Because of this, the quantitative evaluation was performed using the map metrics defined in Section 3.4.3.

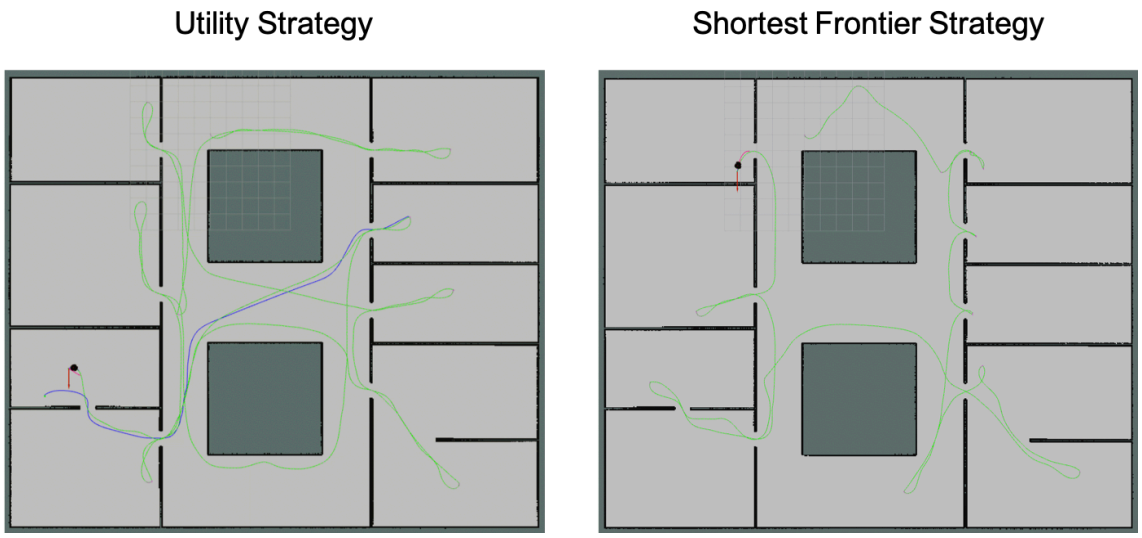


Figure 3.13: An example of the resulting maps for the proposed utility strategy and the comparative shortest frontier strategy in the Lab world. The green lines correspond to the paths executed by the Pioneer in the simulation. The blue path corresponds to the most recent path considered by utility computation.

In Table 3.3 the map evaluation for all test runs is shown. We can observe that in general the exploration times for the utility strategy are higher than for the shortest frontier strategy. This is caused by the behaviour of the utility strategy, which always tries to decrease map and pose uncertainty in an optimal way. To decrease both uncertainties in an optimal way, revisiting previously observed regions is essential. The utility strategy achieves better results in the Lab and UTM worlds compared to the others. This could be the result of the bigger loop closures that are possible in the Lab and UTM worlds. In general, the results do not show a clear tendency for better results achieved by the utility strategy.

Table 3.3: Evaluation of tests in the five different worlds. The utility and shortest frontier strategy are compared. The right column shows the difference of these strategies as a percentage, where negative values signify the better score for the utility strategy.

	Shortest frontier	**Utility**	**Difference (%)**
	Closed Maze		
Exploration time	309.3416	535.3938	73.08
Map score	3029.928	3361.854	10.95
SDF score full	20014.24	29145.02	45.62
SDF score relaxed	340.7926	311.0176	-9.57
	Lab		
Exploration time	182.7874	303.0444	65.79
Map score	3292.426	3021.506	-8.97
SDF score full	37227.86	17580.24	-111.76
SDF score relaxed	216.45974	94.7785	-128.38

	UTM		
Exploration time	295.9834	440.1432	48.71
Map score	3707.704	3524.11	-5.21
SDF score full	93210.26	60568.1	-53.89
SDF score relaxed	645.6154	402.1316	-60.55
	Random Boxes and Cylinders		
Exploration time	220.6572	319.0164	44.58
Map score	1702.314	1748.006	2.68
SDF score full	19769.74	23086.46	16.78
SDF score relaxed	190.3798	204.0614	7.19
	Random Boxes		
Exploration time	298.5364	453.9934	52.07
Map score	2503.804	2277.326	-9.94
SDF score full	59441.98	40365.44	-47.26
SDF score relaxed	947.2146	966.6586	2.05

In Figure 3.14 the development of σ along the path length for all Lab tests is shown. It can be observed that the σ values stay very low for both strategies. A growth tendency for the shortest frontier strategy can be observed while the utility strategy remains bounded.

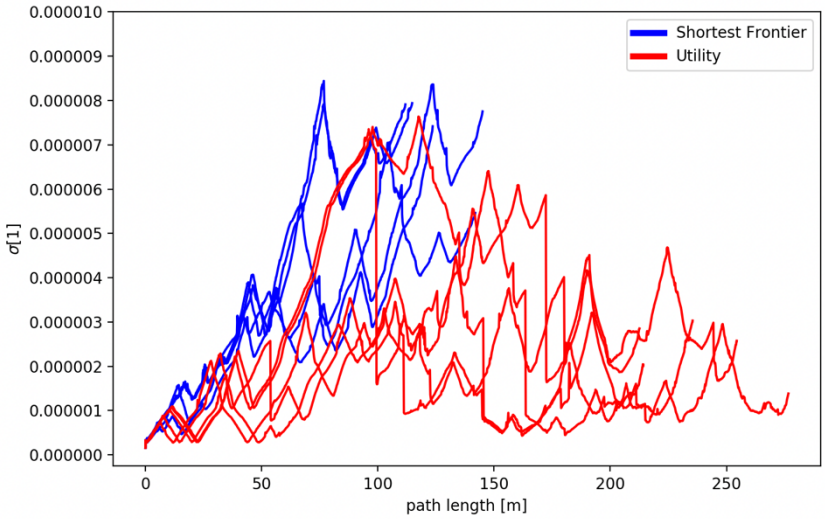


Figure 3.14: The development of σ along the path lengths for all test runs in the Lab world

We can further see the behaviour of the strategies when observing the paths executed by the Pioneer, where the chosen actions differ clearly. The utility strategy enforces revisiting previously observed regions more often to decrease its pose uncertainty. The shortest frontier strategy instead is not able to close bigger loops as it is pursuing the closest goal. Although we would think that the shortest frontier strategy would achieve lower mapping performance caused by the missing loop closures, the results in Table 3.3 show that this effect is not so severe. This can be explained by analysing Figure 3.14, where we can observe that independent of the strategy, the pose uncertainties remain very low. The standard deviations of the pose

uncertainties are much lower than our map resolution. Although in the simulation we use white noise on the laser scans with a standard deviation of 0.01 m, this seems to have little impact on the performance. When pose uncertainty is low, the utility strategy simply chooses actions that reduce the most map uncertainty, i.e. the most uncertain map cells which will be seen along the action plan. In hardware experiments, we expect that the localization uncertainty will be substantially higher in which case the utility strategy will be needed to manage uncertainties between the robot pose and map.

Crowded environments

Figure 3.15 compares the mapping performance in a crowded environment versus a static environment when all observed objects are treated as static. Severe errors and map shifts can be seen in Figure 3.15a. This highlights the need to appropriately deal with laser returns from moving people, since they can cause mapping and exploration failure. Figure 3.16 shows the resulting map when crowded environments are handled appropriately, as presented in Section 3.3. The mapping performance is comparable to that of the static case.

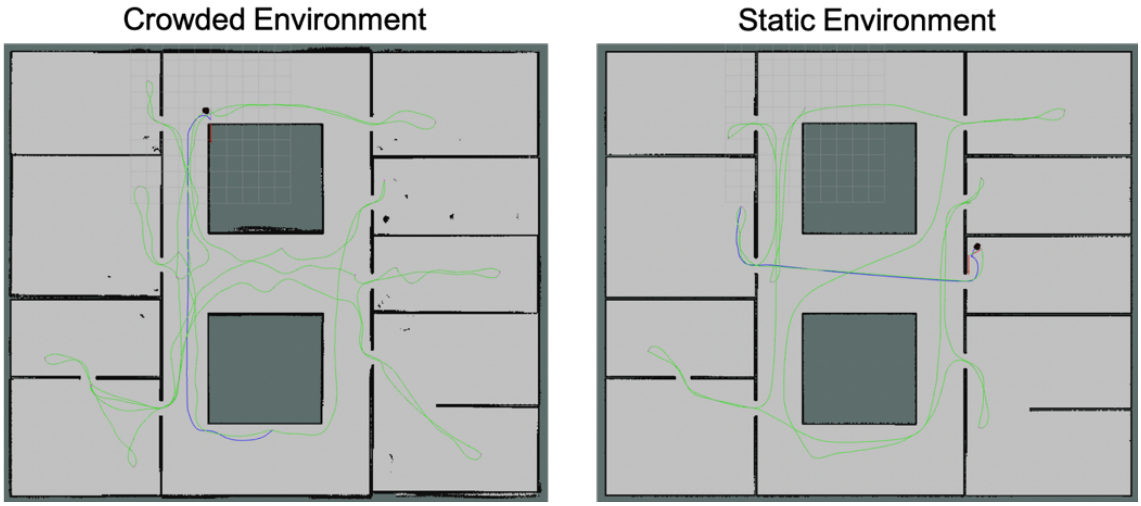


Figure 3.15: Qualitative comparison of resultant map when SLAM is performed under the assumption of a static environment. On the left, the simulation environment was populated with 50 dynamic obstacles, resulting in numerous places where those obstacles were incorrectly recorded as part of the static map. On the right, the mapping performance under static conditions is shown for reference.

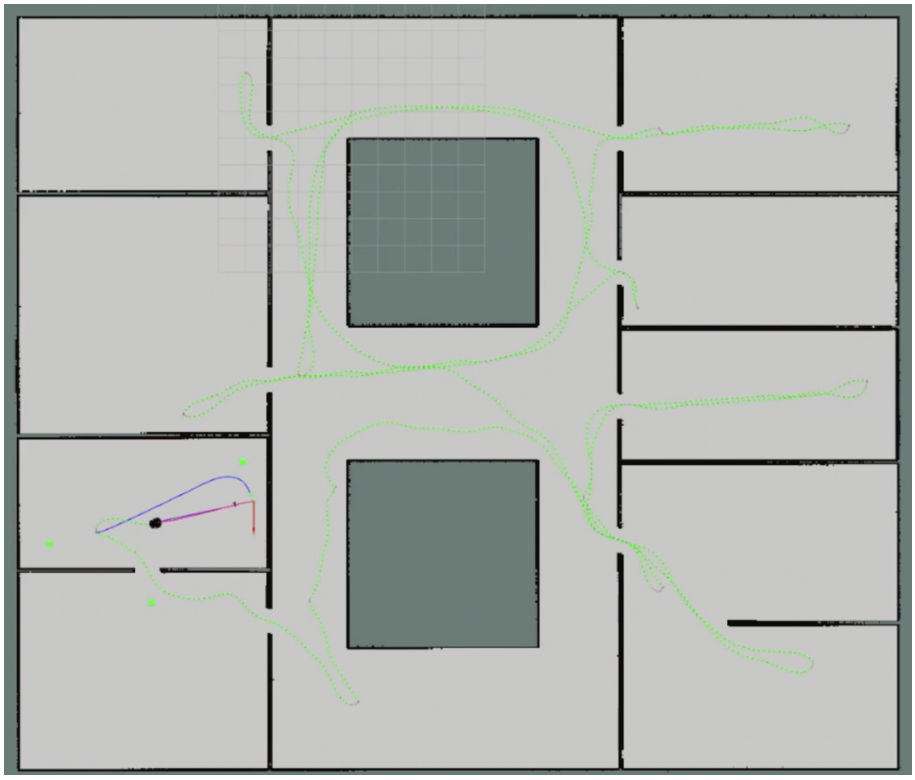


Figure 3.16: The resultant map when detection and tracking of moving obstacles is included in the mapping framework. We are able to recover the static environment despite occlusions from dynamic obstacles in the environment. The green squares near the robot in the bottom left of the image correspond to the detected and tracked moving obstacles in the current scan.

3.5 Mapping demonstration on the Pepper robot

Testing our active SLAM framework on the Pepper robot was conducted at the Autonomous Systems Lab at ETH Zürich. We first present results for the static environment mapping case followed by the results in a dynamic, crowded environment.

Static environment

The results for the static environment were recorded without any moving people in the lab (apart from the person conducting the tests). Since we only use 2D LIDAR scans, some environmental adaptations were necessary to ensure the safety of the tests. For example, glass doors were covered with opaque material at laser height to avoid being treated as free space. Figure 3.17 shows a comparison of two resulting maps using the utility and the frontier strategy. The revisiting behaviour of the utility strategy is again clearly shown, Furthermore, it is able to close bigger loops in contrast to the shortest frontier strategy. In the resulting map for the shortest frontier strategy these missing loop closures cause severe map shifts. This shows that the utility strategy can achieve better results than the shortest frontier strategy in a real-world scenario.

In general, we observed the utility strategy achieving consistently better mapping results compared to the shortest frontier strategy. However, the utility strategy was also fallible to producing poor quality maps. These were mostly due to overlooking actions that were to essential reaching loop closure areas. This can be caused by missing frontiers, as those areas that have already been explored exhaustively will not generate potential global waypoints. Thus, this is not a fault of the utility function per se but is caused by the frontier exploration strategy itself and the fact that only one path to a frontier is considered. To improve this behaviour, loop closure areas could be included as possible goals for actions. Alternative paths to the same frontier could further favour the performance.

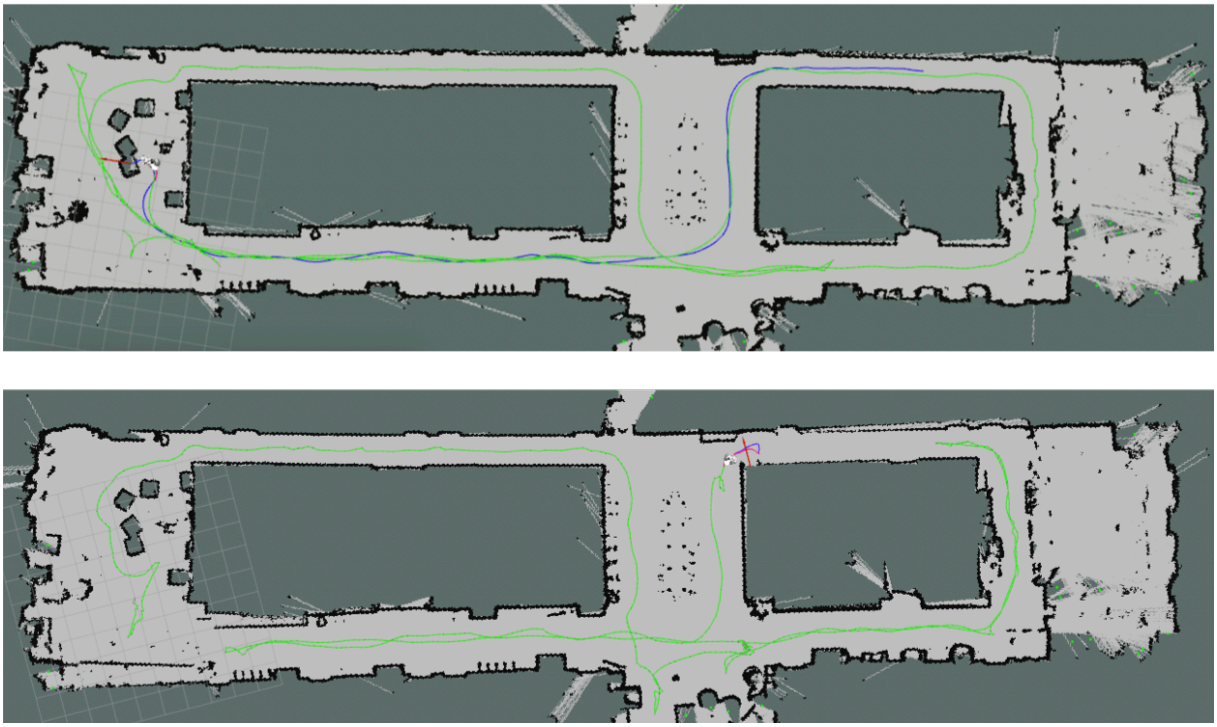


Figure 3.17: Resulting maps of the Autonomous Systems Lab at ETH Zürich (no dynamic obstacles present during mapping) using the proposed utility strategy (above) and the shortest frontier strategy (below).

Crowded environment

Given our previous results in simulation and in the static real-world environment, we only conducted tests with the utility strategy in the crowded environment. Figure 3.18 shows a screen shot during exploration where three people were walking around Pepper in a narrow corridor. The three people are tracked and included into the local planner and such that the robot can find a path to reach its goal.

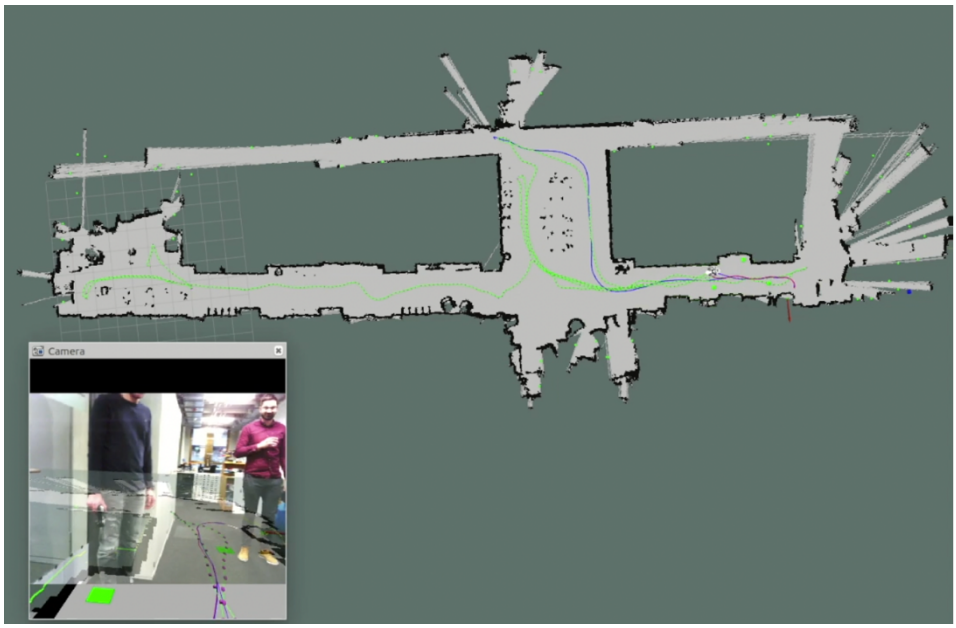


Figure 3.18: Screen shot of an exploration run with Pepper where three people were walking around the robot. The green blocks correspond to the detected and tracked moving people. The red path corresponds to the local plan computed to the current goal.

Figure 3.19 shows a mapping result using the utility strategy in a crowded environment. The exploration was actually successful, but one can observe that the map is not complete. This has been caused by an unfortunate coincidence. One laser beam has been used while mapping, which passes both frontier regions in the long and narrow hallway. This caused the frontier computation to cluster both frontier regions together. The computed centroid is then close to the mapped laser beam line. In this case, the global planner is not able to find a path to this frontier, as it is restricted to only plans paths though known areas. As can be observed, the implementation is not perfect and unexpected occurrences can lead to failures or undesired results.

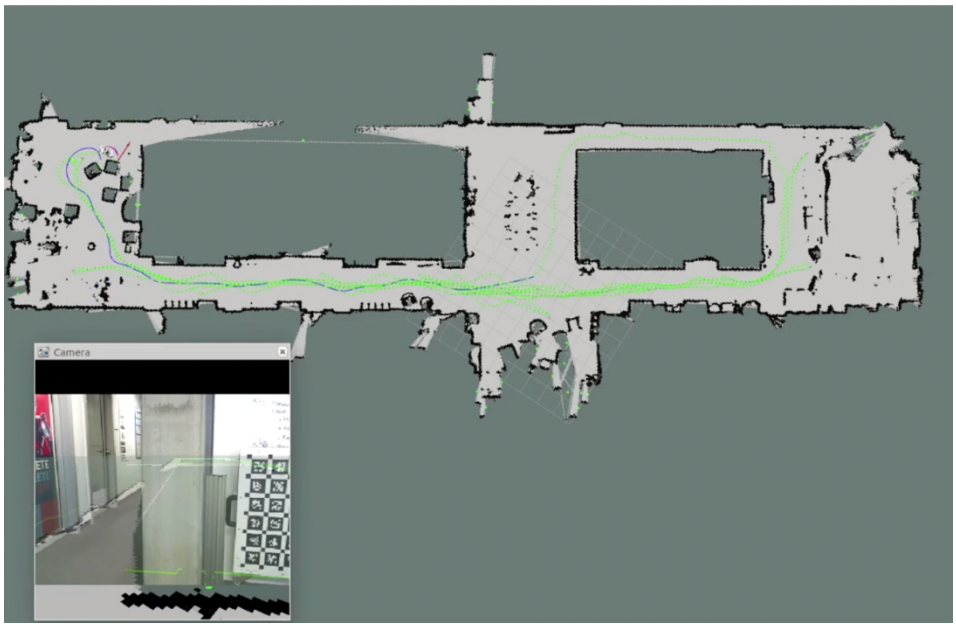


Figure 3.19: Screen shot of the resulting map using the utility strategy in the Autonomous Systems Lab at ETH Zürich.

During testing, further weak points stood out. As only laser odometry is used in the SLAM framework, the long and narrow hallway can cause problems. Especially while detecting and tracking moving people, we further remove scan points from our scan data. This means that our laser scan matcher has even less information from which to get the correct roto-translational parameters in a long, narrow hallway. Laser odometry failures often happen when only laser scan points corresponding to the walls of a hallway and no other object at the beginning or end of the hallway are available. The detection of moving people can also sometimes cause some issues, as the averaging of the velocity takes some time. If people walk too fast, the currently unknown leg tracks can be lost and reinitialised, such that there has not been enough time for the averaging of the velocity. Furthermore, fast-changing movement directions of people can also cause tracking failure.

3.6 Outlook for CROWDBOT mapping

The active SLAM framework developed for CROWDBOT has been shown to perform well in static and crowded environments. The results of our tests in simulation and hardware have demonstrated its efficacy in trading off between exploring new areas of the map and exploiting known regions to improve localisation accuracy. Our detection and tracking algorithm for identifying laser returns from people was also shown to produce substantial improvements to the resultant map quality by effectively de-noising the LIDAR scans prior to map updates.

Although the algorithm requires no tuning, the correct relationship between map and pose uncertainties in active SLAM is still an open issue. We could partially solve this issue with the definition of our α function, but this only results in a scaling relation to the map resolution. Scaling according to the map resolution can, in our opinion, be seen as a design choice, as it prevents having pose uncertainties that are higher than the

map resolution during exploration. This is not a requirement of an optimal exploration strategy, since it is possible that an action, which causes you to temporarily have a bigger pose uncertainty, can result in a better map and better localisation in the long run. The goal would be to find a general relationship between map and pose uncertainties for active SLAM.

Explicitly accounting for the movement of people could also improve the performance of our mapping routine. A highly crowded environment can cause the current framework to fail as the laser data may not have enough information after the detected moving people are removed from the scan. Including this information into the utility computation would open myriad solution possibilities. One idea would be to directly include the predicted future occlusions caused by moving people into the utility computation such that actions are also optimised to reduce occlusions.

Further improvements can also be made to the detection and tracking component. Specifically, by integrating the output of the CROWDBOT Perception work package (WP2). The RGBD learning-based framework for detecting and tracking people in WP2 would be especially well-suited to detecting people even when they are not moving or are not moving with a consistent motion. The incorporation of RGBD sensor data would also improve localisation and mapping performance by providing additional feature channels that are, for example, more robust to long hallway scenarios, transparent surfaces, etc.

4. Localisation

The focus of our localisation algorithm design is to achieve fast, accurate and prior-free initial localisation. Existing open source solutions, such as adaptive Monte Carlo localisation (AMCL)[11], require a prior on the robot pose at initialisation and demonstrate poor performance when the prior pose is poorly set. Similarly, the scan matching solution described in Section 3.1.1 also requires a prior on the robot pose to initialise the point-to-line optimisation routine.

The main methodology of our localisation solution lies in finding the pixel-wise alignment of the current session map to a provided global map of the environment. The current session map can be constructed using any mapping algorithm, e.g. the occupancy grid mapping algorithm described in Section 3.1.3, or the *slam_gmapping*[12] ROS node. The only requirement is that the session map resolution is the same as that of the provided global map. By performing a *map* matching routine rather than simply a *scan* matching routine, we are able to improve the accuracy of the localisation solution over methods that only use the current LIDAR scan. Furthermore, our method can maintain good localisation in highly crowded environments where sensor occlusions may be severe.

To maintain real time localisation, we implemented a branch and bound depth first search inspired by the Google Cartographer[13] package [45]. Details of our implementation are provided in the following sections followed by a quantitative analysis of the localisation performance compared to existing solutions.

4.1 Branch and bound map matching

Our branch and bound routine maintains computational efficiency by searching for matches between the occupied cells of the session map and those of the global map at increasing map resolutions. The lowest map resolution is computed as a function of the size of the session map.

$$H = \log_2\left(\max_{k \in i,j} 2k\right) - 1, \tag{4.1}$$

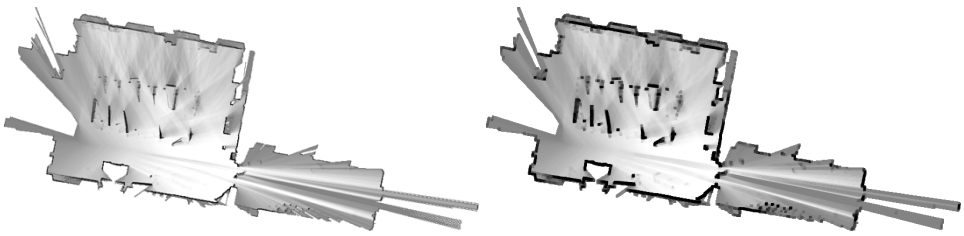
where i and j are the dimensions of the session map and H is the number of downsampling steps.

For each search query, the global map is first downsampled using a max operator, halving the resolution at each layer h until the lowest map resolution, $h = H$, is reached. The global map at each resolution (including the original map resolution, $h = 0$) is stored for fast lookup during the search. See Figure 4.1 for an example of the precomputed maps at decreasing resolutions.

In addition, the occupied cells in the session map are stored for N discrete rotation transformations, where N is determined according to the session map size, the original map resolution and a user-provided $rotation\_downsampling$ parameter which restricts the maximum pixel translation of a single discrete rotation step. That is,

$$dr = rotation\_downsampling \cdot \frac{1}{\sqrt{i^2 + j^2}}, \tag{4.2}$$

$$N = \frac{2\pi}{dr} - 1. \tag{4.3}$$



[11] http://wiki.ros.org/amcl, (Accessed on 20.08.2019)

[12] http://wiki.ros.org/slam_gmapping, (Accessed on 20.08.2019)

[13] https://google-cartographer.readthedocs.io/en/latest/, (Accessed 20.08.2019)

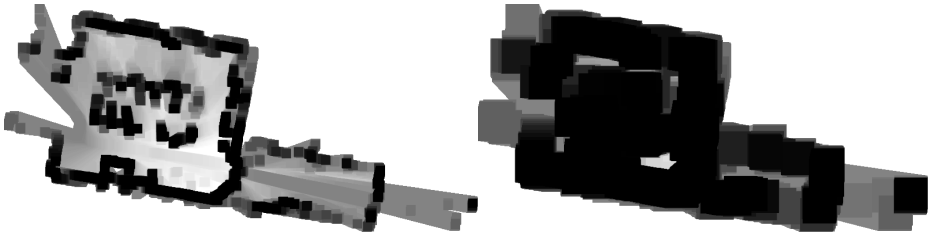


Figure 4.1: Precomputed grids at decreasing map resolution. Figure from [45].

Given the setup described above, the nodes in the search tree contain the (global map) resolution, (session map) rotation and translational information. The initial node list contains all the roto-transformations at the lowest global map resolution. An additional prior sorting according to available heading information can also be performed to further speed up the search.

Each node is scored and sorted according to the percentage of occupied cells in the rotated session map that are also deemed occupied in the current search resolution of the global map. A minimum match threshold is set initially to 50% such that any nodes that fall below this threshold are immediately pruned. Non-leaf nodes that pass this threshold score are expanded by increasing the global map resolution by one depth, effectively adding four more child nodes to the node list. Once a leaf node is reached (i.e. a node at the original map resolution) the existing minimum match threshold is updated to its node score. The search process continues until the node list is empty.

Algorithm 1 provides the pseudocode for our branch and bound depth first search routine.

Algorithm 1: Branch and bound depth first search

Initialise search problem
Compute and store $H + 1$ downsampled global maps
Compute and store N rotations of session map
$node\_list \leftarrow \emptyset$
$threshold \leftarrow 0.5$
For $n = 1{:}\,N$
  $new\_node(H, n, 0, 0)$
  $new\_node.compute\_score()$
  $node\_list.insert(new\_node)$
End For

Depth first search
While $!node\_list.empty()$
  $node \leftarrow node\_list.pop()$
  If $node.score < threshold$
    continue
  End If
  If $node.is\_leaf()$
    $threshold \leftarrow node.score$
    $best\_node \leftarrow node$
  Else
    $child\_nodes = node.expand()$

```
      ForEach child in child_nodes
        child.compute_score()
        node_list.insert(child)
      End ForEach
    End If
  End While
  Return best_node
```

4.2 Localisation performance

4.2.1. Results in simulation environment

We evaluated the performance of our map matcher against the localisation performance of AMCL in the UTM map shown previously in Figure 3.11c. AMCL strictly requires an initial estimate of the robot's 2D pose as well as an initial estimate of the pose uncertainty in the form of covariance values. However, we are interested in cases where no prior pose information is available. To simulate this with AMCL, we set the initial pose estimate to the centre of the environment and assigned initial x and y covariance estimates according to the maximum length scale of the map. The heading covariance was set at $\left(\frac{\pi}{2}\right)^2$. For both localisation routines, the robot was initialised in the environment and then teleoperated until either successful localisation was achieved or 60s had elapsed. The setup in Gazebo is shown in Figure 4.2, the corresponding initial AMCL and map matcher solutions are shown in Figures 4.3 and 4.4.

We compared the prior-free localisation performances for 25 different initial robot poses. Using our map matcher, the localisation solution is recomputed when a map update is generated by the underlying SLAM routine (in this case, *slam_gmapping*). However, AMCL continuously updates its localisation estimate given the incoming LIDAR scans and odometry messages. Thus, it is difficult to directly compare the time taken for successful localisation since this would relate more to the SLAM update rate rather than the map matcher routine itself. Furthermore, each map matching iteration is independent of the previous solution, unlike for AMCL, which uses a particle filter. Thus, for the proposed map matcher algorithm, we simply report the number of map updates needed for perfect localisation (to the resolution of the matched maps), whereas for AMCL, we report the time taken to achieve successful localisation (up to 180s), as determined by a pose error less than the resolution of the matched maps.

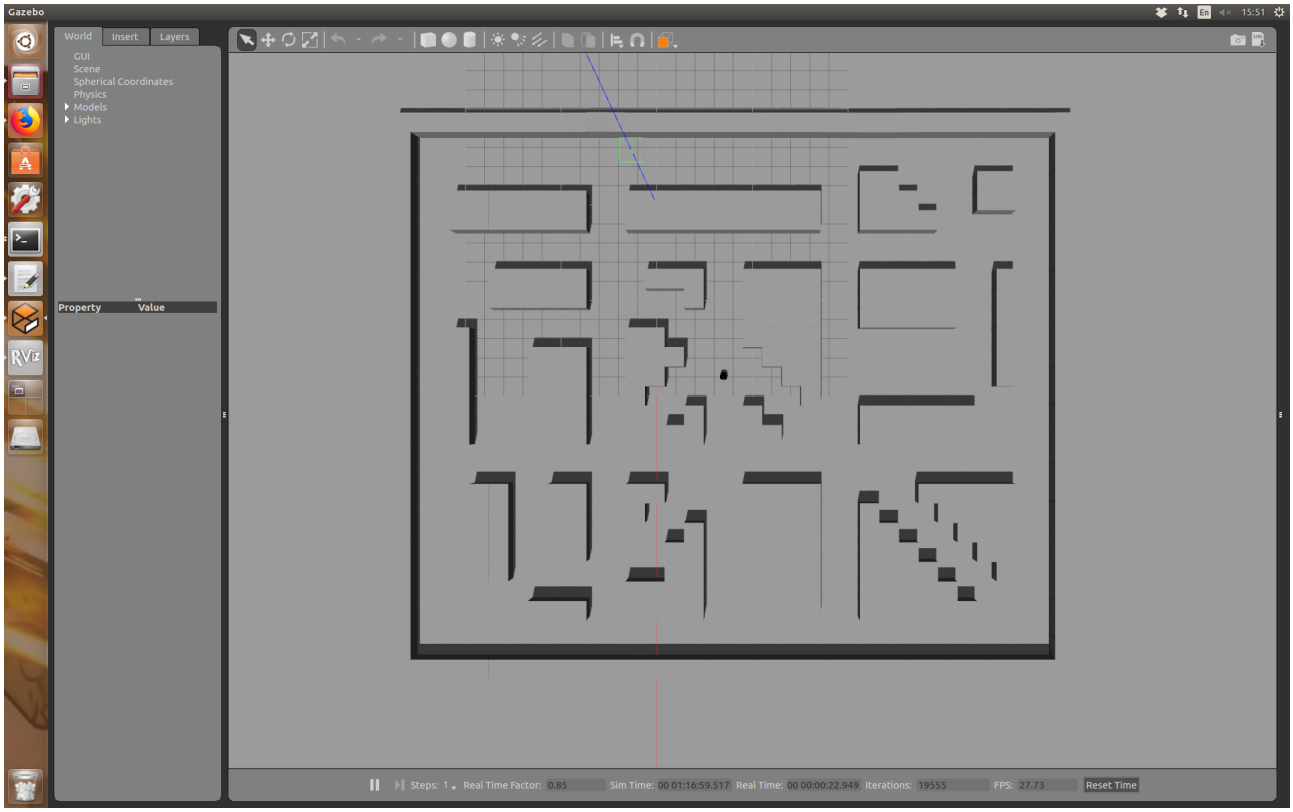


Figure 4.2: Gazebo environment used to evaluate our map matcher algorithm as compared to the ROS AMCL package. 25 different starting locations were tested. Here the robot starts close to the centre of the environment.

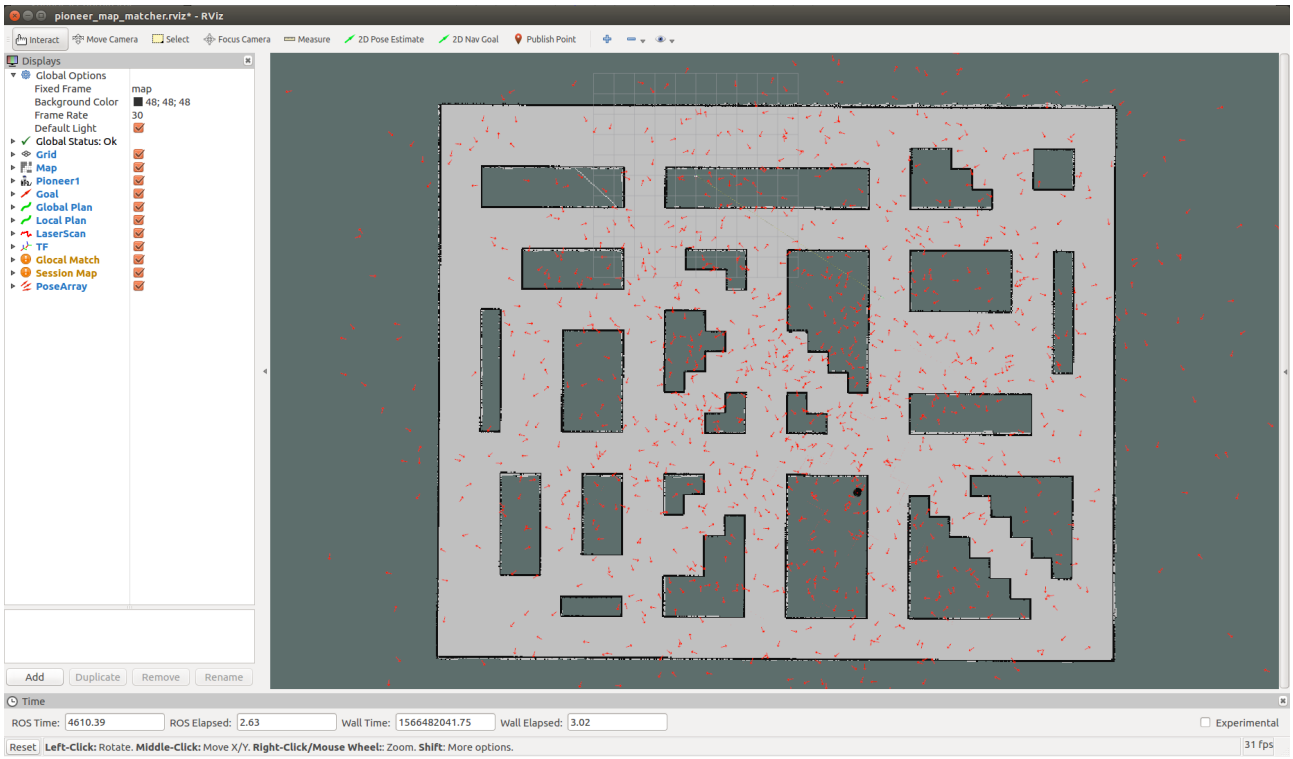


Figure 4.3: Screenshot from Rviz showing the initial AMCL solution at startup. The high pose covariances we set simulate initialising the particle filter with no pose information.

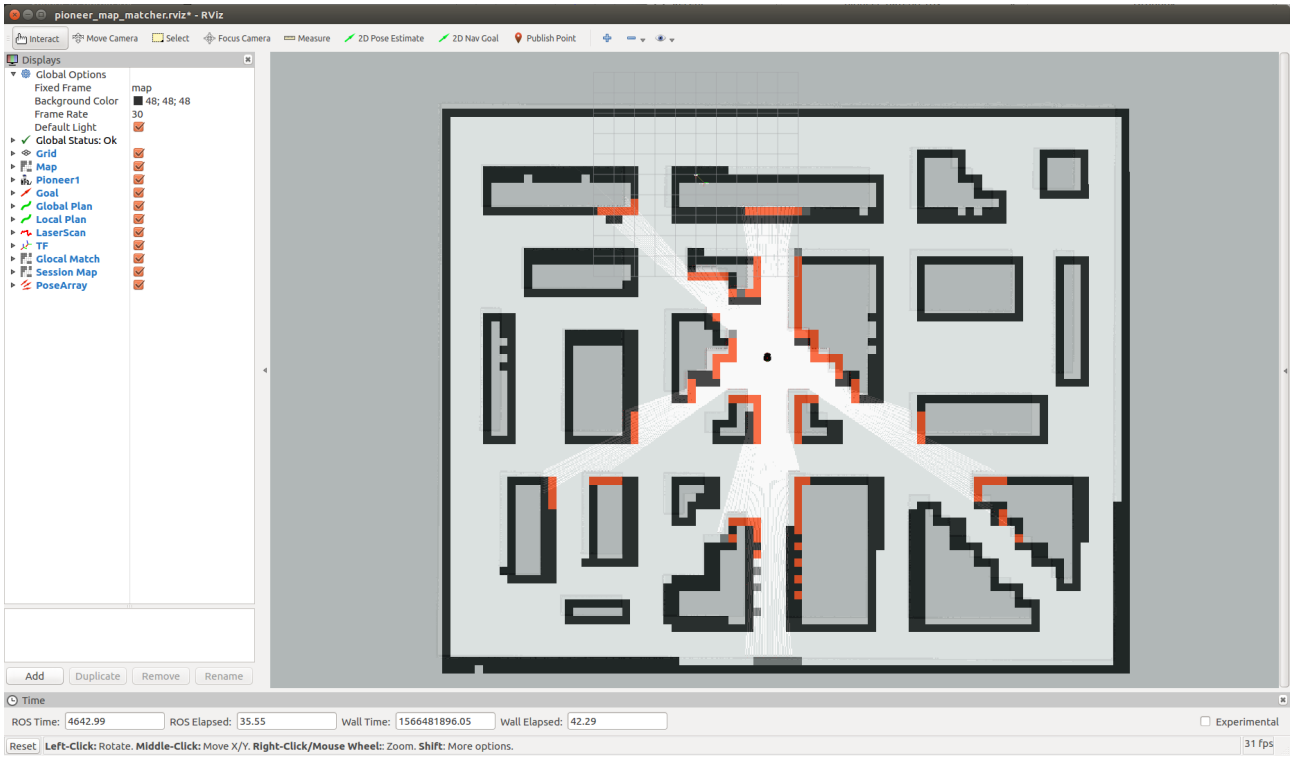


Figure 4.4: Screenshot from Rviz showing the initial map matcher solution after the first session map is received. In red are the correctly matched cells.

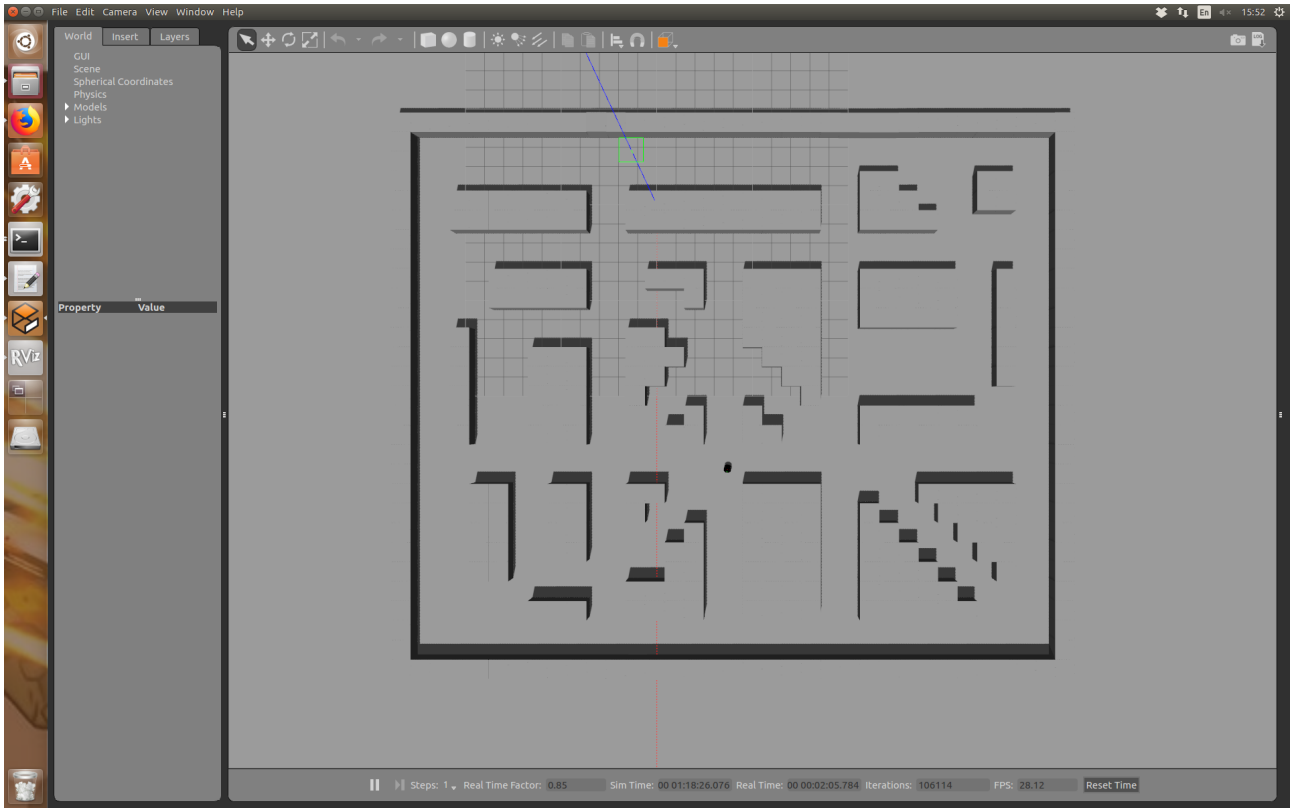


Figure 4.5: The true robot pose after approximately 10s teleoperation is shown in the Gazebo environment.

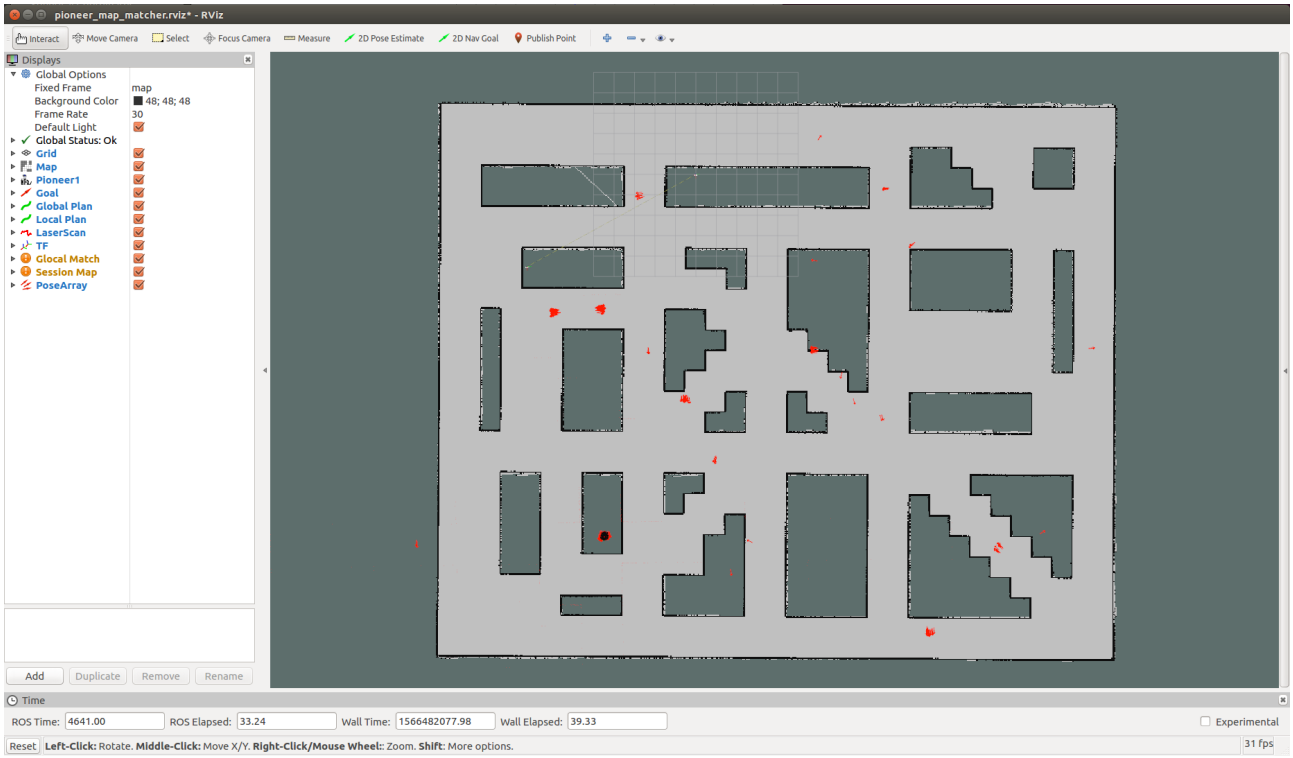


Figure 4.6: The AMCL solution after only 10s teleoperation rapidly reduces diversity in the particles and often converges to an incorrect robot pose. It is especially concerning that none of the remaining particle clusters are situated around the true robot pose, which is shown in Figure 4.5.

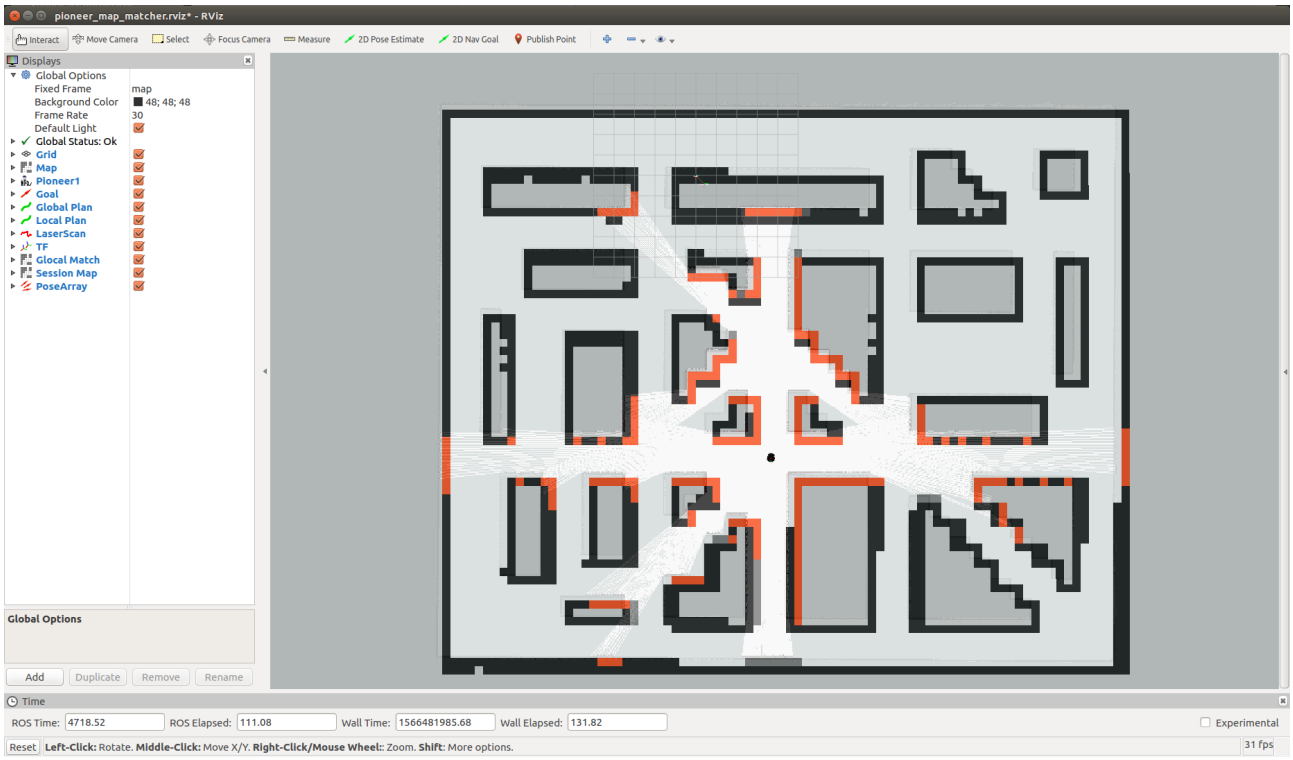


Figure 4.7: The map matcher localisation solution corresponding to 10s teleoperation. Localisation using the current session map and provided global map remains successful. The true robot pose is shown in Figure 4.5.

The results of our evaluation are shown in Table 4.1. It is clear that AMCL without initial pose information struggles to find a correct solution. In only three cases is the algorithm able to localise within the map

matcher resolution. Furthermore, even in the best result, AMCL took over 50s of teleoperation for the solution to converge. In contrast, our map matcher is able to localise using only the initial session map in 17 out of the 25 trials. Figures 4.4 and 4.7 also show an example of the map matcher solution over the first 10s of teleoperation. On average, the SLAM routine was providing map updates at a frequency of approximately 0.15Hz. Thus, in the worst case of 18 map updates for perfect localisation, our map matcher algorithm required approximately 120s. Nevertheless, the maximum deviation of the map matcher solution over all tested scenarios was (1,2) cells in the (i, j) map coordinates, which corresponds to (0.4m, 0.8m).

Table 4.1: Map matcher and AMCL localisation results on 25 different initial robot poses in the UTM world.

	Map Matcher		**AMCL**	
Initial Pose (m,m,rad)	Map updates until success	Max deviation (i, j)	Time to success (s)	Mean deviation (m, rad)
(-2,-11,0)	1	(0,0)	174.5	(7.8138 0.3985)
(-2,-2.5,0)	1	(0,0)	-	(11.2704 0.3525)
(-2.5, 9.5,0)	1	(0,0)	88.9	(6.9533 0.0879)
(-2.5,-19.5,0)	1	(0,0)	-	(17.9163 0.7288)
(2.5,-11,1.5708)	1	(0,0)	-	(22.8310 0.7357)
(4,-2.5,1,5708)	1	(0,0)	-	(15.2883 0.6656)
(2.5,3.5,0)	5	(1,1)	-	(27.8840 0.6953)
(2.5,9.5,-1.5708)	7	(1,1)	-	(18.5793 0.7873)
(2.5,18,-1.5708)	1	(0,0)	-	(16.6764 0.2948)
(7,-8,1.5708)	1	(0,0)	-	(33.4935 0.7234)
(9,3.5,0)	1	(0,0)	-	(18.0784 0.5335)
(8,9.5,-1.5708)	1	(0,0)	-	(26.2645 0.9079)
(8,15.5,-1.5708)	1	(0,0)	-	(31.2455 0.8400)
(13.5,-11,1.5708)	1	(0,0)	-	(22.6516 0.8209)
(13.5,-6.5,1.5708)	1	(0,0)	-	(18.3483 0.5122)
(13.5,-2.5,3.1416)	1	(0,0)	-	(25.1710 0.9695)
(13.5,3.5,3.1416)	1	(0,0)	51.6	(2.7967 0.1112)
(13.5,9.5,-1.5708)	1	(0,0)	-	(21.9407 0.9479)
(13.5,19.5,-1.5708)	1	(0,0)	-	(30.5773 0.4101)
(19.5,-6.5,1.5708)	4	(1,2)	-	(9.6721 0.4420)
(22.5,-11,1.5708)	10	(1,1)	-	(22.6288 0.9220)
(22.5,-2.5,3.1416)	18	(1,1)	-	(26.9089 0.7469)
(22.5,3.5,3.1416)	10	(1,1)	-	(34.3442 0.9964)
(22.5,9.5,3.1416)	5	(1,1)	-	(34.3554 0.9981)
(22.5,19.5,3.1416)	7	(1,1)	-	(26.2604 0.8520)

4.2.2. Results in real world office environments

Locomotec office

Several qualitative tests were conducted by Locomotec on a map of their office environment using recorded ROS bag data. The office is approximately 20mx20m in extent. Results are shown in Figure 4.8.

(a) (b) (c) (d) (e) (f)

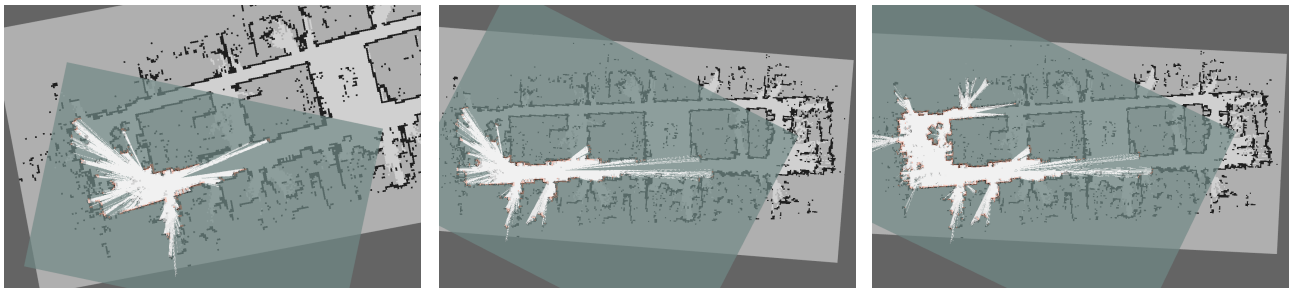
Figure 4.8: Pairs of sequential map matcher updates. Each row shows the map matching solution for consecutive online SLAM updates. Note that the method is robust to minor changes in the environment between the time when the original map data was collected and when the current map was built (some examples are circled in red).

Figure 4.9: Example of map matching failure. The true position of the robot is in the bottom right part of the map; however, the session map is matched to the top left part of the global map.

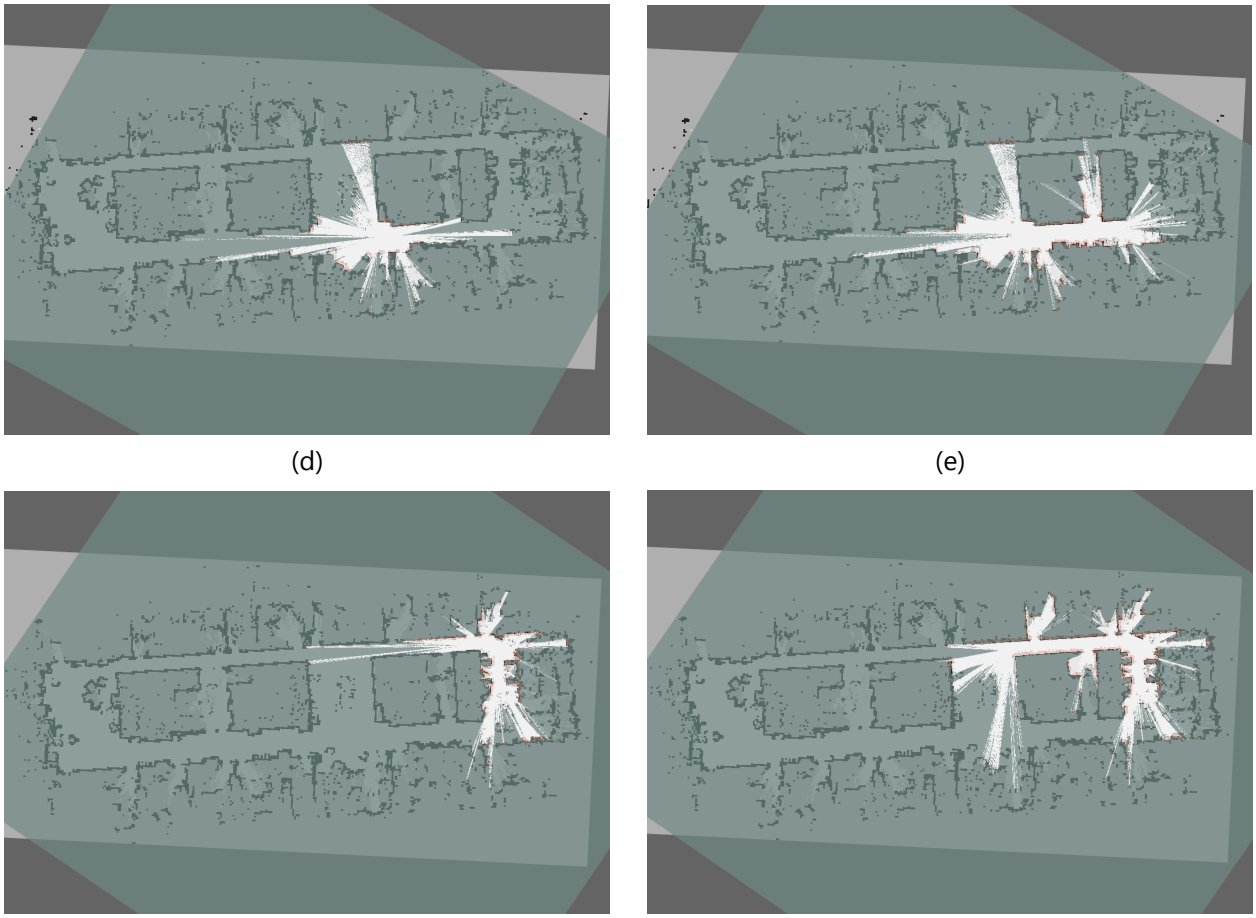
An example of map matching failure is shown in Figure 4.9. Several sections of the session map have been incorrectly aligned to the top left region of the global map (circled in red). Given that the map resolution is downsampled to 0.4m for map matching, and that only the occupied cells are used in the matching algorithm, the transform solution found in this case also results in feasible matches with other portions of the session map, circled in orange. Future work will look into further computational speedups that will allow running the map matching routine at higher map resolutions, which can mitigate these types of alignment errors.

Pepper online localisation in office space

Our map matcher package was also tested online on the Pepper robot while moving through an office space of approximately 56mx22m. Snapshots of the localisation results are shown in Figure 4.10, an example of initial localisation failure followed by recovery is shown in Figure 4.11.



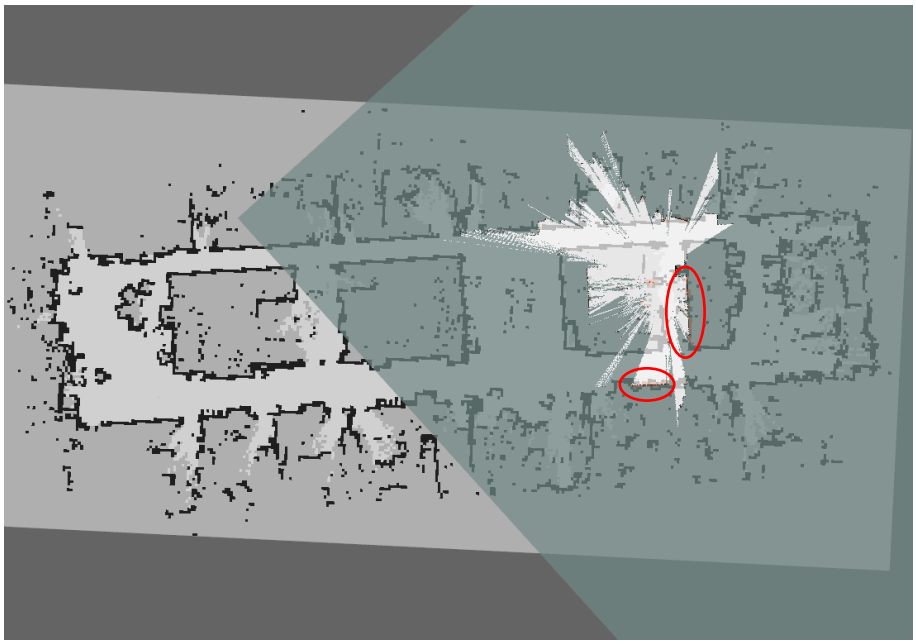
(a) (b) (c)



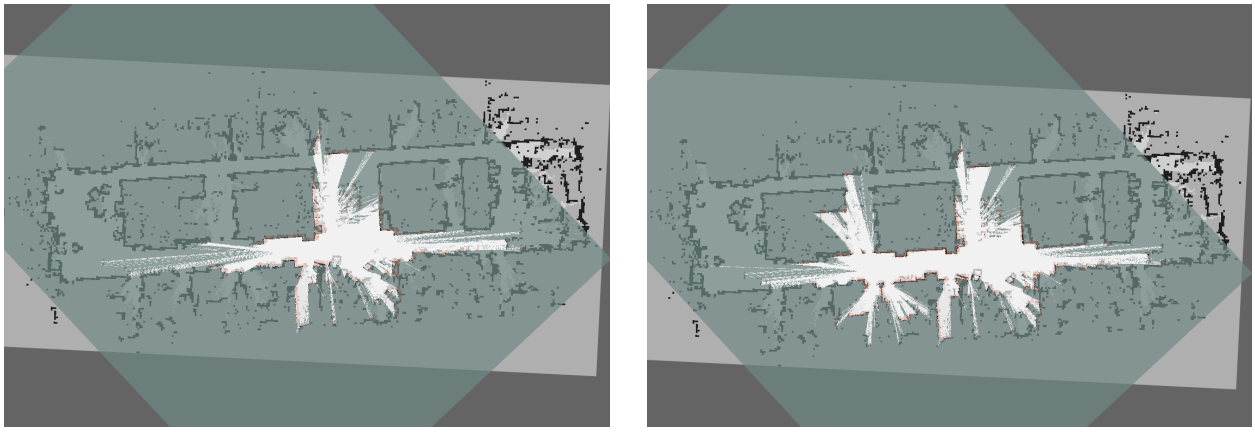
(d) (e)

(f) (g)

Figure 4.10: Online localisation performance of the map matcher routine on the CROWDBOT Pepper robot moving through an office space. Each row shows the map matching solution for consecutive online SLAM updates. Localisation performance was qualitatively tested for different starting locations in the office.



(a)



(b) (c)

Figure 4.11: An example of initial localisation failure (a) and recovery (b), (c). Several false obstacle matches between the session map and global map can be seen circled in red in (a), resulting in the localisation failure at the start. However, given the next map update, the system is able to correctly localise the robot (b) and maintain good localisation in following map updates (c).

These results demonstrate that not only is the map matcher localisation solution able to run in real time with on board compute, it is also robust to real-world noise from the LIDAR measurements. Furthermore, it is able to cope with noisy maps, such as the global map used in the Pepper localisation experiments, which contain various artefacts caused by glass doors and oblique reflections off metallic objects such as chair legs.

4.3 Outlook for CROWDBOT localisation

Our proposed map matcher algorithm produces robust prior-free localisation. However, the method still scales poorly with map resolution, in our tests, we used a map resolution of 0.4m to perform map matching, which is insufficiently refined for safe navigation, and as shown in Figures 4.9 and 4.11a, can occasionally still lead to incorrect matches. Thus, one option is to use the map matcher solution to find an initial pose to feed to a secondary localisation routine that can refine the robot's pose estimate. A simple implementation would be to pipe the output of the map matcher to AMCL and to maintain a watchdog service that monitors any divergence between the AMCL solution and the map matcher solution. In case of disagreement, we would revert to the map matcher solution, which is more robust, and re-initialise AMCL.

Further tests in simulation and on the Pepper robot will be conducted to evaluate the performance of the map matcher in highly dynamic and crowded environments. The purpose of these studies will be to quantify the algorithm's performance against increasing sensor noise and occlusions.

5. Local Motion Planning

The focus of the CROWDBOT project is to develop crowd-specific local motion strategies. However, to test these strategies on an autonomous mobile robot requires a global path to be provided. Thus, we first developed a global path planner based on Dijkstra's algorithm to provide the high-level plan given the available static environment map, which we describe in Section 5.1. In Section 5.2 we detail our proposed local motion planning strategy, which was designed with the goal of maximising the reactiveness of the robot to surrounding dynamic obstacles while maintaining progress along the designated global path. We conclude this section with an analysis of our local motion planner as executed on the Pepper robot.

5.1 Global motion planning

A versatile and proven approach for global planning in static environments is to use the Dijkstra algorithm to calculate the minimum distance-from-goal value at every point in the map. A common way of adapting the Dijkstra algorithm to data represented on 2D grids is to treat each grid cell as a graph node, with edges connecting cells to various nearby cells. Though cells can be arbitrarily connected in the graph, a useful and common method is to use a regular pattern of connectivity, for example one where each cell is connected to its four direct neighbours. This will be referred to as '4-connectivity' for the purpose of this discussion. Inclusion of more nearby cells to the neighbours set can be envisioned, resulting in 4, 8, 16, 32-connectivity and so on.

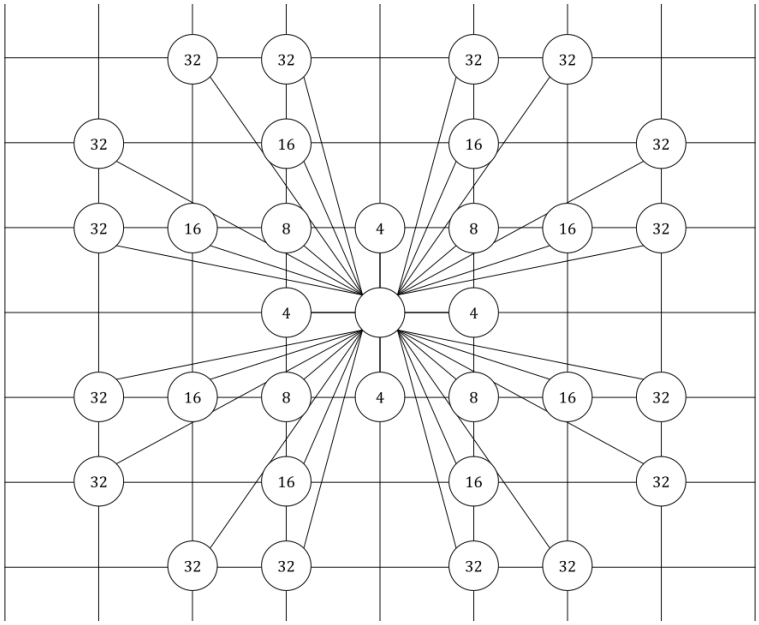


Figure 5.1: An example graph representation of a grid cell's neighbours in 4, 8, 16 and 32 connectivity. Note that the set of neighbours and edges in 8-connectivity is a superset of the set of neighbours and edges in 4-connectivity, the same is true for 16- vs. 8-connectivity, and so on.

Higher connectivities present attractive properties, as they increasingly approximate a 2-dimensional Euclidean distance transform which is blocked by obstacles. Such a field can be intuitively understood as representing a sampling of the minimum distance an agent would have to travel on a continuous 2D space if they could move in any direction. However, higher connectivities also introduce complexities, such as the possibly undesired possibility of 'jumping' over obstacles. This issue can be mitigated by performing local pruning of high-connectivity neighbours based on whether an obstacle lies between them. However, this would increase the computational complexity of the search. In addition, 16-and-higher-connectivities present the possibility of introducing local minima to the resulting field, since the propagation of minimum cost in space during the Dijkstra algorithm is no longer wavefront-like but can 'jump' over cells.

In order to obtain a global path from such a distance field, a simple algorithm can be used. At each node, the algorithm checks the distance field value for each neighbour and picks the neighbour with the smallest value. This guarantees that the path is the shortest possible path to the goal.

In some cases, several neighbours present the same distance value. This means that both paths are viable shortest paths. Expanding these choices for every alternative can yield a large number of equivalent shortest paths to goal. If only a single shortest path is desired, a proposed method is to stochastically pick which of the equivalent possibilities should be conserved. This method is used to obtain all path examples presented in this section, for ease of visualization.

Algorithm 2: Naïve algorithm for shortest path

1 Initialise $n = n(x_{robot})$
2 $Path = [\mathbf{x}(n)]$
3 **While** true:
4 $n_{best} = \arg\min_{n_i}\big(D(n_i)\big), \quad n_i \in Neighbours(n, connectivity)$
5 **If** $D(n_{best}) \geq D(n)$
6 Break // reached local minimum
7 **End If**
8 $Path = [Path, n_{best}]$
9 $n = n_{best}$

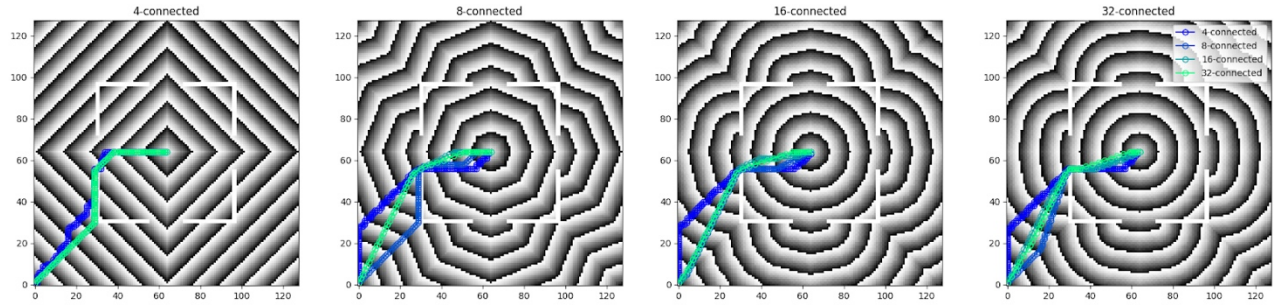


Figure 5.2: Comparison of contours for distance-to-goal fields calculated with the Dijkstra algorithm, using 4, 8, 16, and 32-connectedness on a 2D example map. The agent is located at (0,0) and the goal is located at (64,64). A set of four angled obstacles constrain the path to exit the centre of the map from one of four "doorways".

In practice we observe that higher connectivity causes the algorithm to yield paths which are found qualitatively to be smoother and more intuitive than lower-connectivity solutions.

Another result of testing the Dijkstra algorithm for global planning is that the shortest paths it yields pass close to static obstacles. This is to be expected as the algorithm presented so far does not consider distance to obstacles at all. One possible solution is to pass the path to a local planner, which then adds obstacle avoidance objectives, such as an elastic-band planner.

On the other hand, it is possible to include this objective in the distance field directly, by adding distance penalties in areas close to obstacles. This presents the advantage of incurring the additional computational cost only once instead of many times as in the case of a local planner, so long as the map remains static. A proposed implementation of this objective is to first calculate the Euclidean Signed Distance Field of every point to its closest obstacle in the map (an efficient algorithm for this is presented in [46]. Then, for every edge in the Dijsktra graph, add a penalty based on the ESDF, for example:

$$edge\_cost_{new}(n_i, n_j) = edge\_cost(n_i, n_j) + \frac{1}{ESDF\left(\frac{\mathbf{x}(n_i)+\mathbf{x}(n_j)}{2}\right)}, \quad (5.1)$$

where n_i and n_j are the two nodes connected by this edge.

The concept of adding extra edge costs in the Dijkstra algorithm to serve as area penalties could in theory be extended to other navigation objectives, such as dynamic obstacle avoidance. However, due to computational cost, it is useful to use this method for static elements of the scene only and add local planning components for dynamic elements. This makes it possible to compute the weighted distance field only once, assuming the map is known (in the case of SLAM, at a low-frequency update rate). Thus, we avoid recomputing the fields every time a dynamic agent's position changes.

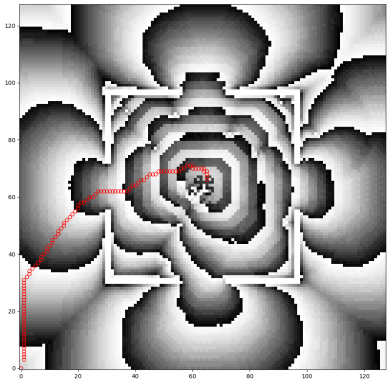


Figure 5.3: Contours of a 32-connected Dijkstra field, which includes area penalties both close to obstacles and to dynamic agents.

One concept which was not explored is the possibility of using Dijkstra distance fields for waypoints instead of a single (x,y) goal objective. Using parameters, this field could encode some objective function representing progress towards a goal while passing through waypoints.

5.2 Local, low-latency planning

Planning algorithms described in the previous section are dependent on the process of detecting and classifying objects and agents in the environment. This process, however, presents several challenges, mainly: i) it can induce additional latency ii) it presents additional failure modes, such as false/true positive/negatives, which then propagate through the subsequent planning steps, leading to unpredictable or undesired behaviours. This fact can serve to motivate the development of end-to-end planning techniques: motion planning methods in which the planner takes sensor readings as inputs directly, and outputs velocity commands. In order to research the viability and test the limits of such methods in the case of CROWDBOT, a low-latency planner was designed to function on the Pepper platform modified by ETHZ.

The low latency planner described in this section is similar to the Dynamic Window Approach, and Velocity Obstacle, in that it creates an estimate of reachable velocities at the current time, and searches in the sampled space of those velocities for those which maximize an objective function $J(\mathbf{v})$. The space of reachable velocities is defined as the intersection of two sets, the set of velocities which satisfy the platform velocity limits, V_s, and the set of velocities reachable from the velocity at current time according to the platform acceleration constraints, V_a,

$$V_s = \{\mathbf{v} | \mathbf{v} \in \mathbb{R}^2, \|\mathbf{v}\| < v_{max}\}, \quad (5.2)$$

$$V_a = \left\{\mathbf{v} \middle| \mathbf{v}, \mathbf{v}_t \in \mathbb{R}^2, \|\mathbf{v} - \mathbf{v}_t\| \frac{1}{dt} < a_{max}\right\}. \quad (5.3)$$

The objective function J at current state S is defined as,

$$J(\mathbf{v}, S), \ s.t. \ S = \{\boldsymbol{x}_{goal}^t, O^t\}, \quad (5.4)$$

with $\boldsymbol{x}_{goal}^{t}$ as the spatial coordinates of the goal in the robot frame at the current time, and O^t as the set of obstacle coordinates $\{\boldsymbol{x}_{obstacle_1}^{t}, \boldsymbol{x}_{obstacle_2}^{t}, \dots\}$. Furthermore, $J(\mathbf{v}, S)$ is,

$$\text{undefined} \quad \text{if} \quad d_{goal}^{t+1} > D \text{ and } p_g < 0, \tag{5.5}$$

$$p_g \quad \text{if} \quad \left(d_{goal}^{t+1} \leq D \text{ or } p_g \geq 0\right) \text{ and } d_{closest}^{t+1} \geq R \text{ and } d_{closest}^{t} > R, \tag{5.6}$$

$$0 \quad \text{if} \quad \left(d_{goal}^{t+1} \leq D \text{ or } p_g \geq 0\right) \text{ and } d_{closest}^{t+1} < R \text{ and } d_{closest}^{t} > R, \tag{5.7}$$

$$p_o + 0.1p_g \quad \text{if} \quad \left(d_{goal}^{t+1} \leq D \text{ or } p_g \geq 0\right) \text{ and } p_o > 0 \text{ and } d_{closest}^{t} \leq R, \tag{5.8}$$

$$0 \quad \text{if} \quad \left(d_{goal}^{t+1} \leq D \text{ or } p_g \geq 0\right) \textit{ and } p_o \leq 0 \textit{ and } d_{closest}^{t} \leq R, \tag{5.9}$$

where,

$$\mathbf{x}^{t+1} = \mathbf{v}\, dt, \tag{5.10}$$

$$p_g = \left\|\mathbf{x}_{goal}^{t}\right\| = \left\|\mathbf{x}^{t+1} - \mathbf{x}_{goal}\right\|, \tag{5.11}$$

can be understood as the progress towards the goal in meters.

$$d_{goal}^{t+1} = \left\|\mathbf{x}^{t+1} - \mathbf{x}_{goal}\right\|, \tag{5.12}$$

$$d_{closest}^{t} = \min_i\left(\left\|\mathbf{x}_{obstacle_i}^{t}\right\|\right) \text{ for } \mathbf{x}_{obstacle_i}^{t} \in O^t, \tag{5.13}$$

$$d_{closest}^{t+1} = \min_i\left(\left\|\mathbf{x}^{t+1} - \mathbf{x}_{obstacle_i}^{t+1}\right\|\right) \text{ for } \mathbf{x}_{obstacle_i}^{t+1} \in \hat{O}^{t+1}, \tag{5.14}$$

$$\hat{O}^{t+1} = \text{Predictor}(O^t), \tag{5.15}$$

and

$$p_o = d_{closest}^{t+1} - d_{closest}^{t}, \tag{5.16}$$

can be understood as the progress away from the closest obstacle in meters. R is a constant representing the comfort radius of the robot. D is a contant for the maximum distance the robot is willing to be away from its goal.

If no prediction of the future position of the obstacles $\hat{O}^{t+1}$ is available, one can make the assumption that obstacles can be considered as static over the planning timescale and replace $\hat{O}^{t+1}$ with $\hat{O}^{t}$ in the calculation of $d_{closest}^{t+1}$.

In words, the objective described above can be outlined as follows:

(1) If the robot is too far away from the goal and does not make progress towards it, J is undefined.

Otherwise:

If the robot is free from obstacles:

(2) If the resulting position is also free, J equals progress towards goal in meters.

(3) If the resulting position is not free, J equals 0.

If the robot is not free from obstacles:

(4) If the resulting position yields progress away from obstacles, J equals progress away from obstacles plus a small contribution of progress towards goal.

(5) If the resulting position yields no progress away from obstacles, J equals 0.

In the case of Pepper, the highest field-of-view and precision sensor for obtaining an estimate on the position of obstacles is LIDAR. For this reason, the planning algorithm is run with each new LIDAR sensor reading. The LIDAR scan is filtered to remove noise, and the set of (x, y) coordinates of LIDAR points is assigned directly to the set of obstacle coordinates O^t. $\mathbf{x}_{goal}$ is obtained from Pepper's state estimation.

The execution of the algorithm is described in Algorithm 3.

Algorithm 3: Local, low-latency planner

1 *At given time* t*, obstacle coordinates* O^t*, goal coordinates* $\mathbf{x}_{goal}^{t}$

$S = \{\mathbf{x}_{goal}^t, O^t\}$
$J_{best} = 0$
$\mathbf{v}_{best} = (0,0)$
ForEach $\mathbf{v} \in V_a \cap V_s$
  $J^{t+1} = J(\mathbf{v}, S)$
  If $J^{t+1} > J_{best}$
    $J_{best} = J^{t+1}$
    $\mathbf{v}_{best} = \mathbf{v}$
  End If
End ForEach

5.3 Outlook for CROWDBOT local motion planning

Tests on the platform result in responsive behaviour of the robot, though several issues are raised, leading to challenges for future work:

- The highly dynamic commands generated by the algorithm were observed to cause toppling of the robot, due to its high centre of gravity. This motivates a modification of the approach taking into account platform dynamics. A candidate for such an approach would be a Model Predictive Control formulation.
- When an obstacle is directly between the goal and robot, the above formulation does not lead to contouring around the obstacle. This is due to the goal progress metric being a naïve Euclidean distance to goal. A proposed solution would be to implement a specific sidestepping objective. On the other hand, estimates for the velocity of obstacles would be useful in determining good avoidance behaviour. Another possibility is the use of a more complex progress metric, such as one based on Dijkstra distance as specified in the global planning section.
- When combined with a global planner for path following, the above formulation can have issues entering narrow corridors. If the parameter R, the comfort radius of the local planner, is larger than half the corridor width, the planner will not promote trajectories passing through the corridor as it opposes the local obstacle avoidance objective. On the other hand, small values of R will cause the robot to get too close to dynamic obstacles. This is problematic as it causes failures in avoiding obstacles that are moving towards the robot. In the worst case, the distance between the robot and dynamic obstacles acts as a buffer which allows the robot to react before contact is made, despite unavoidable latencies in the hardware-software-hardware pipeline. Proposed solutions are to use a smooth function instead of a hard boundary for the radius or to vary the radius R and corresponding allowed velocities based on the obstacle velocities or scene density. This would require a velocity estimate for obstacles, which is not trivially measurable. It should also be noted that for a sufficiently complex velocity estimator, the end-to-end specification of this planner would no longer apply as additional latencies, sources of errors, and other downsides are to be expected.

5.4 Dynamical systems-based control for avoiding concave obstacles

When navigating in crowds, avoiding obstacles very rapidly using low-level command is crucial as these obstacles may appear rapidly and time to react is very limited. This work [47] extends the Modulated Dynamical System (MDS) approach to perform obstacle avoidance (reported in Period 1), see also [48], to handle general shapes of the obstacles, and describes a complete on-board implementation and a corresponding experimental demonstration.

5.4.1. Method

Given an occupancy grid representing the environment, each connected component gives rise to one (possibly concave) obstacle. The approach obtains a smooth boundary for each obstacle by fitting a Bézier spline to the occupied cells. The dynamical system which guides the robot around the obstacles to reach the attractor is computed in three steps for each obstacle (see Figure 5.4).

First, a coordinate transformation maps the position of the robot and the obstacle at hand into a space wherein the obstacle is a circle. There, the robot's angular position is such that it preserves across both spaces the arc length along the obstacle circumference between the projections of the robot and the attractor on the obstacle boundary. The robot's distance to the obstacle boundary in the new space is a monotonic function of the original distance.

Second, the MDS [48] computes the velocity command in the new space, and third, the command is transformed back into the actual space by multiplication with the Jacobian of the inverse transformation. The final velocity command results as the distance-weighted average of the velocity commands computed in this manner for each individual obstacle.

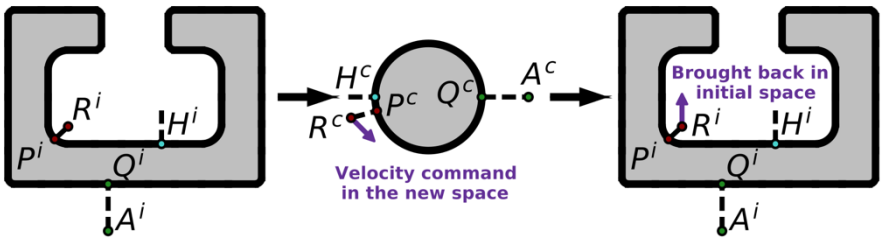


Figure 5.4: The dynamical system for obstacle avoidance is computed in three steps per obstacle.

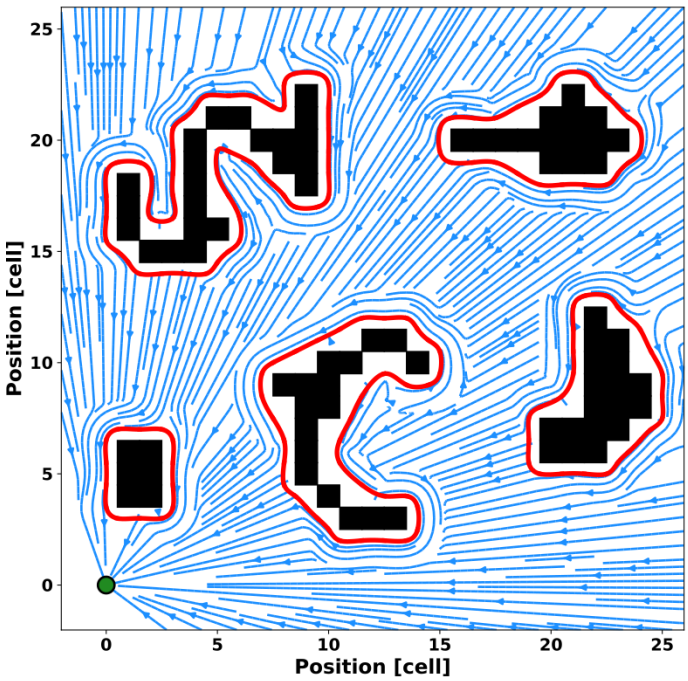


Figure 5.5: The example shows the velocity field for an arrangement of several convex obstacles with guaranteed convergence to the attractor.

To limit the influence of each obstacle to a small range, the weight of any contribution becomes zero if the distance to the respective obstacle is larger than that range. Convergence to the attractor is guaranteed for

any trajectory which does not enter a region which falls into the range of two or more obstacles. The velocity field for an exemplary obstacle arrangement is shown in Figure 5.5.

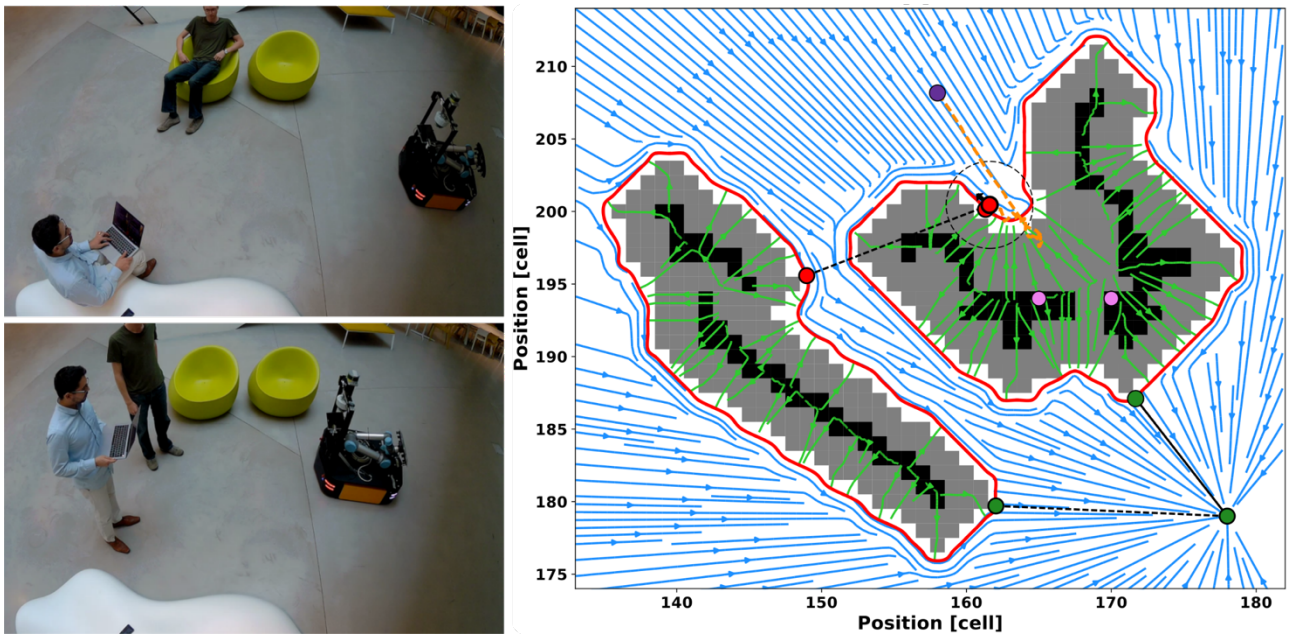


Figure 5.6: In this experiment, the robot adapts to dynamic obstacles, namely two pedestrians who block its path, and follows a new flow leading around the composite obstacle (on the right).

5.4.2. Experimental results

The performed experiments test the real-time avoidance of static and dynamic obstacles. Figure 5.6 shows how the platform adapts its path towards the attractor in response to two pedestrians. While the original flow leads between the obstacles (Figure 5.6, top left), after this gap closes (Figure 5.6, bottom left), the newly computed flow leads out of and around the concave obstacle formed by the two pedestrians and the static obstacles (Figure 5.6, right). The experiments demonstrate that the implementation can respond to and avoid such dynamic obstacles (in addition to static obstacles) in real-time and converge to the desired position.

References

[1] H. Carrillo, P. Dames, V. Kumar, and J. A. Castellanos, "Autonomous robotic exploration using a utility function based on Rényi's general theory of entropy," Autonomous Robots, vol. 42, no. 2, pp. 235–256, 2018.

[2] S.-Y. An, L.-K. Lee, and S.-Y. Oh, "Ceiling vision-based active SLAM framework for dynamic and wide-open environments," Autonomous Robots, vol. 40, no. 2, pp. 291–324, Feb 2016.

[3] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," IEEE Transactions on Robotics, vol. 32, no. 6, pp. 1309–1332, 2016.

[4] A. Pázman, Foundations of Optimum Experimental Design. Springer,1986, vol. 14.

[5] H. Carrillo, I. Reid, and J. A. Castellanos, "On the comparison of uncertainty criteria for active SLAM," in Robotics and Automation (ICRA), 2012 IEEE International Conference on. IEEE, 2012, pp. 2080–2087.

[6] D. J. MacKay and D. J. Mac Kay, Information theory, inference and learning algorithms. Cambridge university press, 2003.

[7] A. Rényi, "On measures of entropy and information," Hungarian Academy of Sciences, Budapest Hungary, Tech. Rep., 1961.

[8] C. Stachniss, G. Grisetti, and W. Burgard, "Information gain-based exploration using Rao-Blackwellized particle filters." in Robotics: Science and Systems, vol. 2, 2005, pp. 65–72.

[9] C. Leung, S. Huang, and G. Dissanayake, "Active SLAM using model predictive control and attractor based exploration," in Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on. IEEE, 2006, pp. 5026–5031.

[10] R. Valencia, J. V. Miró, G. Dissanayake, and J. Andrade-Cetto, "Active Pose SLAM," in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct 2012, pp. 1885–1891.

[11] J. Vallvé and J. Andrade-Cetto, "Active pose SLAM with RRT," in Robotics and Automation (ICRA), 2015 IEEE International Conference on. IEEE, 2015, pp. 2167–2173.

[12] B. Mu, M. Giamou, L. Paull, A.-a. Agha-mohammadi, J. Leonard, and J. How, "Information-based active SLAM via topological feature graphs," in Decision and Control (CDC), 2016 IEEE 55th Conference on. IEEE, 2016, pp. 5583– 5590.

[13] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, X. Zhao, and T.-K. Kim, "Multiple object tracking: A literature review," arXiv preprint arXiv:1409.7618, 2014.

[14] J. Burlet, T. D. Vu, and O. Aycard, "Grid-based localization and online mapping with moving object detection and tracking," Ph.D. dissertation, INRIA, 2006.

[15] S.-Y. An, L.-K. Lee, and S.-Y. Oh, "Ceiling vision-based active SLAM framework for dynamic and wide-open environments," Autonomous Robots, vol. 40, no. 2, pp. 291–324, Feb 2016.

[16] M. S. Bahraini, M. Bozorg, and A. B. Rad, "SLAM in dynamic environments via ML-RANSAC," Mechatronics, vol. 49, pp. 105–118, 2018.

[17] C. Bibby and I. Reid, "Simultaneous localisation and mapping in dynamic environments (SLAMIDE) with reversible data association," in Proceedings of Robotics: Science and Systems, 2007.

[18] A. Walcott-Bryant, M. Kaess, H. Johannsson, and J. J. Leonard, "Dynamic pose graph SLAM: Long-term mapping in low dynamic environments," in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2012, pp. 1871–1878.

[19] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental smoothing and mapping," IEEE Transactions on Robotics, vol. 24, no. 6, pp. 1365–1378, 2008.

[20] C. Mertz, L. E. Navarro-Serment, R. MacLachlan, P. Rybski, A. Steinfeld, A. Suppé, C. Urmson, N. Vandapel, M. Hebert, C. Thorpe, "Moving object detection with laser scanners," Journal of Field Robotics, vol. 30, no. 1, pp. 17–43, 2013.

[21] R. MacLachlan and C. Mertz, "Tracking of moving objects from a moving vehicle using a scanning laser rangefinder," in 2006 IEEE Intelligent Transportation Systems Conference. IEEE, 2006, pp. 301–306.

[22] L. E. Navarro-Serment, C. Mertz, N. Vandapel, and M. Hebert, "LADAR-based pedestrian detection and tracking," 2008.

[23] J. Yin, L. Carlone, S. Rosa, M. L. Anjum, and B. Bona, "Scan matching for graph SLAM in indoor dynamic scenarios," in The Twenty-Seventh International Flairs Conference, 2014.

[24] L. Beyer, A. Hermans, and B. Leibe, "DROW: Real-time deep learning-based wheelchair detection in 2-D range data," IEEE Robotics and Automation Letters, vol. 2, no. 2, pp. 585–592, 2017.

[25] L. Beyer, A. Hermans, T. Linder, K. O. Arras, and B. Leibe, "Deep person detection in two-dimensional range data," IEEE Robotics and Automation Letters, vol. 3, no. 3, pp.2726-2733, 2018.

[26] G. Grisettiyz, C. Stachniss, and W. Burgard, "Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling," in Proceedings of the 2005 IEEE International Conference on Robotics and Automation. IEEE, 2005, pp. 2432–2437.

[27] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," IEEE Transactions on Robotics, vol. 23, no. 1, pp. 34–46, 2007.

[28] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147–1163, 2015.

[29] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255–1262, 2017.

[30] A. Censi, "An ICP variant using a point-to-line metric," in Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on. IEEE, 2008, pp. 19–25.

[31] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP variants on real-world data sets," Autonomous Robots, vol. 34, no. 3, pp. 133–148, 2013.

[32] A. Censi, "An accurate closed-form estimate of ICP's covariance," in Robotics and Automation, 2007 IEEE International Conference on. IEEE, 2007, pp. 3167–3172.

[33] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," The International Journal of Robotics Research, vol. 31, no. 2, pp. 216–235, 2012.

[34] F. Dellaert and M. Kaess, "Square Root SAM: Simultaneous localization and mapping via square root information smoothing," The International Journal of Robotics Research, vol. 25, no. 12, pp. 1181–1203, 2006.

[35] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," Georgia Institute of Technology, Tech. Rep., 2012.

[36] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics. MIT press, 2005.

[37] B. Yamauchi, "A frontier-based approach for autonomous exploration," in Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. IEEE, 1997, pp. 146–151.

[40] G. Jumarie, Relative Information: Theories and Applications. Berlin, Heidelberg: Springer-Verlag, 1990.

[41] J. E. Bresenham, "Algorithm for computer control of a digital plotter," IBM Systems journal, vol. 4, no. 1, pp. 25–30, 1965.

[42] M. Kaess and F. Dellaert, "Covariance recovery from a square root information matrix for data association," Robotics and Autonomous Systems, vol. 57, no. 12, pp. 1198–1210, 2009.

[43] R. E. Kalman, "A new approach to linear filtering and prediction problems," Journal of Basic Engineering, vol. 82, no. 1, pp. 35–45, 1960.

[44] M. P. Chandra et al., "On the generalised distance in statistics," in Proceedings of the National Institute of Sciences of India, vol. 2, no. 1, 1936, pp. 49–55.

[45] W. Hess, D. Kohler, H. Rapp and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in Proceedings of the 2016 IEEE International Conference on Robotics and Automation, pp. 1271–1278, 2016.

[46] Felzenszwalb, P.F. and Huttenlocher, D.P., 2012. Distance transforms of sampled functions. Theory of computing, 8(1), pp.415-428.

[47] P.-A. Léziart, D. Gonon, D. Paez-Granados, and A. Billard, "Dynamical Systems-based Control for Avoiding Concave Obstacles," in 2020 IEEE International Conference on Robotics and Automation, 2019. Under review.

[48] L. Huber, A. Billard, and J.-J. Slotine, "Avoidance of convex and concave obstacles with convergence ensured through contraction," IEEE Robotics and Automation Letters, vol. 4, no. 2, pp. 1462–1469, 2019.