International Master's Thesis

3D Scan-based Navigation using Multi-Level Surface Maps

Andreas Persson
Technology

Studies from the Department of Technology at Örebro University
Örebro 2009

3D Scan-based Navigation using Multi-Level Surface Maps

Studies from the Department of Technology at Örebro University

Andreas Persson

3D Scan-based Navigation using Multi-Level Surface Maps

Supervisors: **Achim J. Lilienthal**
Todor Stoyanov

Title: 3D Scan-based Navigation using Multi-Level Surface Maps

ISSN 1404-7225

Abstract

The field of research connected to mobile robot navigation is much broader than the scope of this thesis. Hence in this report, the navigation topic is narrowed down to primarily concerning mapping and scan matching techniques that were used to achieve the overall task of navigation nature. Where the work presented within this report is based on an existing robot platform with technique for providing 3D point-clouds, as result of 3D scanning, and functionality for planning for and following a path.

In this thesis it is presented how a scan matching algorithm is used for securing the alignment between provided succession point-clouds. Since the computational time of nearest neighbour search is a commonly discussed aspect of scan matching, suggestions about techniques for decreasing the computational time are also presented within this report. With secured alignment, the challenge was within representing provided point-clouds by a map model. Provided point-clouds were of 3D character, thus a mapping technique is presented that provides rough 3D representations of the environment. A problem that arose with a 3D map representation was that given functionality for path planning required a 2D representation. This is addressed by translating the 3D map at a specific height level into a 2D map usable for path planning, where this report suggest a novel traversability analysis approach with the use of a tree structure.

Acknowledgements

I would first of all like to thank my supervisor Achim J. Lilienthal for the opportunity to work with this topic during the end of my education. Secondly, I would like to thank my co-supervisor Todor Stoyanov for his patience, the good contact and all his help during this project. Next, I would like to thank Amy Loutfi for her assistance during the concluding of this thesis. Also, I would like to thank all people at AASS that I have been in contact with throughout my education - keep up the good work. Especially thanks to Per Sporrong and Bo-Lennart Silfverdal for the work on getting "Alfred" back on track.

Last but not least, I would like to thank all my friends and my family - I hope to see more of you all after the concluding of this thesis. With extra thanks to my partner Sarah Nilsson for her support and understanding.

Contents

Chapter 1
Introduction

In a similar way as humans are using their senses to get an understanding of their environment, mobile robots are using input sensors to *sense* their surrounding environment. In contrast to humans who automatically interpret three-dimensional (3D) information, mobile robots have traditionally been using 2D information. However, research within mobile robots has lately put more and more interest in obtaining 3D information in order to achieve a greater understanding of their environment.

The human brain *memorizes* information about the environment, such information can be recovered and employed during navigation. The same ability to memorize and navigate according to an *internal map* of the environment, is desirable even for mobile robots. However, questions that arise with the idea of a map, particularly when 3D information about the environment is presented, are how to store the information and how much information could be stored. Even in cases where a *map* of the environment is provided, there still could be faulty information within the map. This is because senses though sophisticated could provide faulty information. Also, similarities between different conditions could all cause faulty information to be registered. Hence it could be imprudent to *trust* too much in the provided information. An example of this is Christopher Columbus' discovery of America, where the information about a direct seaway to India across the Atlantic Ocean was truly trusted. By default the human brain attempts to correct and compensate for faulty information along the travelled way when more information is provided. This by default is not the case for mobile robots. Thus research tries to come up with similar solutions, where the map is kept consistent and the information within the map could be trusted for a navigation task.

1.1 Motivation

The world within context of robot navigation has traditionally assumed to be flat, where 2D information is enough for building a map of the environment

[8]. However, the real world is not flat. Thus if a mobile robot should be able to build representing maps of rough, uneven and unknown real world environments, 3D information is required. Consequently, the 3D information must be processed and used in an appropriate way so that a safe and efficient navigation could be guaranteed. Most natural the 3D information would be processed and stored within a corresponding 3D map representation, however a potential problem is that the 3D maps could easily grow out of proportions [12]. Hence, other less complex and memory efficient ways of representing rich 3D information within a map could be of significance.

The reliability within the map representation must also be guaranteed. The navigation is based on the map representation, and the map is built based on information about the environment. The cause of problems within the navigation can thus be traced back directly to the obtained information. Therefore, it is of importance that faults within the information are recognized and compensated for; otherwise a non-consistent reliable map representation of the environment cannot be maintained.

1.2 Objectives

The overall objective of this project was to set up a fully functional mobile robot application for a navigation task, where the application should be able to handle both indoor and outdoor navigation. Already available was a robotic platform with implemented methods for obtaining 3D input sensor data, odometry data, and for planning and executing movements based on a given map model of the surrounding environment. However, still missing was the middleware functionality required for carrying out a complete navigation task. In other words, a way of translating the input data into a map model that is usable for path planning.

Thus, the specific objectives for this project were (1) interpret given 3D data and implement the methods and techniques required for building a 3D map model of the surrounding environment and (2) implement methods and techniques for securing the reliability both in the map model and the robots own estimation of its position with respect to the map model.

Furthermore, the movement commands of the available platform were purely dependent on a human operator and no intelligent methods for collecting data autonomously were available. Nor was a suitable way of presenting the resulting map of the environment available. Therefore human-robot interaction became another central aspect of this work, where a graphical point-and-click interface where users would specify the goal location of the robot and navigation to that point would be handled automatically was to be implemented. Finally, the above methods and techniques were to be implemented as a self-standing *Player* module [1].

1.3 Contribution

The contribution of this work was done through the implementation of techniques for building a map representation and securing the reliability with the same representation. Suggestions of improvements have also been done in order to achieve integration, where the used techniques though can be found within literature has not been tested on the specific robotic platform available.

The contribution can be divided by following:

- Implementation of the *multi-level surface* (MLS) map concept together with a new approach of using a 2D tree structure to represent the map model.
- Implementation of the iterative closest point (ICP) algorithm together with a new suggested approach of using the depth of a 3D tree structure to improve the nearest neighbour search of the algorithm.
- Implementation of a graphical human-robot interface, where the graphical part of implementation was done with use of the *OpenGL* framework.
- Experimental results where the map model together with the ICP algorithms performance on this system is shown.

Even though the overall goal was to set up a fully functional application for a navigation task, the total navigation quality was not evaluated within the scope of this work. The techniques and algorithms implemented was instead the area of focus for evaluation, which opens for future work where the total navigation could better be evaluated.

1.4 Outline

- **Chapter 2** - introduces the methods and techniques used for a navigation task also discusses the overall navigation topic and describes the differences between navigation tasks in indoor and outdoor environments.
- **Chapter 3** - introduces the overall system used for this project. This chapter gives an introduction to the 3D scanner used, and shows how Player software was used together with this project. This chapter also covers design decisions and implementation details about the graphical user interface that was implemented as part of this project.
- **Chapter 4** - gives a description of the scan matching method that has been used to maintain the map model for this project and shows how a consistent map model could improve the localization of the robot as well.

- **Chapter 5** - gives a detailed overview of the multi-level surface concept and shows how the built map model was translated into occupancy maps for the path planner, and how the map model was built with use of a tree structure.
- **Chapter 6** - presents both an indoor and an outdoor experiment that was done to test the application.
- **Chapter 7** - summaries the work of this project and gives directions about further improvements and discusses the impact of the contributions of this thesis.

Chapter 2
Background and Related work

The focus throughout this chapter will be on the techniques both used (through existing implementations) and implemented within the scope of this project. Also comprehensive similar or alternative approaches within each field will be covered.

Techniques within the area of mobile robot research, which by implementation *can* simplify the overall navigation task, and that have been used for this project are:

- *Input sensors* - the very first step towards getting an understanding of the environment is to get input sensor data about the environment.
- *Scan matching* - register a new sensor scan with previous sensor scan(s). Thus attempt to find the best geometrical alignment between two sensor scans, and by so bring one scan into *correct* alignment with the other scan.
- *Mapping* - to build a map model that represent the robot's surrounding environment, where a map model could be represented in either a *geometrical* or a *topological* way.
- *Localization* - keep track of the robots position within the environment, where the location could be obtained either based on landmarks within the environment or as a result of scan matching.
- *Path planning and following* - based on the map model, plan and execute a sequence of movements actions so that the overall goal is achieved. In the context of robot navigation, the overall goal is usually about finding and following the *best* path between the robots start position S and a given goal position G within the environment.

Differences between outdoor and indoor navigation together with related works of inspiration is given in Section 2.1. An introduction to input sensors

that could provide range data is given in Section 2.2, while an introduction to scan matching together with different algorithms is given in Section 2.3. The background of the mapping and localization concept together with an alternative approach to scan matching is given in Section 2.4. An introduction to path planning is given in Section 2.5, before this chapter is concluded with a discussion in Section 2.6.

2.1 Related work

The aim of research within mobile robots is frequently to achieve autonomous navigation, and therefore it would be impossible to cover all related works within the domain. Thus this section will just point out a few works that has inspired this project together with some general aspects about robot navigation that is of interest for this project.

It is sufficient with 3DoF (degrees of freedom) for indoor navigation - the height is of no importance in an assumed flat environment. Thus it is only the position in x and y, and the heading angle θ that is of interest. Though the difference in height is neglected, the height of objects in the environment could be of interest, as described [29], where 3D scans were used to detect clutter objects. However, an assigned indoor experimental environment could easily be controlled; hence clutter objects could be removed. In controlled environments, other aspects that could affect the overall navigation could also better be controlled, e.g. the light conditions. While controlled environments still provides for good test and experimental areas, uncontrolled environments are the main area of interest of today for research within indoor navigation.

The main difference between indoor an outdoor navigation is that outdoor surfaces are not evenly flat - the height is also of interest. Thus the degrees of freedom increase to 6DoF - the position in x, y and z, together with the rotations angles around respectively axis roll α, pitch β and yaw γ. Therefore, methods must be found for both interpreting 3D information and building representing map models that could handle height differences. Other notable differences between indoor and outdoor navigation is that it is much harder to control an outdoor environment and the friction against an outdoor surface is also affected by the weather, season of the year, and terrain. Another aspect of interest for research is the possibility of build a map model that combines indoor and outdoor environments. Well tested algorithm for indoor navigation will normally get problems in an outdoor environment since the structure of the environment is different, and vice verse.

With 3D mapping in mind the main source of inspiration for this project was found within the papers about *multi-level surface* maps [28, 21, 22]. Where a new mapping method is presented together with the use of both scan matching [28, 21] and probabilistic localization [22]. In [22] Pfaff is also addressing the possibility of using multi-level surface maps for combined indoor and outdoor environments. Another work of inspiration is the one presented by Nüchter in

[19], where 3D scan matching was used for localization and to maintaining the map model consistent, thus the map was usable for autonomous planning of the next step of navigation.

2.2 Input range sensors

Input sensors must be used in order to get an understanding of the surrounding environment. In case of a mapping problems, which was the case for this project, sensors that provide range data are also to be favoured.

Range data could be provided by sonar-, laser-, or infrared range scanners or by stereo vision cameras. The type of sensor to choose depends very much on the environment in which the navigation should occur, e.g. using stereo vision would not be advisable for a navigation task within a dark underground environment.

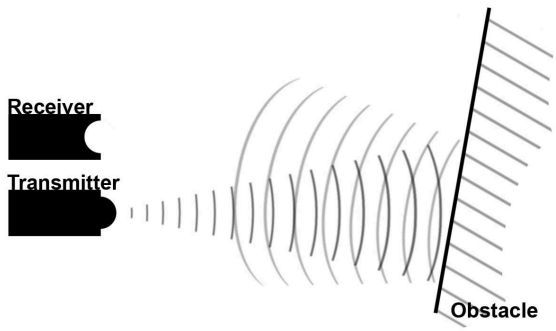

Figure 2.1: The technique behind active range scanners - a beam is sent out be a transmitter and a receiver is catching the reflecting beam.

Sonar-, laser-, or infrared range scanners work by a transmitter which sends out a beam and a receiver which is used to catch the reflecting beam as illustrated in Figure 2.1. Where the distance is given by the phase shift or traveled time. Sensors that both sends out and receives the same energy beam is referred to as *active* sensors, while sensors that directly catch energy from the surroundings, e.g. cameras, are referred to as *passive* sensors [14]. To acquire range information from camera images (at least) two cameras must be used in parallel. From the camera images the same points of interest are localized and the wanted distance is measured between the points of interest and used reference points, where the distance is given by *triangulation*.

In the early days of robotic mapping and navigation range data was mainly acquired through sonar sensors [8], but the technical development has resulted in that laser scanners and stereo vision are the most commonly used at present

day. Even though sonar range scanners are not frequently used today for navigation on solid ground, they still suit very well for underwater navigation where the sonar sound beams could travel further [9].

2.2.1 3D range scanners

The majority of range scanners *sense* in the same way as the human (and most of the mammals) eyes - the scans are taken in one direction and the amount of the environment that could be seen is limited by the field of view (usually merely a 2D *slice*). Therefore, scanners that could capture up to 360° of the environment, in a similar way as the chameleon eyes, are of interest when working with mobile robots. This was the case even for this project.

Figure 2.2: Sick LMS 200 laser scanner.

Even though *off-the-shelf* 3D scanners are comprehensive they are also very expensive. As a result, 2D range scanners are still most commonly used, e.g. the Sick LMS laser scanner seen in Figure 2.2. 2D range scanners have extensively been used for indoor navigation where it is used for obtaining scans parallel with the floor. However, 3D scans could be obtained by attaching a 2D scanner on top of a fully 360° rotating actuator, as demonstrated by Wulf in [29]. Another approach towards 3D scanning is to attach a 2D scanner on a pan/tilt unit, as used by Nüchter in [20], where the scan is taken in an oscillating pattern. 3D scans could also be obtained with the use of multiple 2D scanners, as demonstrated by Früh in [11], with a costal drawback proportional to the number of scanners used. In a similar way, a 3D scan could be acquired with stereo-vision if a camera configuration is used so that the entire 360° field of view is covered.

The desired result of 3D scanning is *point-clouds* as illustrated in Figure 2.3, where each point within the cloud corresponds to the exact position where a beam was hitting a surface or where a point of interest was found within the camera images.

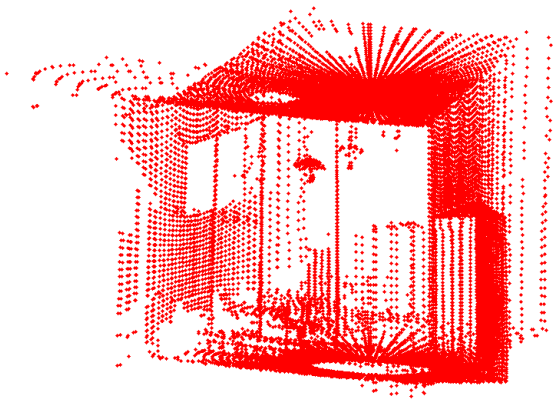

Figure 2.3: A point-cloud obtained as result of 3D scanning.

2.3 Scan matching algorithms

This section will give an introduction to scan matching algorithms that can be found within literature, thus not all of the algorithms presented within this section have been used for this project.

In the input sensor data there is always some kind of noise present. Noise is caused by growing uncertainty with the distance of the scan, bad calibration, moving objects in the surrounding, etc. The same goes for the robot's odometry, which commonly is distorted by slippage. The position and orientation of an obtained scan could therefore never be confident. To handle this kind of problem a *scan matching* (registration) method is used. A scan matching method, used within the context of robot navigation, attempts to match a new sensor scan with previous sensor scan(s). The transformed resulting scan is said to be in registration with previous scan(s).

If a scan matching shall be possible some initial criteria's must also be fulfilled (1) a rough estimation of the initial position must be given (commonly by the robot's odometry), and (2) the scans must be partly overlapped. The coordinates and orientation of the first scan are in general also locked down as the origin.

2.3.1 ICP

The state of art method for scan matching is the *iterative closest point* (ICP) algorithm, [2]. Which also was the algorithm used for this project. The aim of the algorithm is to iteratively find the best geometric alignment between two partially overlapped and rigid point-sets (point-clouds) given by 3D range scanners, where a new scan is referred to as the *data-set*, which is matched

against the previous scan(s) referred to as the *model-set*. By finding the best geometrical alignment between the data-set and the model-set, the noise in both the sensors and odometry could be minimized, and hence a consistent map model could also be maintained. Originally the algorithm was used directly on points (or meshes), but ICP works equally well on features extracted from a map model, as long as the same 3D structure are preserved within the features.

Even though the ICP algorithm is stable and provides good results, it has a bottleneck in form of a tedious pair searching step (nearest neighbour search) - where closest point-pairs must be found in between the both point-sets, for *each* iteration. Nearest neighbours are found by Euclidian distance, and where a naive search must consider all points in both sets; with m points in the model-set and n points in the data-set, resulting in $(O(mn))$ operations. A *kd-tree* structure are therefore commonly used for improving the nearest neighbour search, as described by Nüchter in [19]. The points in the model-set are thus stored within this tree structure of *k-dimension*, and where the nearest neighbour is found by searching further down the tree until a match is found on the leaf level (this is further addressed in Chapter 4.1.1).

Even though a tree structure is used, the time for convergence is still proportional to the number of points within the used point-sets. A brute solution to this is to down-sample the number of points within the point-set. However, as always with down-sampling, this could result in loss of valuable information. When searching for nearest neighbours one must also only considering overlapping points, as illustrated in Figure 2.4, otherwise non-overlapping points could *drag* the bias of the both point-sets further away from each other. The criterion to fulfil to count as overlapping points is never found with ease, because a large distance could be as result of a bad initial alignment. A solution to this is to assign a weight value to each point-pair, where pairs with large distances are lower weighted than pairs with small distances.

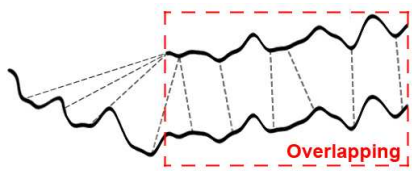

Figure 2.4: Only overlapping points (red/light gray rectangle) should be considered during the nearest neighbour search. Otherwise non-overlapping points (outside the rectangle) could drag the both point-sets further away from each other.

Both the nearest neighbour search and overall reduction of convergence time has by itself becoming areas of research, as discussed by Rusinkiewicz in [23], and many solutions of improvement has been suggested. A couple of those

solutions together with an entire different scan matching approach are briefly presented throughout the rest of this section.

The ICP algorithm will be further described in Chapter 4.

2.3.2 TrICP

One suggested approach for improving the nearest neighbour search is the *trimmed ICP* (TrICP) algorithm, which was presented by Chetverikov in [3]. The algorithm suggested that the point pairs should be sorted in descending order according to the Euclidian pair-distance, and that a fixed number T of the total pairs with large distance in between should be rejected. Hence N_T pairs with low pair-distance should be used, rather than the original N pairs, such that $N_T < N$.

An approach related to the TrICP algorithm is presented in Chapter 4.1.2.

2.3.3 IDC

Another suggested improvement is the *iterative dual correspondences* (IDC) algorithm. The algorithm was presented by Lu in [17], in which a method was suggested for speeding up the convergence time of the original algorithm. For each iteration the ICP algorithm is calculating the translation t and the rotation R by which the data-set is to be transformed. In the original algorithm with pair matching done by Euclidian distance is the translation converging quickly while the rotation is fine-tuned during remaining iterations. The IDC algorithm suggests that dual pair matching should be used, where a second nearest neighbour search should be done based on the polar coordinates of the points - searching for neighbouring point within a fixed angular interval. The result of matching with nearest point-pairs based on the polar coordinates are opposite the original algorithm, a quickly converging rotation.

An overall improvement in convergence time is achieved by transforming the data-set by applying the translation t_1 obtained by the original matching, and the rotation R_2 obtained by the latter matching.

2.3.4 NDT

The *normal distributions transform* (NDT) algorithm is in its original form meant for 2D problems, but a 3D solution is presented by Magnusson in his thesis [18]. The method suggests that rather than using the points directly, the point-clouds should be represented by normal distributions. Hence, the probability of finding a point at a certain position can be used and well tested standardized numerical methods, e.g. Newton's method, could be applied on the scan matching problem.

The major benefit with the algorithm is that no nearest neighbour search is required since the normal distributions are computed during a single pass of the

model-set. Removal of the otherwise computational costly nearest neighbour search step decreases undeniably the computational time of the scan matching. However, removal of the nearest neighbour search is not always desirable since the cause of the search could also be used for pre-processing purposes (as described in Chapter 4.1.2).

2.4 Mapping and Localization techniques

Mapping techniques are commonly assuming a known position within the environment. Though, the position given by the robot's odometry could never be fully trusted because of noise (e.g. slippage), is a localization method needed for finding the true position. Localization together with mapping is a classical *chicken and egg problem* - if a concitent map is provided the position could be found with ease, whilst a known position would simplify the mapping.

This section will discuss different techniques and methods for both mapping and localization. In following sub-sections, each problem will be addressed and where the structures of the sections are divided so that the techniques used for this project will be presented prior alternative techniques.

2.4.1 Mapping with range data

The mapping concept has traditionally been divided into two directions; *geometrical maps* and *topological maps*. An example that illustrate the difference between geometrical and topological maps are shown in Figure 2.5, where both approaches of representing the same geographical area are presented. This work is entirely focusing on geometrical maps, thus topological maps are presented just to point out the alternatives.

Geometrical representation

In the dawn of mobile robots the navigation was based on given maps, e.g. a blueprint, and where geometric features were extracted and used as abstract templates to be matched against sensor readings. Such approach provides only for short term decisions of the navigation and was unable to handle changes in the environment. A big turning point for robot navigation came with Elfes presentation of *occupancy maps*, [8]. This approach suggested that a representing map model should be built in a 2D grid pattern and maintained based on the sensor data. Where each cell within the map model should reflect the traversability of the corresponding geometrical area of the environment, and where the state of a cell could either be *drivable*, *unknown* or *occupied*.

When 3D sensors became more frequently used did this result in better accuracy within occupancy maps, since object could better be represented, as described by Wulf in [29]. Another result of using 3D sensors is the concept of

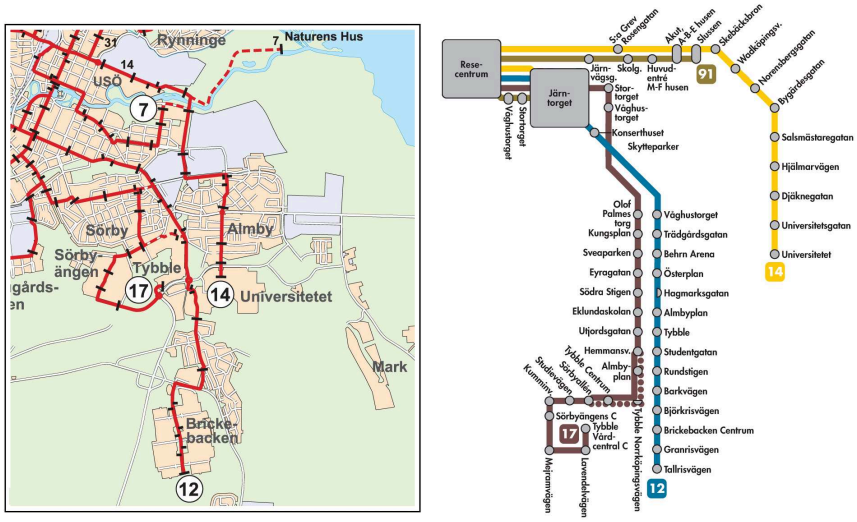

Figure 2.5: Different types of map representations of bus lines in Örebro city. *Right* - geometrical representation, *left* - topological representation. (Images from http://www.lanstrafiken.se - visited in September 2009).

$2\frac{1}{2}$D maps or *elevation map*, where the height information about the environment is stored as a surface within each cell instead of the traversability. On the other hand, it could be difficult to fairly represent a geometrical area with the use of only one surface.

When outdoor navigation has become more in focus, this has consequently also increased the demand of getting full representations of the 3D information within a map model. This has by itself become a popular research area [13], and where the most popular approaches are to represent the map by the raw data points or triangular meshes. A full 3D representation is detailed and can thus easily be textured, but comes with a disadvantage in shape of high memory requirement - which grows linear with the number of points added to the model, and which could grow out of proportions even for smaller indoor environments as described by Hähnel in [12].

As an alternative to both elevation maps and 3D maps, *multi-level surface* (MLS) maps was introduced a couple of years ago [28, 21, 22]. The main idea behind the technique is to represent more than one surface in each grid of an elevation map. Hence roughly represent a 3D environment without the complexity of full 3D maps. With MLS maps underpasses and holes in the ground could better be represented, which could be difficult to represent within normal elevation maps. Another benefit of MLS maps are the possibility of planning

for a path with different height levels, e.g. it is possible to plan a path both over and under a tunnel.

Mapping with geometrical maps will be further addressed in Chapter 5.

Topological representation

Topological maps are in difference from geometrical, not a geometric representation of the environment, but rather a representation *about* the environment. A topological map consists of a list of information about the connectivity between different places of interest within the environment. This will result in a graph representation of the environment. Hence, topological maps are not, in the same way as geometrical maps, limited by the dimension or the storage capacity. As such, topological maps could be very computational cheap, as demonstrated by Duckett in [5], where a new approach was presented which guaranteed to find a global optimal and consistent map.

2.4.2 Localization in relation to mapping

Research related to localization problems has been taking two directions; *scan matching-based* and *landmark-based* localization. Since scan matching was used for this project, the choice of localization approach was likewise given. Thus landmark-based localization is presented to aware the reader of the differences and to give background for later discussion.

Localization may appear trivial with the *GPS* (global positioning system) technology of today. However, the technique is not truly trustful because of areas that are shielded from receiving the GPS signals (could not see all satellites required for estimating the position), e.g. indoor, underwater and underground environments. Another aspect that has been pointing against the use of GPS is that only the global absolute position is provided, while no information about the heading of the robot (or even the rotation around all three axis) is provided. Also, the accuracy of a GPS is within $1 - 3$m, which is too much uncertainty for many applications. Therefore are other types of input sensors that provides more information about the environment to be favoured, together with other types of methods for keeping track of the location.

Scan match-based localization

The location of a mobile robot could be maintained consistent as a side result of scan matching (registration) of sensor data. The initial alignment for the matching algorithm is given by the robots estimated position. Since the aim of the algorithm is to find the best alignment in between sensor data (obtained at estimated position), the same resulting transformation could be applied and thus keep even the location consistent.

Localization together with scan matching will be further addressed in Chapter 4.2.

Landmark-based localization

Methods for landmark-based localization require that distinct landmarks are found within the environment. Such landmarks (electronic beacons) must be extracted from the input sensor data. In the early days of robotic navigation no effective method was present for this kind of problem, and the localization required that given landmarks such as bar-code reflectors, ultrasonic beacons, light beacons, visual patterns, etc., was used within a very controlled environment. Consequently, research has been done in attempts to present alternative solutions to controlled landmarks. Such as *scale-invariant image features* (SIFT) [24], which is a method for directly extract distinct scale invariant landmarks from stereo vision images. Another method, based on point data has been described by Wulf in [30], where 2D landmarks was generated from the points classified as flat surfaces within a 3D map model. Research has also been done in attempts to find solutions where distinct landmarks with ease could directly be found within the environment, such as the solution presented by Launay in [16], where landmarks were extracted by letting a camera pointing upwards to detecting the position and the orientation of the fluorescent tubes in the ceiling. Regardless used method, for any approach with landmarks are the following assumption the same; if distinct landmarks are known, but the robots position is unknown, it is possible to estimate the robots position with respect to the landmarks.

For a complete survey of mapping together with landmark-based localization methods see [27].

2.5 Path planning and following

After when a map model is provided, movements within the environment could be planned for based on the provided model. In the context of mobile robots, this is referred to as *path planning and following*, and which includes:

- *Path planning* - based on the given map model, plan a safe path from the start position S (the robot current position) to a given goal position G within the environment.
- *Path following* - traverse the planned path by carrying out movement actions.
- *Obstacle avoidance* - avoiding obstacles and keep a safety distance along the traversed path.

Along the traversed path is more information about the environment provided - which must be reflected within the map model. Also, the uncertainty

within the sensor data is growing with the scanned distance thus is also the uncertainty within a path growing with the distance from the robots current position. Therefore is usually the total path divided into sub-paths (way points), where sub-goals are assigned and the path could be re-planned when more information is provided along the traverse.

Obstacles are detected by the sensors in an early stage of the navigation, and are e.g. marked as occupied cells within an occupancy map [8]. However, obstacle avoidance is required since all obstacles are not detected during the mapping, this because of *sensor shadows* (unseen areas behind other obstacles) and changes within the environment. Within a static structured environment this is not a major problem. However, in unstructured and changing environments, like crowded places, this could be challenging to handle. Thus research has been done within the area, e.g. the algorithm presented by Hähnel in [13], where data corresponding to moving people was filtered out from the sensor data.

Throughout this chapter, the focus has been on navigation based on a given map model. However, a map model is by no means necessary for carrying out a navigation task. As described by Franz in [10], navigation could occur as long as movement in space is possible and a way of determine if a goal is reached or not is present. Hence simple instinctive navigation behaviours, inspired by behaviours found within the animal kingdom, could be enough even for a navigation task with a mobile robot.

2.6 Discussion

This entire chapter could be summarized by Figure 2.6, which illustrates the dependencies between the methods and techniques that can be used for the purpose of robot navigation.

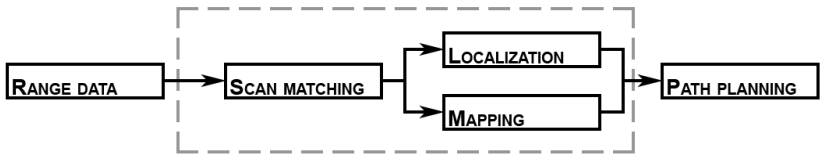

Figure 2.6: Dependencies within the methods and techniques used for a navigation system.

Range sensor data is essential to get an understanding of the surrounding environment, though the challenge remains within how to interpret and use the range information. 3D range data could be acquired by the use of 2D scanner techniques and many different approaches has been presented where the 3D scans are taken in an oscillating pattern. However a common weakness with

such scanning approaches are that scans obtained from a running application can easily by corrupted. Furthermore, factors like sensor noise and bad calibration could also corrupt the range data, hence methods are needed for compensate and secure the reliability within the data. Such results are achieved by scan matching the range data. However, with scan matching are other problems arising, such as computational time which grows proportional against the accuracy within the range data, and which is central for a realtime application. Though, if the time aspect is not a problem or a solution for lowering the computational time is present, scan matching can yield great results for the overall navigation. This because the algorithm does not only secure the reliability within the sensor data, it also secures the location of the robot, which is commonly corrupted by slippage. Reliable data is also crucial for the purpose of representing the range data within consistent map representation of the surrounding environment.

Mapping can be done in a variety of geometrical and topological ways, though should not be said that it is a trivial task and that there exist any standardized method. Opposite, many factors such as the dimension of the range data, the target environment, used robot application, etc., must all be weighted while choosing mapping method. Altogether, if an inconsistent map model is provided as a result of either corrupted range data or badly chosen mapping method, a safe and efficient path could not be planned for based on the map model.

Chapter 3
System setup

The main purpose of this chapter is to introduce the robot platform used for this project - with focus on introducing the 3D scanner that has been used. A description of the *Player software* [1], is also given together with a description of the most important *Player modules* used. All for this project given functionalities and devices are presented in Section 3.1.

The remaining of this chapter is dedicated to present both an overview of the whole system implementation done (Section 3.2), and to give the details about the graphical user interface that was implemented as part of this project (Section 3.3).

3.1 Alfred - the robot

A mobile robot named "Alfred" was used for this project. Alfred once was an electric wheelchair which has been modified and equipped with a standard PC computer, together with a *Sick LMS 200 Laser* module (the main input device for this project) attached on an *Amtech M5 actuator*, a *Hokuyo UTM-30LX Laser* range scanner, and a touch screen, see Figure 3.1.

An advantage of using an electric wheelchair as base for a mobile robot platform is that electric wheelchairs are built for maintaining long battery uptime. Wheelchairs are also designed with focus on robustness and good controllability.

3.1.1 3D input sensor

As sensor input device is this robot using a Sick LMS 200 laser module (see Figure 2.2), which takes 2D scans as *slices* of the environment with the range of 8.1m and a field of view of about 180°. To get a 3D scan the Sick laser was attached on an Amtech M5 actuator, with a full 360° rotation range, in a similar way as described in [29]. This provided for 3D point-cloud representations of the environment.

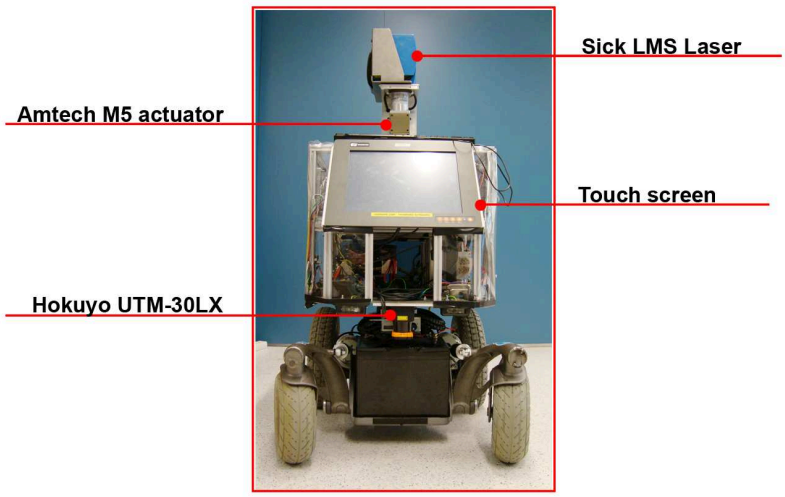

Figure 3.1: Robot Alfred and the devices of interest for this project.

Even though this setup provided for 3D scans with the use of 2D scanner technique, it was not flawless. The actuator require rotating-time to make a full 360° rotation, and the 3D scan is built from 2D scans obtained during the rotation, hence a whole 3D scan obtained while the robot is moving will be distorted as shown in Figure 3.2 (left). Result in that the normally spherical 3D scan is *stretched* over the travelled distance. There are ongoing projects about compensation for the movements while simultaneous scanning and moving [26], however this is not addressed within the scope of this project. Instead, the simple solution of only scanning while the robot was stopped was used. The result of this approach was spherical overlapping scans as shown in Figure 3.2 (right).

3.1.2 Player

A fundamental part of this project was the use of *Player* client and server software, [1]. Player provides an abstract device interface to the underneath complex hardware. This simplified the later implementation thus devices needed for this project were already implemented as Player modules - which provides such control interfaces. Player also provides for better portability between different applications, because all devices with the same functionality but of different physical design use the same abstract interface.

Player work in a *client-server* fashion, where a Player-server is running together with a *configuration file* that specify which devices and corresponding

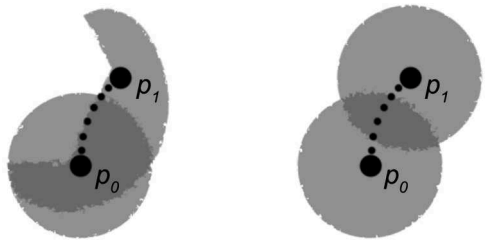

Figure 3.2: *Left* - result of scanning while moving from point p_0 to p_1, *right* - result of a scanning first when the robot has reached point p_1. Darkgray areas represent overlapping areas and scanning at position p_0 was done while the robot was stopped.

interfaces to use. The Player-client is the implementation used for the specific application, where communication with used devices are done through corresponding interface.

The Player modules used for this project, that had direct communication with the system implementation of this project, are as follows:

- *Laser3DScanner*[1] - provided 3D point-clouds as result of 3D scanning (such as described in previous section).
- *Position2dProxy* - provided information about the position in 2D and heading of the robot (odometry data).
- *MapDriver*[1] - provided support for dynamically handle occupancy maps.
- *PlannerProxy* - provided standardized path planner (further described in following section).

Detailed Player configuration files used for this project is given in *Appendix A*.

Wavefront path planner

Path planning is not a trivial problem, and the choice of algorithm depends on the situation in which it is to be used and the provided map representation. However, since the *Wavefront* path planner was given trough a Player module [1], and was thus used for this project, the choice of algorithm was omitted and the effort in implement and integrate the same could be simplified by far.

The wavefront planner works in a wave pattern from the given goal position G outwards until, if possible, the start (current) position of the robot S has been reached. Where all visited cells along the wave are assigned with a travel cost

[1]Specifically written for the Learning System Lab., not part of the normal Player software.

value so that the robot traverse a path that follows the gradient of the closest neighbouring cell, as shown in Figure 3.3.

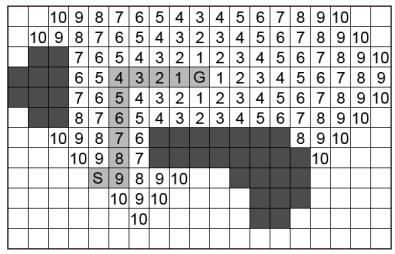

Figure 3.3: Wavefront path planner - the algorithm is working its way outwards from given goal G to current position S while assigning a travel cost value at each visited cell. (Obstacles are represented by darkgray areas, and the best path is given by lightgray area).

The Player Wavefront planner is close connected with the odometry (as part of the functionality behind Player), and thus is the start position S always known by the planner. Therefore, during the implementation it was only required to provided the planner with coordinates of given goal location G and a representing occupancy map. However, the planner requires a 2D occupancy map while the mapping used for this project provides approximated 3D maps. Hence is translation from 3D maps into 2D occupancy maps an important aspect of this project, and which will be further addressed in Chapter 5.2

3.2 Implementation overview

Figure 3.4 gives an overview class diagram of the implementation for this project. The implementation is written in C++, and thus a class hierarchy was used. This class hierarchy will be presented throughout this section with a short description of each implemented class - though with focus on the *Navigation* class since this is the centre of the implementation and which both ties together and separates the remaining classes and the Player software:

- *Navigation* - main purpose of this class is to provide an interface between the Player software and the rest of the application, and thus limit the Player-dependency to only one class. The task for the Navigation class is to receive point-clouds from a *Player Laser3DScanner* module, scan match the received point-cloud with previous point-cloud through a ICP-object, and add the matched point-cloud to the global map model, represented by a MLS-object. Other Player related task for the Navigation class is to keep track of the robots position through its odometry and

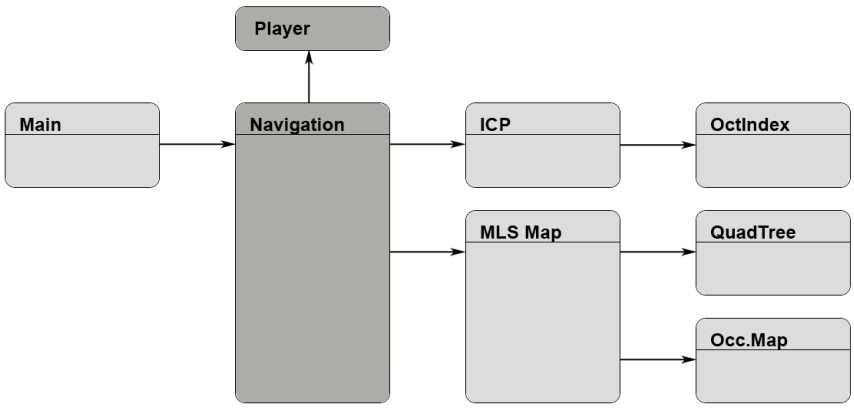

Figure 3.4: Overview class diagram of the whole implementation for this project.

to receive movement commands from the user and translate those commands into movements for the robot - all through communication with the Player software.

- *Main* - provides a simple human-robot interface. The implementation of the interface was written in OpenGL; hence the normal class representation could not be used since OpenGL requires its standard listening methods. Instead a global *Navigation*-object was used, with corresponding wrapper methods that was processing input commands forwarded by OpenGL listening methods.
- *ICP* - this class handles the scan matching of point-clouds provided by Player and passed down by the Navigation class, and are therefore also the class that consumes the majority of the computational time of the application. Because of the computational time aspect, all objects of this class were running as self-standing processes to secure that the rest of the application was not locked during the registration.
- *OctTree* - this class provided an implementation of a tree structure with up to eight children of each node. This tree structure was used by the ICP class to improve the performance of the nearest neighbour search.
- *MLSMap* - this class handles the building of the map model, and was thus together with the QuadTree class, also working as the whole map model. This class also handles the traversability analysis, where an occupancy map was created based on the map model.

- *QuadTree* - provided another tree structure, where each node had up to four children and in which all surfaces of the map model was stored. User feedback was also provided by this class in form of rendering of the map model, which was done directly within the nodes of the tree structure.
- *OccMap* - used to build an occupancy map, thus created as a result of the traversability analysis. This class also had the functionality of sending the occupancy maps directly to listening *Player MapDriver* module over a TCP connection (this connection was not done in the normal Player client-server fashion, and was thus also done without interfering the Navigation class).

The reason of using this class hierarchy was that the implementation, which is now part of the implementation repository of the Laboratory in use, could be reused and modified with ease. Where the used techniques could be replaced with similar solutions, while the main part of the application remains unchanged (assumption that the same methods and connectivity functionality are provided).

3.3 Graphical user interface

Since Alfred was not providing a suitable interface for human-robot interaction, and it was required to get feedback from the system in shape of resulting map representations, a graphical human-robot user interface was to be implemented within the scope of this project.

Available on the market of today there is a huge variety of interfaces, all from very simple e.g. touch buttons and LCD displays on the front of kitchen machines, to the very advance e.g. flight simulators. However, for this project a simple and easily controlled interface was to be favoured. The reasons for simplicity is that the robot is equipped with a touch-screen and that all human-robot interaction should preferable also go through that screen (or together with the use of a normal PC-mouse).

3.3.1 Design description

Though the goal was to design an easily controlled interface, some basic functionality was required. However, the main interest throughout the implementation was related to getting feedback about the robots surrounding:

- It should be possible to give simple movement commands. This should be done by just pointing within the scene of the environment.
- A 3D view of the map model should be given.

- A smaller digitalized-model of the robot should been given. This model should also match the real robots rotation roll α, pitch β, yaw γ, and keep the robots real position x, y, z, in the centre of the screen.
- It should be possible to zoom and rotate the rendered map. This for a better overview since other perspectives of the environment could be viewed.
- It should be possible to switch to a view over the occupancy map (at the robots current height level) instead of the map model.
- For clarity, it should also be possible to switch to a top-view of the map model instead of the 3D view. Where only surfaces at same height level as the robot should be rendered for display purposes.
- It should be possible to save maps and load previously constructed maps.
- Logging the mapped data should also be possible.

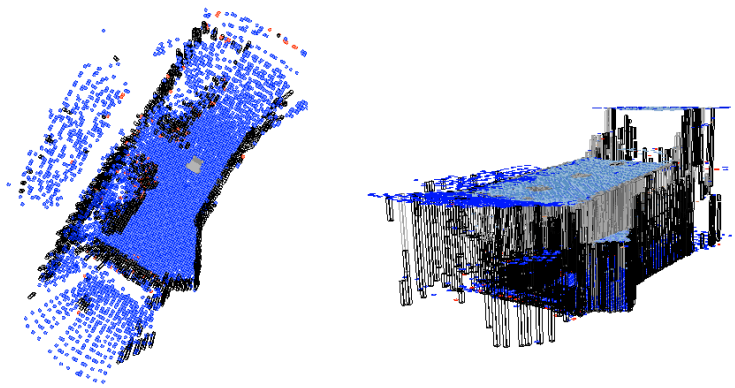

Figure 3.5: A map that represent the end of a corridor. *Left* - top-view with the robot seen, *right* - 3D view with the robot unseen within a *box* representation of a corridor.

The reason for using more than just the *standard* 3D view with the view-point above and slightly behind the robot, is that this view-point would not provide a clear overview of the whole environment. Because of e.g. corners and objects in the environment, the whole 3D scene could not be *seen* in the same way as it could if a top-view is provided. A *dull* example of this can be seen in Figure 3.5, where the robot is driving within the end of a corridor. Where the robot can be seen if a top-view (left) is used, whereas a 3D view (right) is used instead, with the view-point slight above and *outside* the corridor, the robot is unseen within a *box* representation of the corridor.

The reason for using the possibility to switch to a view over the occupancy map is that this provides a better understanding of the path planner's decisions about possible paths within the environment.

3.3.2 Implementation details

For the implementation of the user interface OpenGL was used. This provided standardized functionality for graphical implementations. OpenGL requires methods that are part of the OpenGL framework, thus the normal class representation could not be used for this part of the implementation. Instead a global Navigation-object with corresponding wrapper methods was used, shown in Figure 3.6.

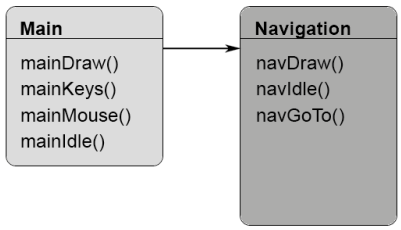

Figure 3.6: Class diagram of the graphical interface part of the implementation, with Main OpenGL methods and corresponding wrapper methods in the Navigation class.

OpenGL requires standardized listener methods for input devices (keyboard, mouse, etc.). With corresponding wrapper methods in the Navigation class was this handled - which controlled the rest of the application based on the given input commands. OpenGL listener methods *mainKeys* and *mainMouse*, listen for input commands. Those commands are sent further to related wrapper methods in the Navigation class, where *navGoTo* receives given goal coordinates, while *navDraw* receives graphical related commands - which map-view to draw. The trigger of the rendering is the *navDraw* method, which also triggers the rendering of the entire map model. OpenGL also has an idle method, *mainIdle*, which is calling the *navIdle* method - the main Player-communicating method.

Chapter 4
Scan matching

A robots ability to estimate the position with use of the odometry could never be fully reliable; especially not in an outdoor environment where the friction against the ground is affected by the weather, season of the year, and terrain. Smaller position errors are usually accumulating together with smaller errors in the heading of the robot, which consequently also affect the estimation of the position, and together and over time they will have an enormous effect on the total estimation of the global position. Thus, neither the position nor the orientation of provided 3D scans are reliable, and where the scans are further distorted by sensor noise. An example of this problem is shown in Figure 4.1, where the sensor data, obtained along a straight hallway, is added to the map model without any compensation for position errors.

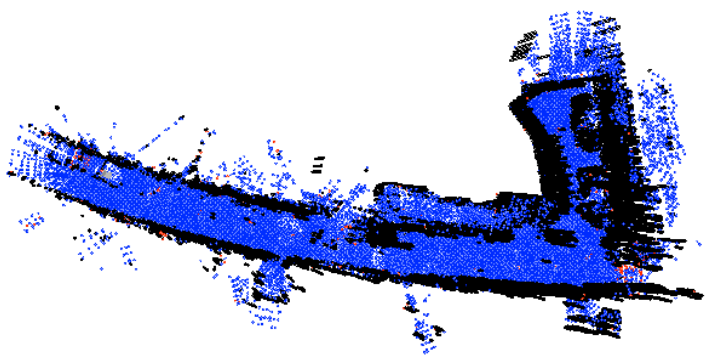

Figure 4.1: The result of adding input sensor data directly to the map model without any compensation for position errors (top-view images).

To compensate for this kind of growing position errors, the *iterative closest point* (ICP) algorithm was used for this project, where the implementation was

completely done from scratch without any use of existing third part solutions. A solution of using a *kd*-tree for both improving the nearest neighbour search and to remove non-overlapping point-pairs has also been implemented. Since available 3D scanner was providing raw point-clouds, the algorithm could directly be applied on the sensor data for this implementation. Consequently, the latter mapping was done with the assumption that position errors had been compensated for and that scans added to the map model was in alignment. As an alternative to the ICP algorithm, a landmark-based localization method could have been used. However, such approach would require that landmarks was extracted from the sensor data (as described in Section 2.4.2).

This chapter is structured so that the notation followed during the implementation of the ICP algorithm is presented in the beginning of Section 4.1, and the remaining of the section describes the implemented strategies for improving the algorithm by reducing the computational time of the origin. While Section 4.1 focus on the theory behind the implemented scan matching algorithm, Section 4.2 extends the theory to describe localization as a result of scan matching. Aspects of interest together with decisions about the done implementation is presented in Section 4.3. To conclude this chapter and to illustrate the ICP algorithms effect on this application, a minor validation is presented in Section 4.4.1, where alternatives to operating on the raw point-clouds are presented as well.

4.1 Iterative closest point

The iterative closest point (ICP) algorithm could be used for any kind alignment problems, where in case of a mapping and navigation problems the goal is to find the Euclidian transformation that brings a new 3D sensor scan, the data-set $D = \{d_1, ..., d_{N_D}\}$ of size N_D, into the best possible alignment with previous sensor scan(s), the model-set $M = \{m_1, ..., m_{N_M}\}$ of size N_M.

The basics of the algorithm is divided into four main parts and is given in Algorithm 1, which will be explained throughout the rest of this section:

Algorithm 1 - ICP

1: Find closest point-pairs.
2: Compute the transformation that minimizes the *mean squared error* (MSE).
3: Apply the calculated transformation on the data-set D.
4: Terminate if error is below threshold, otherwise *go to 1*.

1. For all points in the data-set D, find by distance the closest point (nearest neighbour) in model-set M:

$$dist(d_i, M) = \min_{m \in M} \|m - d_i\|, \qquad i = 1, ..., N_D \tag{4.1}$$

The number of points in both point-sets N_D and N_M are necessary not of same size $N_D \neq N_M$ (depending of how many beams that actually was hitting a surface), and the resulting set of closest point-pairs will naturally be of size N_D - which will in the rest of the section simply be referred to as N. Also, the new set of neighbouring points from the model-set will noticeably not be the same as the original model-set, and will thus be denoted X where all points in set $X = \{x_1, ..., x_N\}$ such that $x_i \in M$.

2. Compute the transformation described by the rotation matrix R and translation vector t, that minimizes the *mean squared error* (MSE) between the found neighbouring point-pairs:

$$E(R, t) = \frac{1}{N} \sum_{i=1}^{N} \|x_i - (Rd_i + t)\|^2 \tag{4.2}$$

The first step towards the final computation of the transformation is to calculate respective centroids of each point-set, defined by:

$$\mu_D = \frac{1}{N} \sum_{i=1}^{N} d_i, \qquad \mu_X = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{4.3}$$

And by the calculated centroids, compute the variance for each point in respective point-set:

$$\hat{D} = \{\hat{d}_i = d_i - \mu_D\}_{1,...,N}, \qquad \hat{X} = \{\hat{x_i} = x_i - \mu_X\}_{1,...,N} \tag{4.4}$$

Those variance point-sets are in next step used to compute the correlation matrix S, defined by:

$$S = \sum_{i=1}^{N} (\hat{d}_i \hat{x}_i^T) = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{bmatrix} \tag{4.5}$$

with $S_{xx} = \sum_{i=1}^{N} (\hat{d}_{ix} \hat{x}_{ix}^T)$, $S_{xy} = \sum_{i=1}^{N} (\hat{d}_{ix} \hat{x}_{iy}^T)$,...

The optimal desired rotation $\hat{R}$ was obtained by direct decomposition of a orthonormal matrix (assuming a matrix size of 3x3 and that all eigenvalues are positive), [15]:

$$\hat{R} = S(S^T S)^{-0.5} \tag{4.6}$$

The square root of the matrix was found by *Denman-Beavers square root iteration*, [4]:

$$Y_{k+1} = \frac{1}{2}(Y_k + Z_k^{-1}) \tag{4.7}$$

$$Z_{k+1} = \frac{1}{2}(Z_k + Y_k^{-1}) \tag{4.8}$$

With initial values $Z_0 = I$ (identity matrix) and $Y_0 = S^T S$. Such that Y_k converges against the square root, while Z_k converges against the inverse of the square root - the one of interest for this implementation.

3. Apply the computed transformation on the data-set D. Where the new alignment was calculated by the updated rotation matrix R and translation vector t (initialized as identity matrix and a zero vector respectively):

$$R_{k+1} = R_k \hat{R} \tag{4.9}$$
$$t_{k+1} = \hat{R} t_k + (\mu_X - \hat{R}\mu_D) \tag{4.10}$$

And the data-set was updated by the rotation matrix and transformation vector (denoted R_{k+i} and t_{k+1} above):

$$d_i = R d_i + t, \qquad i = 1, ..., N \tag{4.11}$$

4. Calculate the new mean squared error by the updated data-set:

$$e_k = \frac{1}{N} \sum_{i=1}^{N} \|x_i - d_i\|^2 \tag{4.12}$$

If the difference between the new error and previous error is below a fixed threshold, $e_k - e_{k-1} < \tau$, then stop the algorithm. Otherwise, go to step no.1 and start over with a new point-pair search.

The ICP algorithm has frequently been used together with the mapping concept during the past decade and has shown to yield excellent results as described by Nüchter in [20]. However, the algorithm in its origin form is also inhibited by a tedious nearest neighbour searching (as described in Section 2.3.1) - which grows proportional with the size of the point-sets. Thus are improvements of this step commonly of interest within research with use of the ICP algorithm, and solutions of such improvements are presented throughout the rest of this section.

4.1.1 kd-tree

A step towards overall improvement of the ICP algorithm is through improvement of the nearest neighbour (point-pair) searching step. The most commonly used approach, and which has demonstrated to yield good results as described by Nüchter in [19], is to improve the searching by a *kd-tree* - a tree structure of k-dimensions (kd) with non-fixed node sizes.

However, for this project *octtree* was used, which is a version of the kd-tree but with fixed node sizes. A octtree is a 3D tree structure with up to eight children in each node; where the space for the children is divided into eight

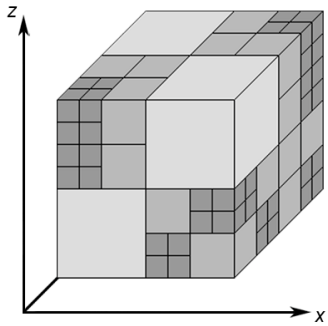

Figure 4.2: An example of an octtree, where every node could have up to eight sub-nodes.

evenly sized cubes, shown in Figure 4.2. During the building of the tree it is the coordinates of each point that decides which sub-node the point falls within, and the number of points within each sub-node decides the depth of the tree.

The tree is built during a single pass of the model-set M, but is used N_D times to find closest point-pairs, for each iteration. Therefore, it is advisable to dedicate time to the construction of the tree and to set up as good and effective searching methods as possible. To improve the searching further was for this project the tree built with a fixed minimum leaf size of 1dm. Thus prevent the tree from growing uncontrolled - resulting in unnecessary searching in the depth for a closest neighbouring point. Also, rather than building a tree consisting of the raw points from the model-set M, the model-set was only passed down the tree by reference, while the index of the points was stored in the tree structure.

For this project the implemented tree structure gave a great computational time improvement compared to a solution with a *brute search* (all elements in both set compared against each other), where the time required for a search with the tree structure was only 0.16% of the time required for a brute search.

4.1.2 TrICP

The *trimmed ICP* (trICP) algorithm was first introduced by Chetverikov in [3], and the main purpose of the algorithm is to lower the number of point-pairs used by removing pairs with a high distance in between, and thus reduce the computational cost. The algorithm works in similar way as the origin ICP, but with a few changes related to the closest neighbouring search:

2. Sort the point-pairs in ascending order and select a number of N_T closest pairs with low pair-distance for later use, while the remaining point-pairs are removed. This will result in a smaller amount of total points used

Algorithm 2 - Trimmed ICP

1: Find closest point-pairs.
2: Sort the point-pairs in ascending order and select the N_T closest pairs.
3: Compute the transformation that minimizes the *mean squared error* (MSE).
4: Apply the calculated transformation on the data-set D.
5: Terminate if error is below threshold, otherwise *go to 1*.

compared to the original approach $N_T < N$, and in Eq. 4.2-4.5, 4.11, 4.12, is N consequently replaced with N_T.

For this project an approach similar to TrICP was used both to lower the number of points used and to secure that only overlapping regions was used. Since all points were accessed through an octtree (described in previous section), and point-pairs were found by recursively going further down the tree, the branch level could directly be used to find the minimum distance to a neighbouring point in this case. The main idea behind this is that a sub-node of a node is allocated during the construction of the tree if a point falls within that sub-node. Thus, in case the search for a neighbouring point reaches a node where next recursive step would be to travel down a *unallocated* sub-node, the closest distance to a neighbouring point is within the current node. And if such a node is reached on a high level, this means that the minimum distance to a neighbouring point is also to be found on a high level - minimum distance is the same as the resolution of the node. For this project a level was chosen so that the minimum accepted distance between point-pairs was 0.25m.

The difference from the origin TrICP algorithm was thus not the removal of a fixed number of N_T pairs with high neighbouring-distance, rather was all pairs with a neighbouring-distance above a fixed threshold removed. This approach resulted in a further time improvement where the required time for a search was 0.85% of the time required for search without removed point-pairs. But more important, this approach did guaranteed that only overlapping points was considered.

4.2 Localization

An uncertain estimation of a robots location within the environment, given by the odometry and distorted by noise, could be corrected if a trusted map model is provided as a result of scan matching. Odometry are commonly obtained by measuring a robot's wheel rotations, and are consequently also described as the position in 2D and the heading as the rotation around one axis. If the position in 3D and the rotation around all three axis are of interest, must normally additional input devices be used, e.g. an internal compass. However, if a 3D transformation was given as result of registration, the location and rotation

in 3D could be approximated with the use of the same transformation, even though 2D odometry was provided.

Given by the scan matching algorithm was for each registration the rotation matrix R and translation vector t by which the data-sets was transformed. Thus, the total global rotation and translation could be maintained with the result of each registration:

$$R_g = R_g R \tag{4.13}$$

$$t_g = R t_g + t \tag{4.14}$$

The odometry provides a rough estimation of the robots position x_{odom} and y_{odom}. Denote the global position of the robot by $p \in \mathbb{R}^3$, such that:

$$p = \begin{bmatrix} x_{odom} \\ y_{odom} \\ 0.0 \end{bmatrix} \tag{4.15}$$

Hence the robots position in 3D could be calculated with the use of the global rotation matrix R_g and translation vector t_g:

$$p = R_g p + t_g \tag{4.16}$$

The rotation however, is by the odometry provided as the rotation θ_{odom} around one axis (z-axis). To calculate the global orientation around all three axis - rotation angles roll α, pitch β and yaw γ around axis x, y and z respective - the rotation matrix R_g must be used to calculate respective *Euler* angle, as described in [25]. Given the global rotation matrix:

$$R_g = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz}] \end{bmatrix} \tag{4.17}$$

Two possible *pitch* angles β could be calculated:

$$\beta_1 = -\sin^{-1}(R_{zx}) \tag{4.18}$$

$$\beta_2 = \pi - \beta_1 = \pi + \sin^{-1}(R_{zx}) \tag{4.19}$$

Even though two possible angles exist, the assumption was made that only the *closest* angle was of interest if an adequate initial alignment for the matching algorithm was provided. Thus the pitch angle was shortly denoted by β and which refers to the closest angle. The *roll* angle α was calculated by:

$$\alpha = \operatorname{atan2}(\frac{R_{zy}}{\cos\beta}, \frac{R_{zz}}{\cos\beta}) \tag{4.20}$$

And the total *yaw* angle γ is finally obtained as the sum of the Euler angle and the heading from the odometry:

$$\gamma = \operatorname{atan2}(\frac{R_{yx}}{\cos\beta}, \frac{R_{xx}}{\cos\beta}) + \theta_{odom} \tag{4.21}$$

Where $\cos\beta$ is used in above equations to assure the right sign as a chain-result of the possibility of two solutions for the *pitch* angle.

However, all above angles could only by calculated with this notation if $R_{zx} \neq \pm 1$. Otherwise exists two cases and where roll angle is irrelevant and could be set $\alpha = 0$. In the first case where $R_{zx} = -1$, will the pitch and yaw angle be calculated as follows:

$$\beta = \pi/2 \tag{4.22}$$

$$\gamma = \alpha + \mathrm{atan2}(R_{xy}, R_{xz}) + \theta_{odom} \tag{4.23}$$

And in the second case where $R_{zx} = 1$:

$$\beta = -\pi/2 \tag{4.24}$$

$$\gamma = -\alpha + \mathrm{atan2}(R_{xy}, R_{xz}) + \theta_{odom} \tag{4.25}$$

4.3 Implementation details

Even though the scan matching was improved with a tree structure and trimmed point-pairs, the entire registration required some computational time[1] - which was proportional to the initial alignment between and the size of the both matched point-sets. Hence from the implementation point of view, the first evident step was to secure that the rest of the application was not locked up during the registration. This was accomplished by implementing the scan matching algorithm as a self-standing process, represented by the dashed line in Figure 4.3.

Another aspect that was considered during the implementation was memory allocation. The goal here was to allocate as little memory as possible during the scan matching - which moderately was accomplished by using point-references, instead of raw points, during the construction of the used tree structure.

The implementation of the ICP algorithm was otherwise following the notation described throughout this chapter. The Navigation class creates a new ICP-object when a new point-cloud is provided by the 3D scanner, and passes the new cloud together with the model-cloud (the last and corrected cloud) by reference. The ICP class builds an index of octtree structure (OctIndex), which is used during the nearest neighbouring search, and the algorithm is executed until convergence. All classes and their dependencies are shown in Figure 4.3.

4.3.1 Fault control

As mentioned in Section 2.3, it is of importance that the ICP is provided with a good initial alignment and that the two point-clouds are partly overlapping. This is as a rule not the case for a running robot application, therefore a couple of fault controls were added to the algorithm:

[1] Around 5s on a Acer Aspire One laptop with 1.6GHz processor and 500MB memory.

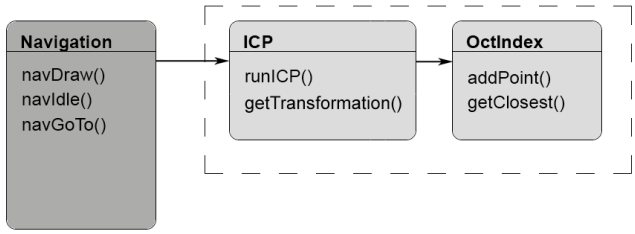

Figure 4.3: Class diagram of the scan matching part of the implementation.

- If there exist a best alignment in between two point-clouds, this alignment should be found in a finite number of iterations. To prevent the algorithm from searching endlessly for a non-existing alignment was a fixed maximum number of iterations used.
- The initial alignment is roughly given by the odometry of the robot, and the resulting transformation is used to *fine adjust* the alignment between the both point-clouds. Therefore, final transformations that resulted in a large translation, $\mathtt{length(t)} > 2.0\mathrm{m}$, were discarded.

4.4 Validation

To illustrate the power of the ICP algorithm and to give an indication about the algorithms effect on this application, a simple validation experiment was done. This experiment was done on a series of point-clouds obtained along the hallway right outside the Laboratory in use, shown in Figure 4.4. During the travelled distance was a total of 15 point-clouds[2] acquired, where the raw point-clouds was stored together with the odometry readings at each scan. The reason of storing offline point-clouds instead of running the experiment in real-time was that the experiment could be repeated and done with different settings - which will be addressed in following section.

4.4.1 Types of point-set

As mentioned in Section 2.3.1, the ICP algorithm could be applied either directly on the raw point-set or on features extracted from a point-set or a map model, assuming that the original 3D structure is preserved within the features. Features are generally represented by a lesser amount of data than the original point-sets, which consequently also requires lower computational cost during registration. Therefore, of interest it was to examine if the original input data

[2]Originally, there was 16 point-clouds taken, but one cloud in the middle of the series was corrupted and had to be removed - resulting in a more challenging matching.

Figure 4.4: The hallway outside the AASS laboratory.

could be represented by features instead, and meanwhile maintain or even improve the performance.

Features are commonly found by locating patterns within the points, e.g. points representing lines, corners, etc. Though for this project the height of the surface together with the x and y coordinate of the grid-cell for the surfaces within the multi-level surface map was used. Thus, resulting point-sets could not be considered as *real* features, but rather as a down-sampling of the original point-set. However, for simplicity even this approach was considered as a feature-based approach, and where the combination of different approaches was divided as follows:

- *Point-to-point* - the both raw point-sets are directly matched against each other (the origin approach).
- *Point-to-feature* - the raw data-set is matched against a point-set given by features from the map model.
- *Feature-to-feature* - both the data-set and the model-set are given by features extracted from models (where the prior is given by a local map).

While keeping the same performance, the main parts of interest here was; how much could the total number of points within the point-clouds be reduced[3], and how much could the computational time be decreased. To examine this, the same series of point-clouds was used as for the rest of this experiment. The results are presented in Table 4.1, where the results are presented as the mean over all registrations done on the complete series of 15 point-sets.

[3]The reduction in size of point-clouds are naturally affected by the resolution used for the surfaces within the map model.

	Mean used:		
	Points	*Time*	*Transformation*[4]
Point-to-point	100.0%	5.0s	1.00m/3.13°
Point-to-feature	63.8%	4.7s	1.56m/3.73°
Feature-to-feature	23.2%	1.1s	1.04m/2.30°

Table 4.1: Comparison between different types of point-sets.

As seen in Table 4.1, reduction in computational time than using point-to-features registration was neglected compared to point-to-point registration. The reason of the low *time improvement* was that even though one of the both point-sets was reduced in size, the closest point-pairs must still be found, and where all points within both point-set must be considered.

Full map match

A resulting benefit of using a feature-based approach is that the new model-set could be built based on the entire map model. Thus, all previous scans could be considered while matching a new scan, instead of just the previous scan(s). The resulting of this is a more confident map, where faulty registrations do not start of an accumulation error within the map model - as would be the case if a new scan was faulty matched against the previous scan(s).

4.4.2 Results

The result of scan matching the point-clouds before adding them to the map model is shown in Figure 4.5, and the final global rotation matrix and translation vector is presented in Table 4.2. Through comparing the result of this experiment with Figure 4.1, given in the beginning of this chapter, it is verified that accumulating errors in the estimation of the position was indeed present for this application. Though the resulting translation indicates that the data was not only bent vertical, as seen in Figure 4.1, but also horizontal since the final translation was relatively high in all three dimensions.

[4]Transformation described as the absolute distance in translation and the rotation as the absolute sum of the Euler angles.

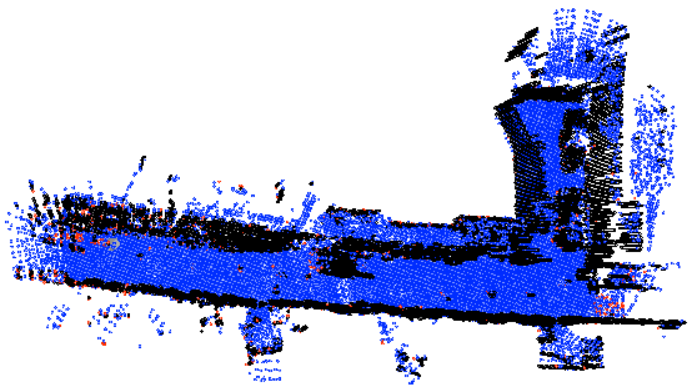

Figure 4.5: The result of adding input data after scan matching with the ICP algorithm.

Rotation matrx (R)		
0.837224	-0.489855	-0.243102
0.524468	0.845148	0.103238
0.154886	-0.213933	0.964491

Translation vector (t)		
-6.41118	-5.76178	-4.61427

Table 4.2: The final global rotation matrix and translation vector given as a result of scan matching the series of point-clouds used for this validation.

Chapter 5
Mapping

The traditional way to represent the environment for a mobile robot is through a *occupancy map* [8], where the cells in the grid are either *drivable*, *unknown* or *occupied*. This is usually enough information for indoor navigation on a flat surface (e.g. in an office complex with overhanging objects mainly represented by tables and desks). However, for outdoor navigation, where the surface is less than flat, and where overhanging objects (e.g. branches on trees, tunnels, low roofs, etc.) together with holes (or even abysses) in the ground are more common, a 2D representation of the environment is insufficient, since drivable areas under overhanging objects could be considered as occupied and holes in the ground could be considered as unknown areas. A more suitable alternative for outdoor navigation would therefore be to use a map model where heights could be represented, e.g. a elevation map or a 3D map. However, in a similar way as for occupancy maps, overhanging objects and holes in the ground could moderately be represented within elevations maps, while 3D maps could easily grow out of proportions.

An alternative to both elevation maps and 3D maps are *multi-level surface* (MLS) maps, which provides a rough 3D representation without the complexity of a 3D map. This map representation was thus used for this project, and where the theory behind MLS maps which was followed during the implementation is described in Section 5.1. The implementation of the mapping algorithm was completely done from scratch, and where the used notation was assuming that as result of scan matching, the sensor data was in alignment.

A new approach of using a 2D tree structure to represent the MLS map model is also presented throughout this chapter. As mentioned in Section 3.1.2, the Player Wavefront module requires a 2D occupancy map, whilst the MLS map is providing a 3D representation. Hence, translation between MLS maps and occupancy map became another important aspect during the implementation of this project. In this case, translation refers to *finding* the traversability of a geographical area at specific height level. Thus this translation turned into a problem of searching nature. With the tree structure used for nearest neigh-

bour search described in Section 4.1.1 in mind the inspiration of representing even the map model with tree structure was given, and which is described in Section 5.1.1. The traversability analysis and the importance of considering the neighbourhood is presented in Section 5.2. An approach for reflecting changes in the map model is presented in Section 5.3, before this chapter is concluded by pointing out aspects of interest related to the implementation in Section 5.4.

5.1 Multi-level surface maps

Multi-level surface (MLS) maps consist essentially of a 2D grid structure, where each cell c_{ij}, $i, j \in \mathbb{Z}$ stores a list of *surfaces* $S^1_{ij}, ..., S^k_{ij}$. A surface is represented by the mean μ^k_{ij} and the variance σ^k_{ij} of all height values z_{ij} classified as belonging to surface k that fall within the area of cell c_{ij}. This will give an 3D representation of the environment, where the mean μ^k_{ij} reflects the possibility to drive at a given height level k while the variance σ^k_{ij} gives the uncertainty of driving at the same level.

To keep the notation consistent with the previous chapters, a 3D point-cloud with N_D points are denoted by $D = \{d_1, ..., d_{N_D}\}$, with $d_i \in \mathbb{R}^3$. Where D is used since a transformed point-cloud D is provided as result of scan matching a new sensor scan with previous sensor scan(s), the model-set M. In difference from [28] where uncertainty within measured distance was considering, we are here confident in the point-clouds, as long as the robot is stopped while scanning, and are not considering any uncertainty that grows with the measured distance of each point.

The global MLS map is created as follows:

- For each cell with index (i, j), collect all points $d = (x, y, z)$ belonging to D so that $s(i) \leqslant x \leqslant s(i+1)$ and $s(j) \leqslant y \leqslant s(j+1)$, where s denotes the size of each cell given by the fixed cell-side-length r.

- From here on it is only the height values z of the collected points that are of interest. For each cell, calculate a set of surface intervals from the collected height values, where height values are classified as belonging to the same surface interval if they all fall within the Gaussian given by the mean μ of all height values within an interval, and a fixed variance represented by a gap size γ. This guarantees that the surface intervals are at least separated by a distance of γ meters. The gap should be chosen so that the robot could navigate without any problem in between the intervals, and should therefore also be larger than the total height of the robot. For this project a gap size of $\gamma = 2.0\text{m}$ was chosen so that the robot with a total height of 1.5m, without restraints could navigate in between the intervals.

- Calculate all surfaces in each cell, where a surface is stored as the calculate mean μ value for each interval together with the true variance σ given by the height values in each interval, shown in Figure 5.1.
- Classify each surface in each cell according to the variance; *flat surface* if $\sigma < 5\text{cm}$, *rough surface* in case the surface is not flat and $\sigma < 10\text{cm}$, and *occupied surface* if $\sigma \geqslant 10\text{cm}$.

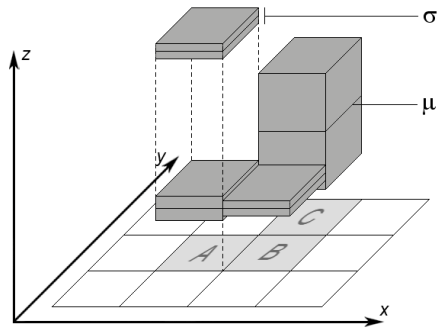

Figure 5.1: Each cell in the MLS map could have many surfaces (cell A), where each surface is represented by the mean μ and the variance σ of the height values within each cell. Surfaces with low variance (cell B) and with a *gap* distance to other surfaces (cell A) are considered as drivable surfaces, while surfaces with high variance (cell C) are considered as obstacles.

The above notation is only true for empty cells in the map. If a new point-cloud D is added and points from the cloud fall within a cell with already calculated surfaces, all previous added points must also be considered. By keeping track of the total number of points n that falls within each cell and each surface, the mean and variance of already existing surfaces could be updated with the height value z of a new point that fall within:

$$\mu_{k+1} = \frac{n\mu_k + z}{n+1} \tag{5.1}$$

$$\sigma_{k+1} = \frac{n\sigma_k + (z - \mu_{k+1})(z - \mu_k)}{n+1} \tag{5.2}$$

After all surfaces have either been created or updated, the point-cloud can be removed from the map model. This step provides the biggest difference between full 3D maps and MLS maps, since all points would have to be stored within a full 3D map.

5.1.1 Quadtree

In difference from similar experiments with multi-level surface maps [28], where a normal gridmap representation was used, this project used a quadtree to represent the map model over the environment. This was done with the intuition that a tree structure would simplify the latter traversability analysis, where the traversability of a corresponding geographical area at specific height must be extracted from the map model.

A quadtree is a 2D tree structure in which each branch (node) has up to four children (sub-nodes). Traditionally, each node is filled up with samples (points) until a certain amount is reached, and when the amount is reached, the node is split into sub-nodes and the points are reassigned to a corresponding sub-node, as shown in Figure 5.2 (left). However, for this project a less common method was used of assigning each point to corresponding *leaf-node* directly by splitting down each node until the fixed leaf level was reached, shown in Figure 5.2 (right).

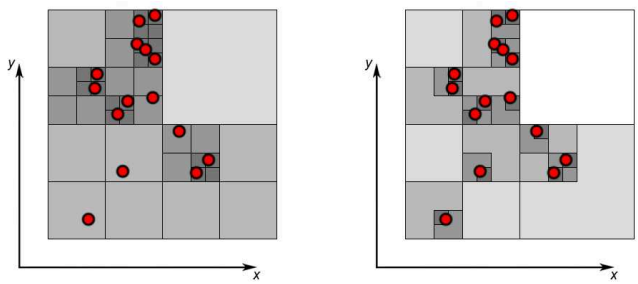

Figure 5.2: *Left* - a quadtree built in the traditional way by splitting nodes into sub-nodes when a certain amount of samples within the node has been reached (split on 2 samples in this case), *right* - a quadtree used for this project where the nodes were split directly into sub-nodes until a fixed leaf level was reached. (Graylevel in each node represent level of allocated memory.)

An advantage of using a tree structure is that quadtrees do not have fixed root size. Hence, if the map grows outside the boundaries of the map, the root could simply be extended and with the previous root as one of the sub-nodes of the new root. However, the main advantage of using quadtrees are that tree structures are very suitable for searching problems. By recursively going further down the tree until the desired leaf is found, the time for finding the wanted surface could substantially be decreased compared to a *normal* grid representation, where in worst case and without any indexing, all cells within the grid have to be considered before the wanted surface is found.

5.1.2 Pre-processing

Even though the 3D scanner was providing detailed information about the robots environment, it was not flawless, and where the flaws were much related to the measured distance. However, not with far distances as could suspected, but rather with short distances. Because of the size of the robot and the location of the 3D scanner, it occurred that the robot *saw* itself (beams was hitting the top of the robot) - resulting in that the robot was mistaken for an obstacle. To solve this problem pre-processing was performed, where all points within a radius $r = 1m$ of the 3D scanner were removed prior adding the point-cloud to the map model.

Another consequence of attaching the 3D scanner on the top of the robot was that the area right underneath could not been seen, shown in Figure 5.3 (left). This problem will naturally be solved after a couple of scans when the robot has moved from the initial position. However, in an attempt to accelerate this, the assumption was made that the robot was initially standing on a relatively flat (and existing) surface. Therefore, a flat disc surface with a radius $r = 1.25m$ was fitted underneath the robot, as shown in Figure 5.3 (right). This surface did also become the initial ground for the entire map model and to which the global coordinate system was locked.

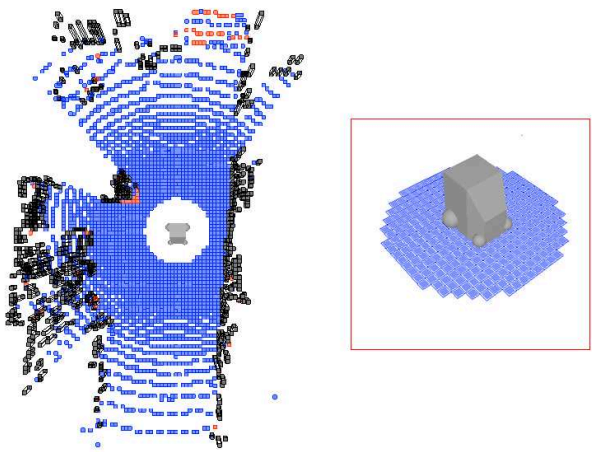

Figure 5.3: *Right* - unseen area underneath the robot, *left* - a flat disc fitted underneath the robot (red border added for clarity).

For simplicity, all pre-processing operations were done directly on the raw point-clouds. The alternative would be to locate and operate on the surfaces with corresponding pattern within the map model. Pre-processing was done

through a single pass of the point-cloud. Hence, a more accurate map model could be maintained at low computational rate.

5.2 Traversability analysis

Traversability analysis together with the MLS concept is about translating a given height level k of the MLS map into a 2D occupancy map - a slice of the MLS map at given height k. An occupancy map is like the MLS map a grid structure, but for this kind of maps a value $0.0 \leqslant t \leqslant 1.0$ is stored in each cell c_{ij} instead of a list of height values. This value determines the state of the each cell such as; a *drivable* cell is represented by a maximum value of 1.0, a *occupied* cell is represented by a minimum value of 0.0, and all values in between is representing a *unknown* cell. This is by basic given by the same classification of surface that is described in Chapter 5.1, but with the neighbourhood in consideration in addition:

- *Drivable* - the cell is classified as a flat surface or the cell is empty but the majority of the neighbourhood is classified as flat surfaces at given height level k.
- *Occupied* - the cell or any of the cells in the neighbourhood are classified as occupied surface (variance $\sigma > 0.1\text{m}$) at the given height level k. A cell is also considered as occupied if the height difference between surfaces in the neighbourhood is above a level of $> 0.1\text{m}$, or if the majority of the neighbourhood is classified as rough surfaces.
- *Unknown* - the cell and the majority of the cells in the neighbourhood are without surfaces at the given height level k.

The traversability analysis results in an occupancy map, such as shown in Figure 5.4, which for this project was sent further to a *Player MapDriver*[1] module. By functionality, this MapDriver module was not working in the regular *Player server-client* way, but was rather listening directly for incoming network messages - hence the occupancy map was also sent as a network message.

One interesting aspect with using many surfaces of different height levels within a map model is that objects with many height differences could also better be represented. The upper orange/lightgray area of Figure 5.4 (left) corresponds to stairs. A stair is normally seen as a pure occupied area. However, with a MLS map is this area merely classified as a rough area (orange/lightgray), even though the area is too rough to be considered as drivable, as seen in upper part of Figure 5.4 (right).

[1]Specifically written for the Learning System Lab., not part of the normal Player software.

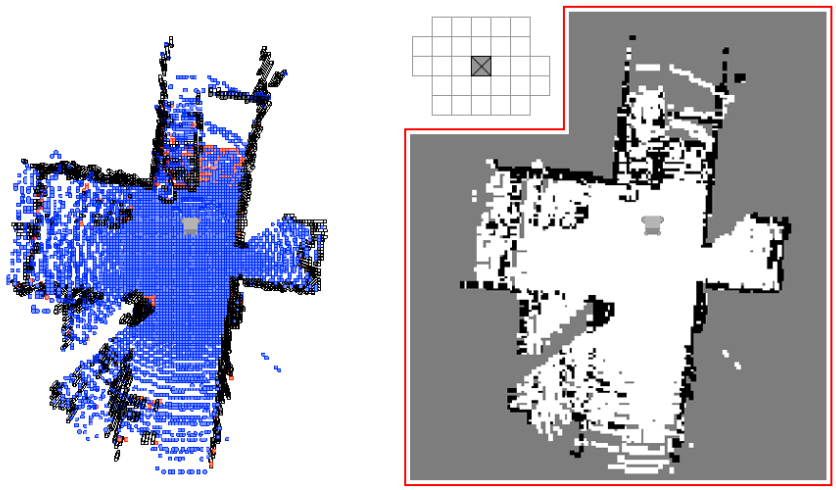

Figure 5.4: *Left* - an example of MLS map seen from above, *right* - corresponding occupancy map (without the neighbourhood in consideration). Upper right part of figures corresponds to a stair, which is by MLS maps only considered as an rough area but is rightly classified as an occupied area in the occupancy map.

5.2.1 Neighbourhood

The traversability analysis is considering the neighbourhood during the calculation of the occupancy map, but the *neighbourhood concept* is not a clean-cut when considered from the quadtrees point of view. One kind of neighbourhood is already present in sense of the relationship within the tree structure - all subnodes of a node are by default also neighbours with each other. As a result all neighbours are easily accessed through the common parent node, but this neighbourhood is also asymmetric. If a perfectly symmetric neighbourhood is to favour, this could be achieved to the cost of further travelling up and down the tree, shown in Figure 5.5, where the traversed depth could in worst case go all the way from leaf-level to root-level.

Rather than complicate the implementation by applying the neighbourhood concept on the tree structure, the concept was applied on occupancy map afterwards the basic traversability had been extracted. The resulting occupancy map is a 2D gridmap thus the neighbours of a cell c_{ij} within the map could with ease be accessed by index, $c_{(i\pm1)(j\pm1)}$.

X-neighbourhood

When applying the neighbourhood concept on a 2D occupancy gridmap, a greater neighbourhood could likewise easily be accessed. Because of that, a

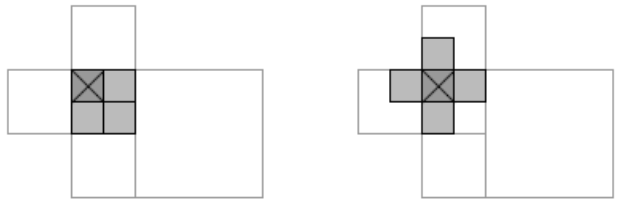

Figure 5.5: The neighbourhood of the cell marked by *X*. *Right* - the neighbourhood given by the rest of the sub-nodes within a node of the tree structure, *left* - a better more symmetric neighbourhood.

couple of different neighbourhood sizes were tested. By fitting a *kernel* over the neighbourhood was this tests done, where the *4-neighbourhood* and *8-neighbourhood* was tested. Result together with tested kernel is shown in Figure 5.6.

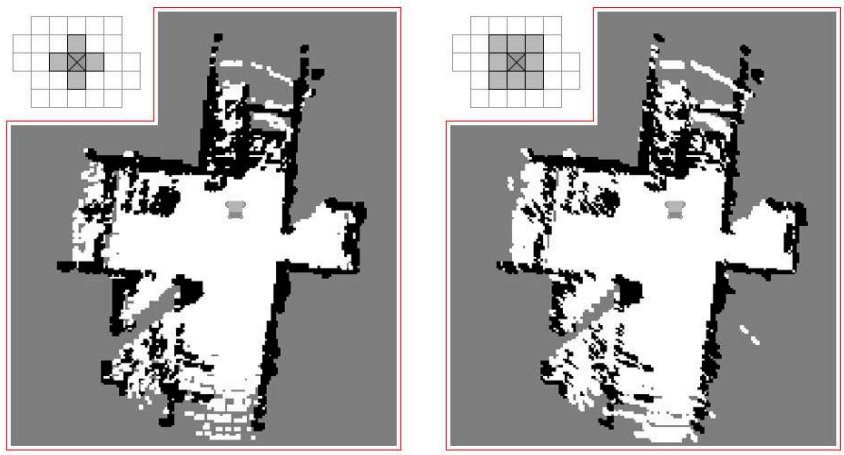

Figure 5.6: *Left* - occupancy map as a result of considering the 4-neighbourhood, *right* - occupancy map as a result of considering the 8-neighbourhood. In both tests was the same MLS map used as seen in Figure 5.4 (right), and the used kernel is illustrated in upper left corner of each image.

Since the use of the 8-neighbourhood kernel resulted in the *smoothest* occupancy map was that the kernel that was implemented and permanently assigned for this project.

5.3 Dynamic surfaces

The objective of this project was to implement an application that could handle navigation within a broad range of environments. The majority of the environments are though uncontrolled and changing, e.g. crowded with people, close to traffic, etc.; hence a way to handle changes within the environment was needed. To address this problem, all calculated surfaces were considered as dynamic - new points that fall within a previous calculated surface could change the structure of the surface. This must however be done with care, since faulty made changes could escalate and in worst case corrupt the whole map model.

In difference from similar projects, [28], surfaces were in this case updated every time points were falling within a surface, and all points were considered even though points were already within the Gaussian of a surface. Over time this could reflect changes in the environment and error surfaces (surfaces as result of false points, outliers and moving objects in the environment) could automatically be corrected. An example of this is shown in Figure 5.7, where a previous calculated surfaces (illustrated as the entire shaded gray area) could change the mean and variance if new points fall within only one half of the original Gaussian (as illustrated with the lower darkgray area).

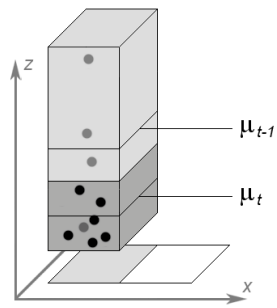

Figure 5.7: The structure of a surface represented by a Gaussian would change if new points only fall within a smaller interval covered by the original Gaussian.

5.4 Implementation details

The whole implementation of the MLS maps was divided into two levels, as seen in Figure 5.8. The MLS Map class use one object of the QuadTree class - which together works as the whole map model. The MLS Map class also creates a new object of the OccMap class every time a point-cloud is added to the map model. The functionality of the OccMap class is to represent a corresponding 2D occupancy map of the map model at a certain height level,

and which also had the functionality of sending the occupancy map further over a TCP connection to the *Player MapDriver module*.

The reason for dividing the whole MLS map into several classes is to provide for portability in case of further work of this project. As long as the methods and the functionality underneath are the same, other type of data/tree structures could been used for representing the map model.

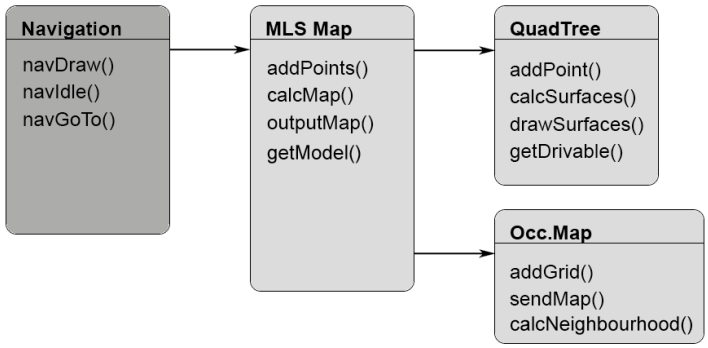

Figure 5.8: Class diagram of the mapping part of the implementation.

The rendering of the map model was implemented such that it was done on the leaf level of the tree structure for this project. A surface was represented by a flat horizontal square at the mean of the surface and with the same resolution as the leaf of the tree. Since the tree was split directly down to the leaf level, all nodes of the tree with points within were rendered in this case. The result of this was that leafs with few points within also were rendered (even though obviously cased by error points or outliers). With the traditional approach of collecting a certain amount of points before splitting the nodes, the rendering could either be prohibited or done on a higher level in case the leaf level was not reached.

Chapter 6
Experiments

This chapter will cover experiments done to test the application. Both an indoor and an outdoor experiment were done for diversity and to test the mapping ability together with scan matching in different environments. The locations used for those experiments together with a short motivation are presented throughout each section of this chapter, and where each section is summarized with the results.

6.1 The hallway just outside the laboratory

The first experiment was done in the hallway just outside the AASS Laboratory in use. For this experiment the experimental area was a flat indoor environment, where 2D scans would normally provided sufficient information for a navigation task. However, the reason for using this area as experimental area was that the hallway is also containing staircases and an overhanging walkpass, as well as cluttered objects like benches and tables, as can be seen in Figure 6.1 and Figure 4.4, and which could be very challenging to represent within an map model.

Figure 6.1: Different angles of the hallway outside the AASS laboratory.

The scans were acquired in the *stop-scan-go* fashion as described in Chapter 3.1.1, with a distance of $2 - 3$m in between. The total series for the experiment consist of 27 scans and where the scans consist of around 22000 points each. *Point-to-point* and *feature-to-feature* registration was tested during this experiment since the validation in Chapter 4.4.1 implied that *point-to-feature* registration gave weak results in comparison to the original point-to-point approach. For comparison and to show that accumulating errors within the odometry was present even throughout this experiment, the result of mapping the point-clouds without any registration is shown in Figure 6.2, where the figure assumption perfect conditions should containing straight parallel lines rather than *curved* lines.

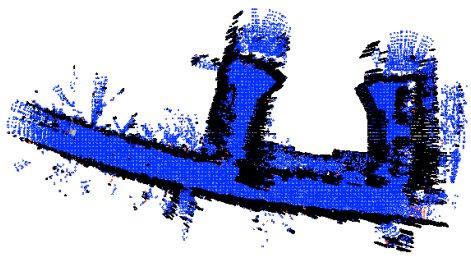

Figure 6.2: Result of mapping the hallway data series without any registration (top-view image).

6.1.1 Results

The result of scan match the point-clouds with point-to-point registration is seen in Figure 6.3, which also was the approach that gave the best result in sense of re-shape the data series into an straight hallway. Since traversability analysis together with MLS maps has been covered within the scope of this project, the resulting occupancy map is also shown in Figure 6.3.

The final global rotation matrix and translation vector of the point-to-point registration is shown in Table 6.1.

Rotation matrix (R)		
0.937545	-0.288614	-0.194194
0.317939	0.937463	0.141696
0.141155	-0.194588	0.970675

Translation vector (t)		
-6.25219	-5.5898	-5.02241

Table 6.1: The final global rotation matrix and translation vector as result of point-to-point registration.

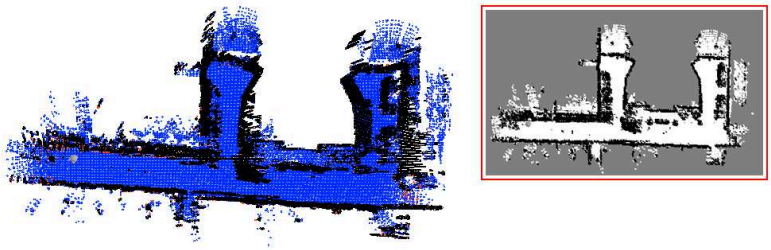

Figure 6.3: Result of mapping the hallway data series after point-to-point registration. *Right* - top-view image of the map model, *left* - corresponding occupancy map.

The result of feature-to-feature registration is seen in Figure 6.4, where the hallway was not re-shape (in this case) into the same original shape as with the point-to-point approach, even though the result was adequate compared to mapping without any registration (Figure 6.2).

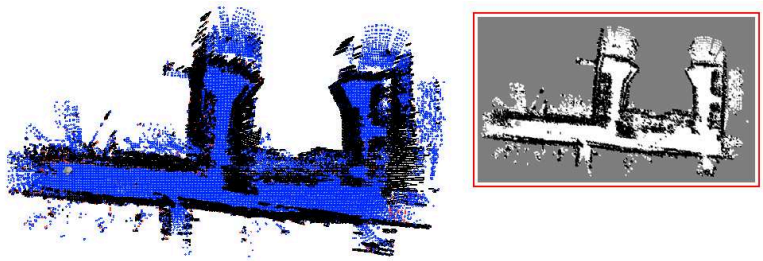

Figure 6.4: Result of mapping the hallway data series after feature-to-feature registration. *Right* - top-view image of the map model, *left* - corresponding occupancy map.

The final global rotation matrix and translation vector of feature-to-feature registration is shown in Table 6.2.

The scan matching managed to re-shape the data series and thus a map model could be built similar to a representation of a straight hallway. However, the scan matching was not done without great effort. Halfway the travelled distance the hallway is getting narrowed, as can be seen in lower middle part of the map images in Figure 6.3 and 6.4, caused by an staircase. The scan obtained right after passing the staircase was difficult to match since it was matched against points that was representing an area that became *hidden* behind the same staircase.

Rotation matrix (R)		
0.955767	-0.260787	-0.136012
0.278499	0.951135	0.133346
0.094591	-0.165327	0.981692

Translation vector (t)		
-4.8638	-1.56498	-3.61238

Table 6.2: The final global rotation matrix and translation vector as result of feature-to-feature registration.

Bent in horizontal plane

As a negative side-effect of the scan matching, the data series was bent upwards in horizontal plane. The bent effect can be seen as the high translation in z in Table 6.1 and Table 6.2, but to illustrate this further, a 3D view of the map model is shown in Figure 6.5. Where the bent effect is most severe at the end of the traverse, and which could be seen in the rightmost part of the figure.

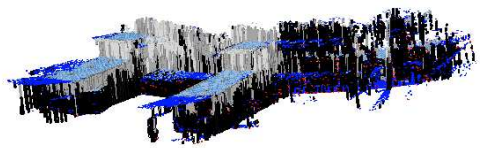

Figure 6.5: 3D view of the map model, where it could be seen that the data series is bent horizontally after registration.

6.2 A short trip on campus

Unfortunately the campus of Örebro University is a very flat geographical place. Thus it was difficult to find a good outdoor environment with height differences for testing the application. However, after some searching a good spot was found just outside the Musical building, where height differences has resulting in a man-made slope in front of the building, as shown in Figure 6.6. Mapping an outdoor environment was very challenging with this system setup. Since the robot's odometry was only given in 2D, the rotation around all three axes and the position in 3D could only be estimated as the result of scan matching.

The total series for the experiment consist of 27 scans, obtained with with a distance of $2 - 3$m in between, and where the scans consist of around 8400 points each. Because of the low number of points in each scan and the challenge in mapping an outdoor environment, only *point-to-point* registration was tested during this experiment. Also, for comparison and to illustrate that there was some problems along the traversed way, especially at the turning points,

Figure 6.6: Different angles of the front of the Music building at campus of Örebro Univeristy.

the result of mapping the data series without any registration is shown in Figure 6.7, where the figure assumption perfect conditions should containing a solid straight line. Within all images of the data series shown throughout this section, a point pattern along the traversed way will be seen. This pattern corresponds to a human operator that for safety reasons followed Alfred during the experiment.

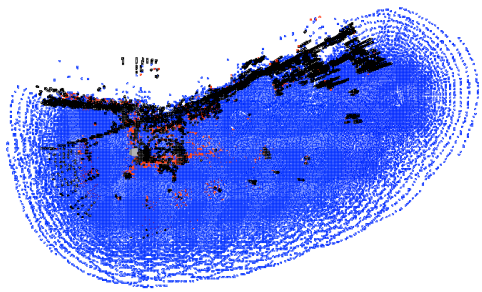

Figure 6.7: Result of mapping the outdoor data series without any registration (top-view image).

6.2.1 Results

The result of register the campus outdoor data series is seen in Figure 6.8, where an top-view image (left) of the map model is seen together with corresponding occupancy map (right).

The final global rotation matrix and translation vector of scan matching the outdoor data series is shown in Table 6.3.

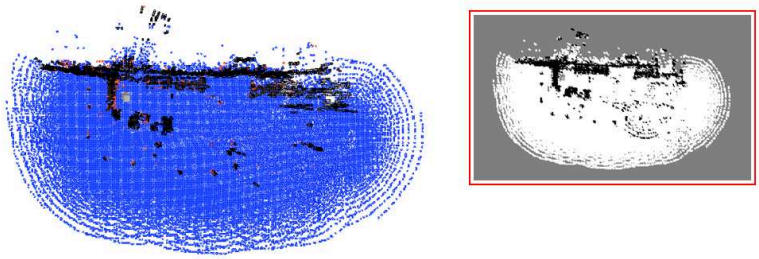

Figure 6.8: Result of mapping the campus outdoor data series after scan matching. *Right* - top-view image of the map model, *left* - corresponding occupancy map.

Rotation matrix (R)		
0.957338	0.288728	0.011837
-0.288970	0.956453	0.041145
0.000558	-0.042810	0.999083

Translation vector (t)		
-2.85147	1.20569	0.790796

Table 6.3: The final global rotation matrix and translation vector after scan matching.

Within this experiment, there was some truly problematic turning points along the traverse - which can be seen by the high *yaw* angle in the final global rotation matrix in Table 6.3. Translated into degrees, this angle corresponds to an final heading error of 45°. Nonetheless, to show that height differences was present throughout this outdoor experiment, a 3D view of the map model is shown in Figure 6.9, where the robot could be seen standing on higher ground.

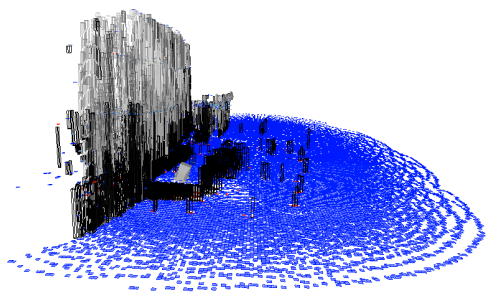

Figure 6.9: 3D view of the map model, there it could be seen that height differences was present during the experiment.

Chapter 7
Conclusions and future work

7.1 Summary

The main objective of this project was to implement an appropriate mapping method that could interpret provided 3D range scans. This objective has been achieved through the implementation of memory efficient *multi-level surface* (MLS) maps. Such MLS maps can be used to represent many height levels within an elevation map and thus get a rough 3D representation of the environment, without the complexity found within full 3D maps. Throughout this project, it has also been shown how a grid based map could be represented with a tree structure, and how the trees recursive structure is well suited for problems of searching character. Translation between MLS map and occupancy map was another important aspect of this project, where the traversability over a neighbourhood of a cell was considered together with height differences in between the same neighbouring cells.

Secondary objective of this project was to secure the reliability within provided sensor data. For this purpose, the *iterative closest point* (ICP) algorithm was implemented. The aim of the algorithm was to find the geometrical alignment between two provided 3D scans. As a result, keep the map model consistent and compensate for errors in the robots self-location. It has also been shown how a *kd*-tree structure was used for improving the nearest neighbour search of the ICP algorithm and how the depth of the tree structure was used to locate and discarding neighbouring point-pairs with high distance in between.

In order to get feedback and to control the application with point-and-click commands, a graphical human-robot interface was also implemented within the scope of this project. This interface was implemented with the use of OpenGL framework, and where the focus was on providing different view-points of the map representation. Beside the primary objectives, the connection between used techniques provided by *Player* client-server software and implemented techniques, became another important aspect of this project. Hence, a strict hierarchy was used during the implementation to avoid nested dependencies and to

provide portability for further work. Though remains yet to implementing the entire implementation done for this project as a self-standing Player module.

7.2 Future work

Both the experiment in Chapter 6 and the validation in Chapter 4.4 implies that the data series was bent in horizontal plane as an negative side-effect of the registration. Hence, the cause of this effect and the elimination of the same must be of highest priority on the list of further work for this project. A better self-localization could be a solution to this problem. The robot's odometry is provided in 2D, while the goal of this project was to build and maintain 3D representation of the environment. Therefore, additional devices so that the odometry given as the robots position in 3D and the rotation around all three axis are certainly of significance for further investigation.

The aim of this project was to provide middleware functionality required for carrying out a complete navigation task, therefore remains still to test the total navigation quality of the application. Naturally, different middleware functionalities could also be of interest, especially other combinations of scan matching algorithms together with the MLS maps concept.

7.2.1 Landmark-based localization in addition

To further improve the mapping, a landmark-based localization method could be used in addition to the registration, which in this case would add further certainty within the map model. For this system setup this would mean to extract distinct landmarks from the 3D sensor data. Thus correct the robots position and get a better initial alignment for the registration.

7.2.2 Loop closing

Another aspect that is not covered within this project is *loop closure*, [6, 7]; detect when a robot returns to a previously mapped region or re-observe previously seen landmarks after a large traverse. Closing a long traversed loop could be used as a good *validation* of the applications ability to estimate the global position. Because even smaller errors in the scan matching algorithm or in the localization method, usually prevents the application from recognize (or return to) a previous visited region.

Appendix A

This appendix shows the Player configuration file used for this project:

```
driver (
   name "sicklms200"
   provides ["6665:laser:0"]
   port "/dev/ttySICK"
   pose [0 0 0]
   resolution 100
   range_res 1
   connect_rate [38400 9600 50000]
   transfer_rate 500000
   invert 1
   alwayson 1
)

driver (
   name "amtecM5"
   provides ["6665:actarray:0"]
   connstring "RS232:5,9600"
   module_count 1
   directions "1"
   offsets "0"
   normalize_angles "0"
   module_ids "11"
   debug_level 0
)

driver (
   name "vmc"
   plugin "../drivers/vmc/lib/libvmc.so"
   provides ["6665:position2d:0"]
   alwayson 1
   interpolate 0
   debug_on 0
)


driver (
   name   "utm30lx"
   provides ["6665:laser:1"]
   plugin "../drivers/hokuyo/player-driver/laser/libutm30lx.so"
   get_intensities 1
   portopts "type=serial,device=/dev/ttyACM0,timeout=1"
   alwayson 1
   invert 1
   pose [0 0 0]
)
```

```
driver (
   name "IntegratedLaser"
   requires ["6665:laser:1"]
   provides ["6665:laser:2"]
   plugin "../drivers/LaserDriver/lib/libintlaser.so"
   invert 0
   alwayson 1
)

driver (
   name "vfh"
   provides ["6665:position2d:1"]
   requires ["6665:position2d:0" "6665:laser:2"]
   safety_dist_0ms 0.15
   free_space_cutoff_0ms 1000000.0
   distance_epsilon 0.5
   angle_epsilon 10
   max_turnrate_0ms 15
)

driver (
   name "MapDriver"
   plugin "../drivers/MapDriver/lib/libmapdriver.so"
   provides ["6665:map:0"]
   port 12345
   resolution 0.1
   alwayson 0
)

driver (
   name "wavefront"
   provides ["6665:planner:0"]
   requires ["output::6665:position2d:1" "input::6665:position2d:0" "6665:map:0"]
   safety_dist 0.15
   alwayson 0
)
```

Bibliography

[1] The player project. Online: http://playerstage.sourceforge.net/ - visited in June 2009.

[2] P.J. Besl and N.D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[3] D. Chetverikov, D. Svirko, D. Stepanov, and Pavel Krsek. The trimmed iterative closest point algorithm. In *In International Conference on Pattern Recognition*, pages 545–548, 2002.

[4] E. D. Denman and A. N. Beavers Jr. The matrix sign function and computations in systems. *Appl.Math.Comput.*, 2:63–94, 1976.

[5] T. Duckett, S. Marsland, and J. Shapiro. Learning globally consistent maps by relaxation. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3841–3846, 2000.

[6] H. Durrant-Whyte and T. Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms. robotics and automation magazine. *IEEE Robotics and Automation Magazine*, 2, 2006.

[7] H. Durrant-Whyte and T. Bailey. Simultaneous localisation and mapping (slam): Part ii state of the art. robotics and automation magazine. *IEEE Robotics and Automation Magazine*, 2, 2006.

[8] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[9] J. Folkesson, J. Leederkerken, R. Williams, A. Patrikalakis, and J. Leonard. A feature based navigation system for an autonomous underwater robot. In *6th International Conference on Field and Service Robotics - FSR 2007 Field and Service Robotics Springer Tracts in Advanced Robotics*, volume 42 of *Springer Tracts in Advanced Robotics*,

Chamonix France, 2007. Springer. I.: Computing Methodologies/I.2: ARTIFICIAL INTELLIGENCE/I.2.9: Robotics.

[10] M. O. Franz and H. A. Mallot. Biomimetic robot navigation. *Robotics and autonomous Systems*, 30:133–153, 2000.

[11] Christian Früh and Avideh Zakhor. 3d model generation for cities using aerial photographs and ground level laser scans. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:31, 2001.

[12] D. Hähnel, W. Burgard, and S. Thrun. Learning compact 3D models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems*, 2002. to appear.

[13] D. Hähnel, D. Schulz, and W. Burgard. Map building with mobile robots in populated environments. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[14] Martial Hebert. Active and passive range sensing for robotics. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '00)*, volume 1, pages 102 – 110, April 2000.

[15] B. K. P. Horn, H.M. Hilden, and S. Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society A*, 5(7):1127–1135, 1988.

[16] F. Launay, A. Ohya, and S. Yuta. Autonomous indoor mobile robot navigation by detecting fluorescent tubes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA01)*, 2001.

[17] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic Systems*, 18:249–275, 1994.

[18] M. Magnusson. *3D Scan Matching for Mobile Robots with Application to Mine Mapping*. Licentiate thesis, Örebro University, September 2006.

[19] A. Nüchter, K. Lingemann, and J. Hertzberg. 6d slam with cached kd-tree search. In *Robot Navigation*, 2007.

[20] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. Heuristic-based laser scan matching for outdoor 6d slam. In Ulrich Furbach, editor, *KI*, volume 3698 of *Lecture Notes in Computer Science*, pages 304–319. Springer, 2005.

[21] P. Patrick, T. Rudolph, and B. Wolfram. An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *Int. J. Rob. Res.*, 26(2):217–230, 2007.

[22] P. Pfaff, R. Kümmerle, D. Joho, C. Stachniss, R. Triebel, and W. Burgard. Navigation in combined outdoor and indoor environments using multi-level surface maps. In *Proc. of the Workshop on Safe Navigation in Open and Dynamic Environments at the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, San Diego, CA, USA, October 2007.

[23] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, June 2001.

[24] S. Se, D. Lowe, and J. Little. Vision-based mobile robot localization and mapping using scale-invariant features. In *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA*, pages 2051–2058, 2001.

[25] G. G. Slabaugh. Computing euler angles from a rotation matrix, 1999.

[26] Todor Stoyanov and Achim J. Lilienthal. Maximum likelihood point cloud acquisition from a rotating laser scanner on a moving platform. In *Proceedings of the IEEE International Conference on Advanced Robotics (ICAR)*, pages 108–113, June 22–26 2009.

[27] S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[28] R. Triebel, P. Pfaff, and W. Burgard. Multi level surface maps for outdoor terrain mapping and loop closing. In *IROS*, pages 2276–2282, 2006.

[29] O. Wulf, K. Oliver Arras, H. I. Christensen, and B. Wagner. 2d mapping of cluttered indoor environments by means of 3d perception. In *ICRA*, pages 4204–4209, 2004.

[30] O. Wulf, C. Brenneke, and B. A. Wagner. Colored 2d maps for robot navigation with 3d sensor data. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '04)*, Sendai, Japan, September 2004.