TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Tanel Kossas 204260IABM

Tactile Navigation Algorithm For Autonomous Underground Mining Robots

Master's Thesis

Supervisor: Mohamed Walid Remmas
MSc

Co-supervisor: Roza Gkliva
PhD

Co-supervisor: Asko Ristolainen
PhD

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Tanel Kossas 204260IABM

TEHISLIKEL TUNDLATEL PÕHINEV NAVIGATSIOONIALGORITM AUTONOOMSETELE MAA-ALUSTELE KAEVANDUSROBOTITELE

Magistritöö

Juhendaja: Mohamed Walid Remmas
MSc

Kaasjuhendaja: Roza Gkliva
PhD

Kaasjuhendaja: Asko Ristolainen
PhD

Tallinn 2023

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Tanel Kossas

09.05.2023

Abstract

State-of-the-art robot vision techniques have recently enabled outstanding results in the navigation of autonomous robotics. However, these are not suitable for use in caves and mining environments due to the large amount of dust and mud present, in addition to a dark and locally self-similar environment. This prohibits mining from being automated with the help of robots, which is a key goal for the EU-funded ROBOMINERS project. In order to perform navigation in caves and mines, using artificial whiskers as tactile sensors was explored in the thesis. In order to validate their performance for navigation, three algorithms were developed: a wall-follower, a navigation algorithm based on Theta* and an algorithm combining the two for increased navigating efficiency in unknown areas. During tests in both simulation and real-world experiments involving the RM3 robot, it was shown that tactile sensing worked well in complex environments in the case of wall-following, and also when aiding navigation, if a method to estimate pose was also present. In addition, the third algorithm (that combined navigation and wall-following) could improve the traversal time of an unknown map by between $26 - 43\%$ compared to regular navigation without wall-following. This thesis thus lays part of the groundwork for navigation in an autonomous mine as part of the ROBOMINERS project, utilising tactile navigation. In addition, the thesis opens up a plethora of potential research avenues, including optimisations to the simulator, the navigation algorithms, the locomotion system and the whisker configurations, as well as in unifying the robot with external systems.

The thesis is written in English and is 69 pages long, including 5 chapters, 47 figures and 14 tables.

Annotatsioon
Tehislikel tundlatel põhinev navigatsioonialgoritm autonoomsetele maa-alustele kaevandusrobotitele

Kaasaegsed masinnägemise tehnoloogiad on võimaldanud robotitel saavutada märkimisväärseid tulemusi autonoomsel liikumisel. Enamasti kasutavad antud tehnoloogiad kaameraid ja LiDAR-it. Samas on keskkondi, kuhu antud tehnoloogiad ei sobi. Näited sellistest keskkondadest on koopad ja kaevandused, kus kaamerate ja LiDAR-ite töö on häiritud tolmu, muda ja pimeduse tõttu. Lisaks on need keskkonnad tihti ka lokaalselt väga sarnased, st ühe koha eristamine teisest visuaalselt on raske.

Eelnevad asjaolud muudavad robotite abil kaevanduste automatiseerise keeruliseks. See on aga EL poolt rahastatava ROBOMINERS-i projekti üks olulisemaid eesmärke. Selleks, et võimaldada navigeerimist koobastes ja kaevandustes, uuriti antud töös kunstlike tundlate kasutamise efektiivsust taktiilsete sensoritena. Selleks arendati lisaks juhtimisrakendusele välja kolm erinevat algoritmi: seinajärgija, navigatsioonialgoritm Theta* baasil ning algoritm, mis ühendab kaks eelnevat algoritmi, et suurendada navigeerimise kiirust keskkonnas, mida pole kaardistatud.

Simulatsioonis ja RM3 robotiga läbi viidud katsetes oli näha, et taktiilne tajumine töötas ka keerulistes keskkondades seina järgimisel. Lisaks aitas taktiilne tajumine ka märkimisväärselt navigeerimisel, kui oli olemas väline viis roboti asendi määramiseks. Lisaks sellele kiirendas kolmas (st navigeerimist ja seina järgimist ühendav) algoritm navigatsiooni $26 - 43\%$ võrreldes tavalise Theta*-l põhineva navigatsioonialgoritmiga.

Antud töö loob seega osa alusest ROBOMINERS-i projekti raames loodavale autonoomsele kaevandusele, mille tarvis uuritakse hetkel taktiilsete sensorite kasutamist kaevandusrobotitel. Antud töö võimaldab järgnevat teadustööd, mis hõlmab loodud algoritmide optimiseerimist, süsteemide omavahelist ühendamist, simulatsiooni realismi edendamist ja taktiilsete sensorite asetuse optimiseerimist.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 69 leheküljel, 5 peatükki, 47 joonist, 14 tabelit.

List of Abbreviations and Terms

Widespread abbreviations and terms

A*	Informed search algorithm used for path planning
Any-angle path planning	Path planning algorithm which allows turns to have any angle, resulting in more optimal paths especially in open areas
Gazebo	Simulator software used for robots
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LiDAR	Light Detection And Ranging, or more accurately, Laser Imaging, Detection, and Ranging
Path planning	Problem of finding a valid (optimal) path from point A to point B; synonymous with pathfinding and motion planning
PID controller	Proportional-integral-derivative controller
PID controller tuning	Process of finding suitable gain values for the P, I and D parts of a controller
PID setpoint	Desired eventual input value for a PID controller
ROS 2	Robot Operating System 2
RRT	Rapidly-exploring Random Tree - an algorithm used for path planning
SLAM	Simultaneous Localisation and Mapping
Theta*	Informed search algorithm, which uses A* as a base to perform any-angle path planning

Definitions for the present thesis

Collision	Event of any component of the robot, including whiskers, coming into contact with an obstacle
Hard collision	Event of a non-whisker component of the robot coming into contact with an obstacle
Line-of-sight check	A check in a grid to see whether two points have any obstacles between a line drawn directly from one point to another

Navigation	Ability for a robot to arrive at a target destination without collisions using localisation and path planning
Pressure	Term used for the amount of deviation from a whisker's neutral position
RM3	ROBOMINER 3 robot, part of the ROBOMINER project
Whisker	Bio-inspired tactile sensor developed for the RM3 robot
Whisker array	Interface which has several whiskers in a row and can report current orientations of whiskers

Table of Contents

List of Figures

List of Tables

1. Introduction

In recent years state-of-the-art robot vision techniques have started to enable outstanding results in autonomous robotics, as outlined in [1]. One of the most outstanding areas of this is real-time autonomous driving. The state-of-the-art techniques mostly depend on cameras and LiDAR as sensor inputs. However, there are environments where visual sensors are greatly compromised [2]. Two such examples are underground mines and caves — environments that are often dusty, muddy and dark that are also locally very self-similar and at times could also be underwater. In addition, what compromises robot vision even further for these environments is also the inability to use GPS for localisation.

The ROBOMINERS project is an EU-funded project [3, 4] with the objective to create autonomous robotic miner prototypes to mine difficult to access mineral deposits. The project in turn addresses the second pillar of the EU Raw Materials initiative, which is to secure a supply of raw materials from sources within the EU. The ROBOMINERS project aims to change the current paradigm of mining that is currently mostly dominated by humans and conventional large machines. The environment it aims to create is fully artificial, with no human life support underground in the mines. This is seen as a better method economically, and also as a method to reduce the environmental impact of mining and increase safety for workers in the mining industry. [2]

In order to achieve efficient robotic miners, the robots need a method to sense their environment. This includes a way to localise themselves within a map and a way to avoid obstacles. As explained in [2], the current state-of-the-art methods to achieve these are insufficient for use in mines and caves. Thus, novel ways of localisation and sensing need to be developed in order to perform efficient navigation in these environments.

Due to biological inspiration showing great results in recent research, then one of the goals of the ROBOMINERS project was to also seek to use bio-inspiration for solving problems [2]. Thus one of the areas that is being studied for solving the issue of perception is tactile sensing, which is seen in many animals. As part of this, some previous research as part of this project has developed artificial whiskers. These are similar to the ones in mammals, but are a lot thicker in order to offer better performance in mining environments. Previous and current research as part of the project is also developing the use of tactile sensing for Simultaneous Localisation and Mapping (SLAM), in order to explore the potential of whiskers in mapping large environments.

The main focus of the present thesis was to validate the ability to perform navigation using the novel sensing method, and to also lay groundwork for combining the navigation with localisation methods (e.g. SLAM) in order to perform more complex functions. These complex functions could in the future include exploration of a mine, prospecting for ore densities, mining the ore with the best return etc. The goal for the thesis was to develop three navigation algorithms for one specific prototype robot in the project — RM3 — independent of any SLAM algorithms, and a control application for switching between algorithms and providing necessary input.

The first algorithm sought to implement wall-following — a task often used for initial validation of sensing methods — with only tactile sensing used. The main goal of the first algorithm was to validate the overall ability of basic navigation for the robot using the novel sensing method. In addition, the first algorithm could further be used to provide a way for the robot to navigate if localisation fails. In addition, wall-following could aid in speeding up exploration and navigation, by following obstacles instead of trailing back to an unoccupied spot.

The second algorithm sought to implement basic path planning for the robot, with whiskers being used for obstacle detection and building a map. Localisation itself was received from either simulation data or achieved by using an external camera to determine pose. As mentioned before, in the future this is planned to be integrated with a SLAM algorithm for real-world use. Thus the purpose of the algorithm was to move the robot to a given destination (provided by a human or another algorithm) and avoid any obstacles.

The third algorithm sought to implement a combination of the two: speeding up navigation by performing wall-following, especially when exploring unknown areas or navigating through them. Thus the algorithm intended to reach the target in a more optimal way by switching between using path planning and performing wall-following according to the perceived optimality of each algorithm in a given position.

The algorithms were developed in iterations, using both a simulator previously developed as part of the project and also three real-world experiments, in order to validate the effectiveness of the whiskers in real-world conditions and discover any potential issues affecting the tactile sensors. Due to the novel method of sensing and movement, no previous research on the optimal whisker placement for navigation existed. Thus, an additional necessity and a separate goal of the thesis was to evaluate, propose and test configurations of whisker placement on the robot.

To summarise, the contributions of this thesis are:

- development of:
  - a control application for switching between algorithms;
  - a wall-following algorithm;
  - a navigation algorithm based on Theta*;
  - a navigation algorithm combining wall-following and regular Theta* navigation.
- propositions of whisker configurations suitable for navigation.

The thesis is thus organised as follows. In the second chapter, the overall background of the thesis is explored. This includes a detailed explanation of the main aspects of the project, robot and its components. In addition, relevant research in the underlying areas is discussed and the relevant theory behind the solution is explained. In the third chapter, the methodology of developing the algorithms is outlined and the three developed algorithms are explained in detail, in order for them to be reproducible. In the fourth chapter, results of tests are brought out and discussed. In the fifth chapter, a summary of the thesis is provided.

2. Background

In the following chapter the background of the project, related state-of-the-art research and the theoretical background for the underlying thesis are described.

2.1 ROBOMINERS project

The ROBOMINERS project is an EU-funded project [3, 4] to develop a bio-inspired, modular and reconfigurable robot for mining small deposits that are hard to access. The end goal of the project is to create an amphibious robot that would be capable of mining in environments that would otherwise be inaccessible or economically unfeasible. These environments can range from abandoned and operating mines that still have remaining resources, to environments that are very deep underground, inaccessible to humans or hazardous. The long-term concept of the project is outlined in Figure 1. [2]

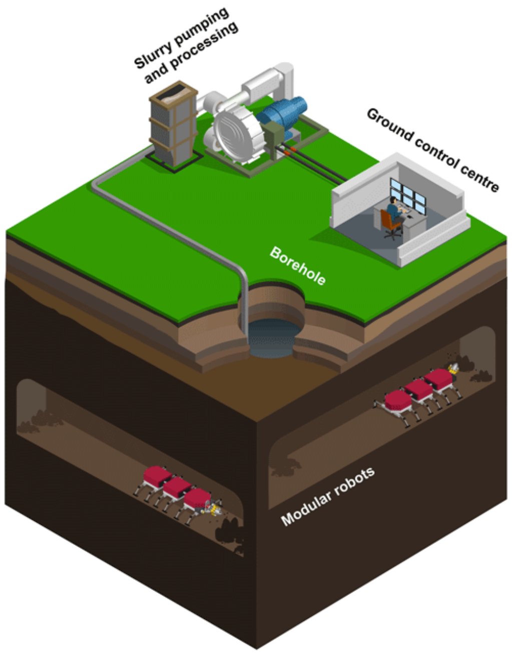


Figure 1. *Main elements of the ROBOMINERS long-term concept [2].*

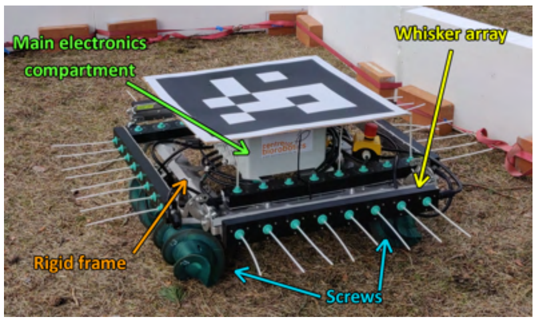
The current conventional methods of perception for localisation and navigation (i.e. GPS, LiDAR) do not work well in mining environments. In addition, the project decided to take biological inspiration as a guideline due to a lot of focus on this area in research in the last few years, and due to many bio-inspired robot designs showing effectiveness in locomotion, energy efficiency and perception. For these reasons, part of the project is to study the use of tactile sensing as a method of perception for localisation and navigation. [2]

2.2 The RM3 robot

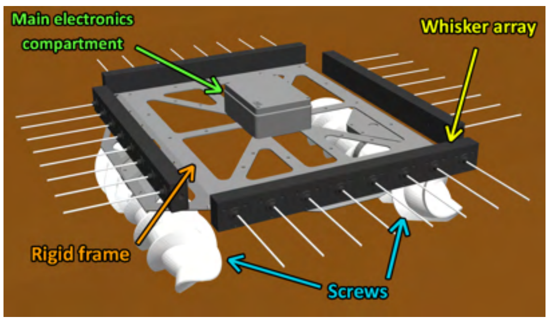
The RM3 (see Figure 2) is a prototype robot that is part of the ROBOMINERS project. The robot is designed to be used as a testing platform for various technologies in order to pave way towards a functional prototype as part of the ROBOMINERS project. It is envisioned to be able to use the robot underground and underwater in a flooded environment, and it is also required to work in slurry and on dry land. For tactile sensing, the robot uses bio-inspired *whiskers* analogous to ones used by mammals in the real world and described in more detail in subsection 2.2.2. The whiskers are placed on each of the 4 sides of the robot in arrays of 8 whiskers each, with different angle configurations possible relative to the robot.

The frame of the robot is 0.79 meters long and 0.74 meters wide, with the whiskers having variable length depending on the test (but the length usually being 0.2-0.3 meters long). In addition to the whisker arrays — developed by TalTech Centre for Biorobotics — the sensors of the robot also include a PI48 IMU, a secondary IMU (BNO080), an MCP9808 temperature sensor and a custom water ingress detector (for detecting if water has entered the main elecronics compartment). More sensors could be added in the future for improving localisation or data collection, e.g. for measuring resource densities.

The robot consists of four main parts: a rigid frame (where other modules can be mounted), the main electronics compartment and two locomotion modules, i.e. screws (described in more detail in subsection 2.2.1).



(a) *RM3 in the real-world.*



(b) *RM3 in the Gazebo simulator.*

Figure 2. *RM3 robot with a side whisker configuration.*

The RM3 robot's software is built onto the ROS 2 (Robot Operating System 2) platform. As ROS 2 is open-source, multi-platform and widely used in both research and industry, then using this platform ensures that the developments made for this prototype could also be applied in the future when the project develops further into actual use. In addition, the platform enables easy use of testing platforms like Gazebo and the ease of re-using most of the program code for both simulation and real-world use. [5]

2.2.1 Locomotion

Modern mining machines usually use tracks or large wheels for locomotion. Due to the small size of the robot, tracks or large wheels were not feasible due to loss in traction and large mechanical complexity. However, there has been research into using Archimedean screws (see Figure 3) for locomotion that has shown good traction in various environments. As screws also have few moving parts and low control requirements, the project ended up using Archimedean screws in their prototype [6].

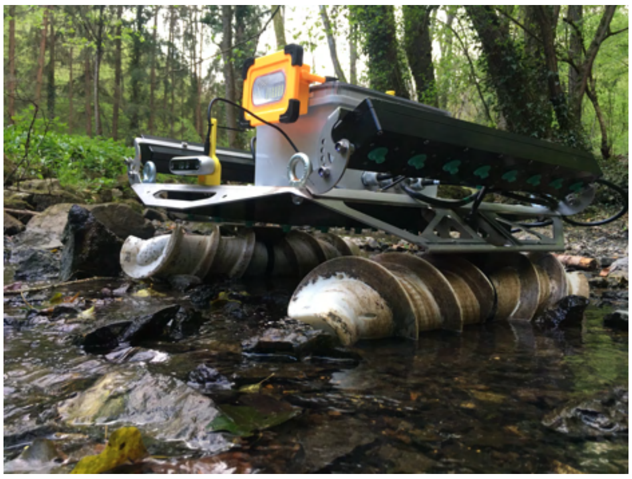


Figure 3. *Screws mounted onto the robot. (TalTech Centre for Biorobotics)*

The locomotion modules both include two counteracting Archimedean screws, individually actuated by two separate motors (see Figure 4). Each motor module has a hall encoder for position feedback, and an INA219 sensor for monitoring voltage and current. The two sides are interlinked with a centre piece, that is used to be mechanically fastened to any kind of leg joint. The aluminum hull of the screw modules is waterproofed using O-rings on both end caps and on the centre piece.

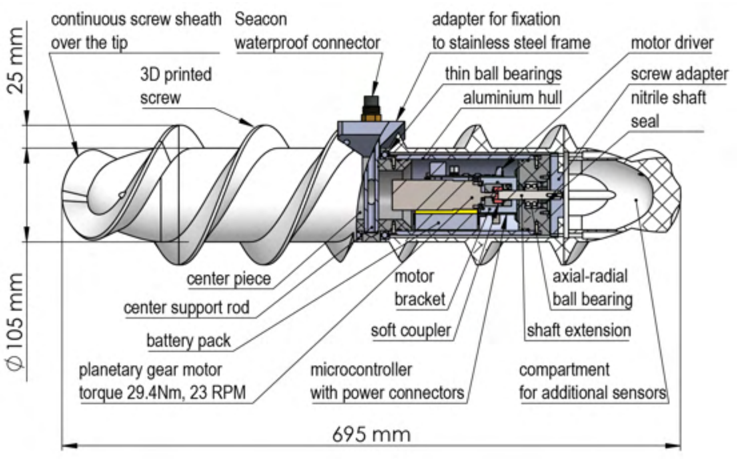

25 mm
continuous screw sheath over the tip
Seacon waterproof connector
adapter for fixation to stainless steel frame
motor driver
3D printed screw
thin ball bearings
aluminium hull
screw adapter
nitrile shaft seal
center piece
center support rod
battery pack
planetary gear motor torque 29.4Nm, 23 RPM
motor bracket
soft coupler
microcontroller with power connectors
axial-radial ball bearing
shaft extension
compartment for additional sensors
∅ 105 mm
695 mm

Figure 4. *Cutout illustration of the construction of the screw mechanism.*

As there are two counteracting helices on each actuator, each degree of freedom on the x-y plane can be separately controlled. Thus the robot can move in each direction separately, which simplifies movement algorithms. Due to the geometry of screws, on different surfaces the speed and optimal direction for movement is different. This should also be considered when seeking to create an optimal navigation algorithm for RM3.

2.2.2 Tactile whisker sensors

Due to the previously mentioned difficulties in operating sensors that are traditionally used for machine navigation, long-reaching tactile sensors are being experimented with, called whiskers due to their resemblance to real-life whiskers on animals.

The whiskers consist of a 3D Hall effect sensor (Infineon TLV493D-A1B6), a magnet mounted on the base of the whisker, a flexible silicone membrane for the whisker and the whisker itself (see Figure 5). The silicone membrane holds the whisker up straight (perpendicular to the array) when the whisker is not in contact with any object and provides a small force to return the whisker back to its resting state when in contact with an object. [7]

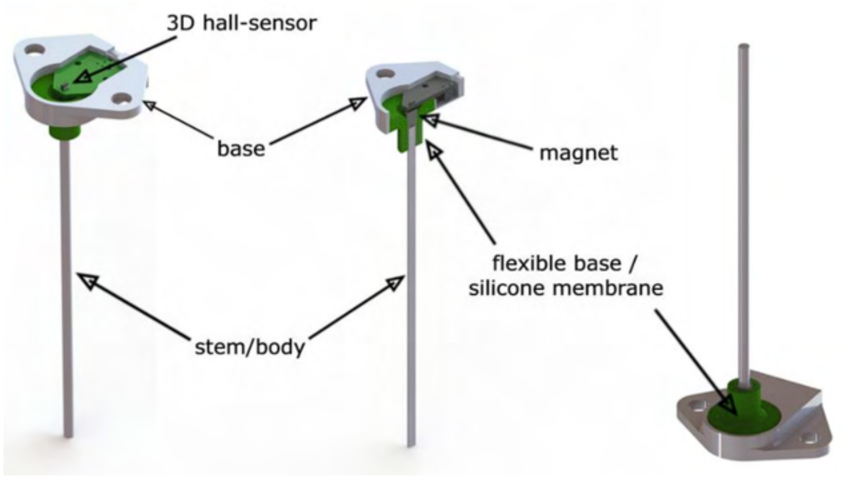


Figure 5. *Whisker sensor components [7].*

The whiskers are configurable: the length of the whiskers can be changed and a suitable material chosen depending on the behaviour that is needed from the whisker [7]. In [7], the whiskers were used in a configuration where they were only in the bottom of the robot, in an 8-by-8 configuration (see Figure 6). The purpose of [7] was to use the given whisker arrays for Simultaneous Localisation and Mapping (SLAM) based on the ground interactions.

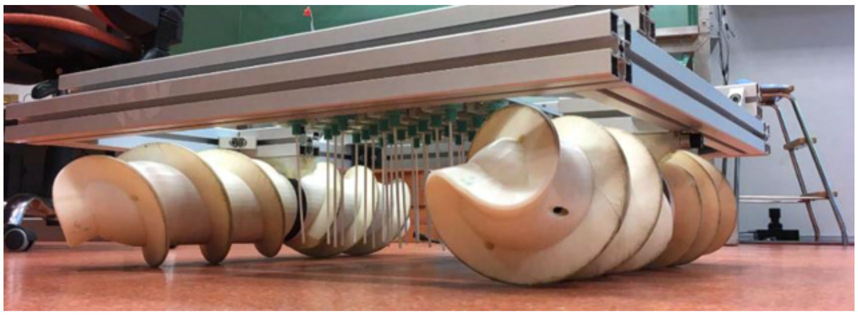
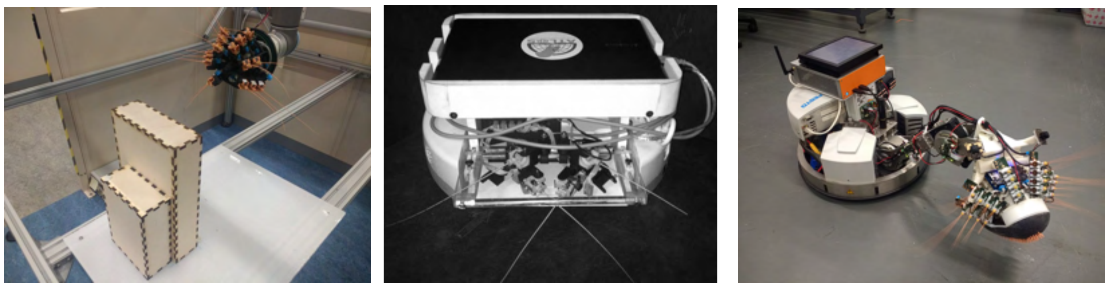


Figure 6. *Whisker configuration for ground-based SLAM [7].*

2.3 Whisker-based sensing

Bio-inspired whiskers have inspired solutions to many different issues. One area they have been used for is flow sensing in otherwise challenging environments, both in air [8] and underwater environments [9]. In both [8] and [9] they showed good performance for a relatively low cost. In addition, both studies also brought out the whisker's size, shape and material as something to be tuned depending on the desired behaviour, as they also contribute greatly to the ability to sense the flow of air and water.

Another area with recent research efforts is object mapping classification with whiskers on a robotic arm. A 2019 paper [10] explored using 18 sparsely located tactile sensors on a rotatable robotic arm for object classification (see Figure 7a). Although the purpose of the present thesis is not to identify objects, [10] showed that basic objects can be fairly easily identified through the use of whiskers. A 2022 paper [11] expanded on this, using just five whiskers on a robotic arm to identify much smaller objects that requiring more precision. Thus using whiskers to map and navigate cave environments — demanding much less regarding precision — shows promise.



(a) *Use of whiskers to classify objects in [10].* (b) *Robot used for tactile SLAM in [12].* (c) *WhiskEye platform used in [13].*

Figure 7. *Whisker-based sensing examples.*

A 2012 paper [12] mapped an indoors environment with SLAM using a robot with no other sensors for besides whiskers (see Figure 7b). The robot functioned indoors around straight-edged objects. The paper determined that using tactile information for SLAM is sufficient for mapping small arenas, as timing information between contacts and prior knowledge about edges can be used. However, [12] determined that it could not compete with SLAM systems using visual or laser sensors. Since 2012 lots of progress has been made to improve tactile sensor-based SLAM however — as can be seen in previous and upcoming examples in this thesis — so expanding it into environments that are both larger and less defined seems possible.

Whiskers have also been used to augment camera sensors, as in the WhiskEye robot platform described in [13]. The WhiskEye platform (see Figure 7c) uses 4 whisker arrays with 6 whiskers each on the rotatable head. The whisker sensors themselves used a 2-axis hall effect sensor to detect whisker deflections, similar to our current robot. [13] found that in their test fusing whisker sensors (camera optimised for long-range and whiskers optimised for short-range) reduced the energy required for localisation by nearly ten times.

2.4 Wall-following algorithms

Wall following is one of the most researched areas in robotics, as it contains a lot of challenges that are also necessary to solve for more advanced use cases of robots. Thus a good start for the novel whisker sensors and the screws is to see how it is able to handle wall-following. Thus it is beneficial to explore wall-following algorithms that have been proposed for various robots.

Wall-following has been successfully done using various sensor combinations. One approach has been to use sonar data and LiDAR and combine them using a fuzzy Kalman filter [14]. One earlier approach was also to use a single ultrasonic sensor to measure the angle of the closest reflecting point and moving in a direction perpendicular to that [15]. More recently, using cameras along with AI-based methods have also become more popular. For example, in 2020 a vision-based framework [16] was proposed for cleaning walls during the COVID-19 pandemic. The framework used deep learning to detect, classify and segment wall and floor surfaces that required cleaning.

In addition, metaheuristic optimisation has also been used for tuning parameters in wall-following algorithms. In 2006, evolutionary learning was used in [17] to generate the values for a fuzzy controller for a wall-following robot using an ultrasound sensor for localisation. In that case as well, navigation could be performed with only the ultrasound sensors of the robot, without any IMU data needed [17]. The algorithm showed relatively good performance in simulation and would likely also show promising results for the RM3 in good conditions, though is currently limited in the realism and speed of the robot's simulator and thus also would likely suffer in real tests unless trained in a wide variety of scenarios.

In 2017, an algorithm [18] was invented for a two-wheeled robot using a laser range finder which scanned different angles in front of and on the side of the robot. The distance measurement points were then used with the least squares method to produce a virtual wall. This in turn simplified the problem of wall following to just adjusting the robot's angle to that of the virtual wall (see Figure 8). In addition, in the resulting algorithm, the velocity of the robot was adjusted based on the distance to the wall to increase obstacle avoidance capabilities, as the sensor has a fixed interval of capturing data. This robot in [18] is in a sense quite similar to the requirements of the algorithm in this thesis, though using different locomotion and sensing tools. Similar to our platform, no easy way to obtain odometry data exists, so the algorithm does not use odometry, and only uses a single sensor at that. What's different though is the much more demanding environment for the RM3, in which straight walls rarely exist. However, a similar case of virtual wall construction could be used.

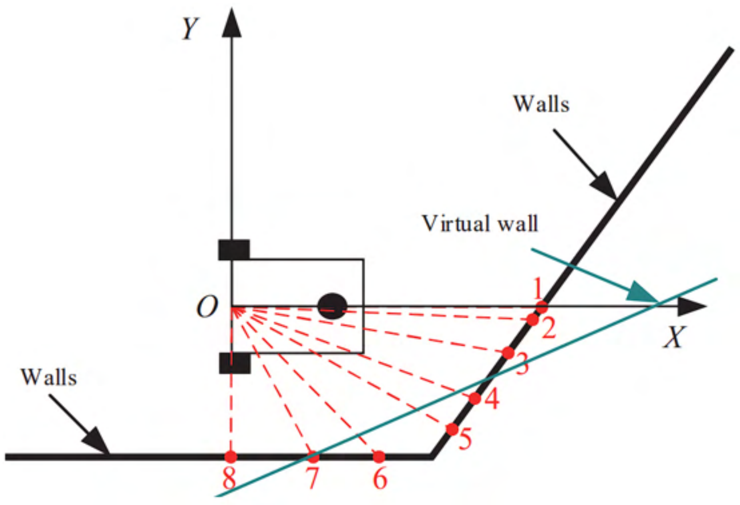
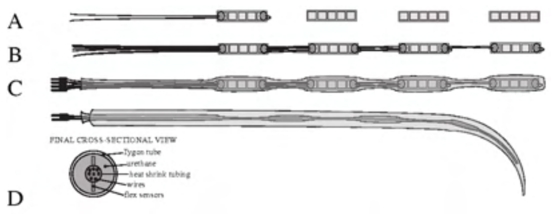


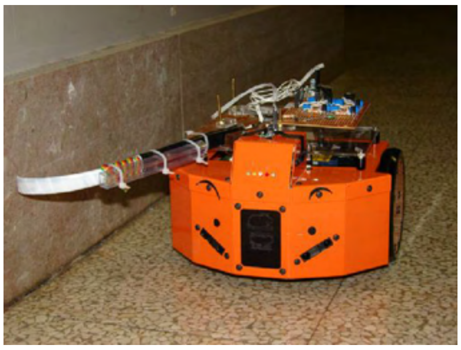
Figure 8. *Use of laser range finder for wall-following [18].*

A 2007 paper explored an artificial antenna for wall following — inspired by the sensory system of cockroaches — with relatively good success. In this case a unicycle-like robot was used. Instead of having many whiskers and one sensor near the body of the robot, they instead had just one antenna, but had flex sensors spread every 2.5cm apart (see Figure 9). The antenna being further ahead than the robot provides leeway for avoiding collisions and thus the robot can be made to be relatively high-speed. The given sensors are however more difficult to manufacture, though potentially less are needed. The processing was quite simple, using just a PD controller. The robot could achieve a 0.5 m/s speed, used wheels and operated indoors on flat ground, with strictly straight walls. [19]

The author considers the closest current research related to the topic of this thesis to



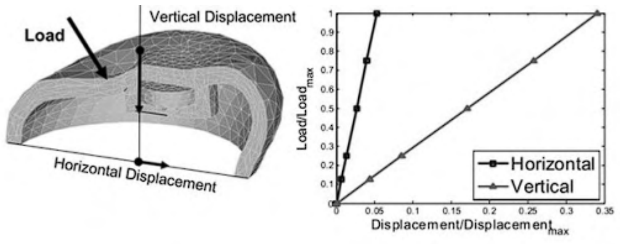
(a) *Build process of the antenna.*



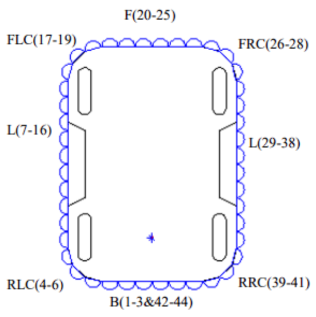
(b) *Wall-following robot.*

Figure 9. *Cockroach-inspired antenna sensor on a robot [19].*

be a thesis from 2013 [20]. The sensors it used were slightly different to whiskers (see Figure 10a), however in terms of function they could be considered similar as they provide information on pressure on the robot's sides (see Figure 10b). The robot was only simulated in Matlab, so the thesis did not involve any real-world experiments. This limits its use for this thesis especially due to a real-world cave environment differing greatly from a simulated indoors environment with no obstacles.



(a) *Used sensor displacement model.*



(b) *Simulated robot model.*

Figure 10. *Robot and sensor used in related wall-following MA thesis [20].*

In addition, the author of the present thesis considers the method of navigation chosen in [20] questionable, as no existing optimal pathfinding algorithms (e.g. A*, Dijkstra) were used. Instead, Gaussian Mixture Models (GMM) were used for an estimation of state for each sensor, after which a relatively complex strategy was created for both pathfinding and wall following, which was built on top of the potentially faulty estimation. In addition, the means of acquiring odometry information outside of simulation for the robot's present or target position was not considered. This means that there's significant potential for improvement using different methods and several issues that need to be solved when developing an algorithm for a similar robot for real-world use. However, [20] still showed that there is potential for navigation for a similar robot, at least in simulation.

As we see, then most research around wall-following is restricted to the indoor environment, i.e. an environment with good lighting, a flat non-slippery floor, no obstacles and straight walls with easily defined edges. What we need out of a navigation algorithm for the RM3 is to however work in conditions that are the exact opposite of the aforementioned indoor environment. Thus the algorithm needs to be rigorous and take into account environmental factors like wind and rain. In addition, environments could have very different terrain, affecting the forces needed for effective navigation.

2.5 Cave navigation and mapping

In addition to wall-following, the requirements for the algorithm include navigation, which also brings with it the requirement to map the surroundings either before or during the navigation. Thus a basic understanding is needed about the how cave navigation and mapping is performed in recent research.

A 2018 paper explored path planning and navigation in off-world caves and dealt with similar issues — dark and large caves difficult for traversing with conventional sensors like cameras and miniature laser sensors. The localisation issue in that case was solved by having 2D laser range finders mounted on a servo that enabled 3D range scans. As they also planned to have multiple robots, then the other robots could track the single moving robot to measure absolute positioning, solving the issue of localisation. Though not the purpose of this thesis, this could be one way of achieving reliable localisation for the RM3 robot. [21]

A 2021 paper [22] and a 2022 paper [23] explored the surveying of caves using autonomous drones. Both used sensor fusion, combining an IMU with a visual camera for pose estimation, and then used LiDAR for generating a 3D occupancy grid. This was then used to survey further locations with a weight function to reduce entropy of the unknown parts of the environment. The positives of using drones for exploring caves include the relative easy of avoiding ground-based obstacles and high possible speed. However, using a drone of course provides a relatively limited amount of time to do mapping due to weight and battery constraints. In addition, due to weight constraints only cargo that is very light could be carried onboard for any other use cases in addition to mapping.

A paper published in 2021 investigated robotic inspection and mapping in cave-like environments using SLAM with LiDAR and a camera. The robot could employ various configurations for movement, including wheels, tracks and legs. Pathfinding was done by converting the mesh cloud obtained through SLAM into a graph, and then using Dijkstra's algorithm to find an appropriate path, considering multiple terrain metrics in the edge weights. [24]

The approach used in [24] would likely not work for our use case, hence the investigation into whiskers. However, for pathfinding a similar approach — Dijkstra's algorithm — could be used with a few modifications to take into account the lack of an existing map of the environment. In addition, the path planning algorithm should be able to handle a large grid, which could potentially also be 3D in the future, and also provide as effective a path as possible due to the robot being relatively slow.

2.6 Path planning algorithms

As the goal of this thesis is to also provide a path planning algorithm in addition to wall-following, then previous research on pathfinding algorithms was also looked into. In addition to Dijkstra, another algorithm (of many possible) that has been used for path planning is the optimal version of Rapidly-exploring Random Tree (RRT*) algorithm, as was done in [25]. The original version is the regular RRT algorithm and another more-known algorithm for pathfinding is A*. In a few comparative papers comparing A*, RRT and RRT*, all of them ([26, 27, 28])), found A* to be the best performing regarding optimality — which is inherent due to the A* providing a guaranteed optimal result and RRT not providing it — but also regarding its execution speed.

The A* algorithm was originally presented in 1968 [29]. It solves for the shortest path between an origin and a destination in a graph. If we can estimate the distance from one node to the destination node such that for every node we will never underestimate the distance, then we can use A* to obtain an optimal solution. This estimate is called a *heuristic* $h(i)$, i.e. estimated distance from node i to destination node t. For this, often the Euclidean distance is used. A* uses the function $d(i) = g(i) + h(i)$ for a total (lowest possible) estimation of the distance to destination node t, with $g(i)$ marking already travelled distance to the node. [30]

The A* algorithm is the following [30]:

1. Begin by setting $d(i) = +\infty$ for each $i \in N$. Next set $g(s) = 0$ and $d(s) = g(s) + h(s)$. Mark the current parent of the start node: $parent(s) = 0$. Finally, let $S = \{s\}$.
2. Node selection: identify node $v \in S$ where $d(v) \leq d(i)$ for all $i \in S$, set $S = S - \{v\}$.
3. Stopping criteria: if $v = t$, stop as the shortest path to destination t has been found; otherwise go to step 4.
4. Expanding: for each node w where $edge(v, w) \in A$, if $g(w) > g(v) + l(v, w)$ then update $g(w) = g(v) + l(v, w)$, set $d(w) = g(w) + h(w)$ and if $w \notin S$ let

$S = S + \{w\}$. In addition, keep track of the current parent: $parent(w) = v$. Continue with step 2.

S is the set of candidate nodes that have yet to be selected as the closest nodes. A is the set of all edges. $l(v, w)$ is the length of an edge between nodes v and w. A* is similar to Dijkstra, but adds a heuristic $h(i)$ to speed up the process of finding a path. In smaller graphs, both perform well, but in large graphs, A* outperforms Dijkstra by a large margin. [30]

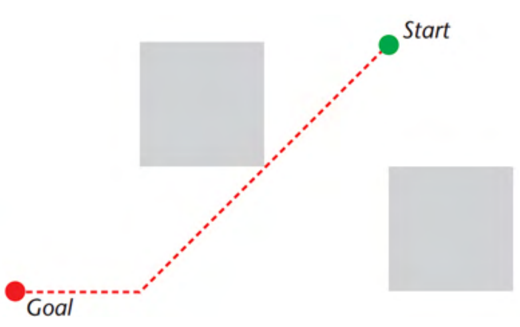
However, in cave environments there is large potential to cross more open areas. In addition, when having more resolution in the grid (i.e. less area per grid node), turning separately for each node can get very time-consuming, especially when there could be cases of movement to a point that would be straight, but not parallel to the axes of the grid. Thus, any-angle path planning algorithms can greatly improve navigation in cave environments.

One of the most known any-angle path planning algorithms for square grids is Theta*. It retains a similar algorithm to A*, but introduces a concept of line-of-sight between nodes. If line-of-sight is present between two nodes, then in most cases it is possible to reduce the path by traversing straight from a node that is not a direct neighbour instead of strictly following the edges between direct neighbours. Thus in most cases the algorithm reduces the traversed distance and also the necessary amount of turning, as can be seen in Figure 11. Steps 1-3 remain the same from the above algorithm, however step 4 (expanding) becomes more intricate. L denotes the set of edges with line-of-sight that have line of sight between the specific nodes. [31].

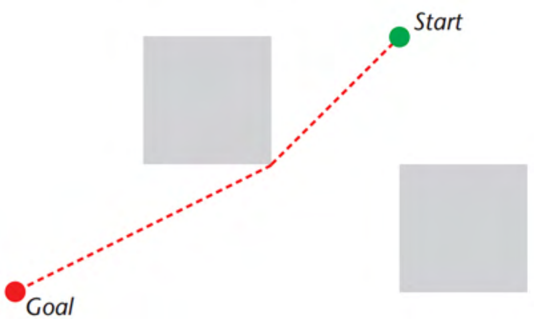
The algorithm is the following (with steps 1-3 remaining the same) [31]:

4. Expanding: for each node w where $edge(v, w) \in A$, assign a potential parent p. If $edge(parent(v), w) \in L$, then assign $p = parent(v)$, else assign $p = v$. Then continue the same as for A*: if if $g(w) > g(p) + l(p, w)$, then update $g(w) = g(p) + h(p, w)$, set $d(w) = g(w) + h(w)$ and if $w \notin S$ let $S = S + \{w\}$. In addition, keep track of the current parent: $parent(w) = p$. Continue with step 2.

In addition, some modifications have also been proposed for Theta*, such as Lazy Theta*, Angle-Propagation Theta*, Incremental Phi* etc, which make improvements in different areas to the algorithm [32]. Although basic Theta* should prove to be good enough for smaller caves that do not require 3D path planning, modifications like Lazy Theta* would allow for expansion of the path planning into larger and more demanding grids, including



(a) *A* path.*



(b) *Corresponding Theta* path.*

Figure 11. *A* and Theta* comparison [32].*

3D cave traversal (taking height into account) and much larger and more complex caves. As a lot of optimization was not required initially, then the basic Theta* path planning algorithm was chosen for the present thesis.

2.7 PID controller

In order to provide more stability to a system in a way which would also be easy to comprehend and control, a proportional-integral-derivative controller (PID controller) can be used. PID controllers are a simple way to implement feedback into a system as a control input and often are sufficient for control problems. As a result they are found in large numbers in all industries. For example, they are a key part in motor control, which is what is required in a navigation algorithm as well. Essentially, they provide a way to take into account the past (I), present (P) and future (D) errors of the system and based on those make a prediction on what the current control input should be. [33, pp. 1–5]

The standard form of a PID controller is presented in Equation (2.1). K is the controller gain, T_i the integral time and T_d the derivative time. $y(t)$ is the current value of the process variable and y_{sp} is the set-point value, i.e. the desired value for the process variable. $e(t)$ represents the error between the current value and the set-point. $u(t)$ represents the output control value, where every addend represents the P, I and D, respectively. [33, pp. 1–5]

$$\begin{cases} e(t) = y_{sp} - y(t) \\ u(t) = K\left(e(t) + \dfrac{1}{T_i}\displaystyle\int_0^t e(\tau)d\tau + T_d\dfrac{de(t)}{dt}\right) \end{cases} \tag{2.1}$$

Due to this, Equation (2.2) is another form which is widely used for a PID controller. In this case the values of K_p, K_i and K_d can be set directly and added together in a simpler way.

$$\begin{cases} u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \\ K_i = \frac{K_p}{T_i} \\ K_d = K_p T_d \end{cases} \tag{2.2}$$

The topic of tuning variables for PID controllers is widely researched. Some of the most known methods were proposed by Ziegler and Nichols in 1942, however these give moderately good tuning in only some scenarios. Often the tuning process starts by setting $T_i = \infty$, $T_d = 0$ and then raising the proportional gain Kp until the process starts to oscillate. Then, based on the gain at which it starts to oscillate and the period of the oscillation, all other variables are set according to specified constants. These values then usually still need to be manually tuned to improve performance. [33, pp. 158, 161, 169]

In [33, p. 225] a new Ziegler-Nichols replacement tuning process was proposed as well, named AMIGO. However, in [34] three tuning rulesets — including AMIGO — were compared with a software-based PID tuning method developed by them, and after tests all four of the methods were concluded to be non-optimal. So, even with software-based methods manual tuning is still required. Manual tuning can be performed using the instructions in Table 1.

Table 1. *Instructions for manually tuning a PID controller [35].*

Parameter Increased	**Rise Time**	**Overshoot**	**Settling Time**	**Steady-State Error**	**Stability**
K_p	Decrease	Increase	Small Change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Decrease Significantly	Degrade
K_d	Minor Decrease	Minor Decrease	Minor Decrease	No Effect	Improve (for small K_d)

3. Algorithm

In the following chapter the requirements for the algorithm, the algorithm implementation itself and the reasoning behind the implementation are described.

3.1 Requirements

The requirements for the algorithm were the following:

- wall and contour tracking
  - Implement a navigation strategy to control the robot's forward velocity while keeping the whiskers in contact with the wall, using a predefined pressure threshold.
- navigation
  - Implement a strategy to move the robot to a given target point, avoiding static and dynamic obstacles, using positioning information originating from an outside source.
- control mechanism
  - Implement a control mechanism with which to control the aforementioned navigation algorithms, i.e. allow to select the algorithm, to change speed, select a waypoint and provide other required inputs.

The reason behind using positioning information for navigation that originated from an outside source and not trying to create a localisation algorithm for the robot was due to this capability being developed as part of a separate study. In addition to the aforementioned requirements, an additional goal of the thesis was to investigate the optimality of the whisker configuration and propose new ones.

3.2 Methodology

To create the wall-following algorithm, an iterating method was used of first creating the most basic version imaginable, testing it out in the simulator, and then making required changes and improving it further. Thus for example the first step was a very imprecise wall-follower without any turning. Then improvements were made to stay parallel to the wall, thus increasing the algorithm's reliability of not losing touch of the wall. Then the gains of the PID controllers were tuned in a variety of simulated environments to provide a fast response time without overly large oscillation, and hard collision avoidance was added to provide certainty for avoiding collisions.

After the algorithm could successfully follow the walls in a variety of environments without collision or erratic behaviour, multiple rounds of real-world testing were done to verify the solution for use outside of a simulator and to fix any issues not occurring in simulation. These turned out to be integration issues such as sensor and motor values differing between simulation and the real robot, but also some issues that were not yet accounted for in the algorithm, such as whisker noise, whisker bias and the need to recalibrate the sensors during real-world use.

Thus, the overall principle for creating the algorithm was to first create a wall-following algorithm that would only use whiskers as the sensors. This would then validate the possibility of using whiskers for relatively simple movements. In addition, this algorithm could then be used in the event of the robot's position and orientation information becoming unavailable (e.g. due to sensor failure).

In addition, a navigation and path planning method was developed in parallel. As no method of localisation was present on the robot itself — but was being developed as part of the ROBOMINERS project — then instead an external camera-based pose estimation was used for the thesis. Using the positioning and orientation information, an algorithm could then be created for mapping out a cave and navigating it, using the whiskers for detecting walls and objects. In addition, incorporating the wall-following algorithm in the navigation could then be used to decrease the time taken to navigate to a point when the map was still unknown. The navigation and path planning methods were also tested out similarly with iterative methods in both the simulator and real-world use, just as with the wall-following algorithm.

During the testing of both algorithms, various whisker configurations were tried as well, as studies brought out in Sections 2.3 and 2.4 confirmed the size, shape, material can greatly affect the sensing. Also the direction and placement can of course greatly affect the sensing of the environment around it, with unidentified obstacles potentially causing the robot to get stuck on them.

3.3 High-level working principles

In the following section high-level working principles for wall-following and navigation algorithms are presented and discussed.

3.3.1 Wall-following

The high-level working principles of the created wall-following algorithm can be seen on Figure 12. The algorithm is repeated every time an update for the orientation of the whiskers is received (at 20 Hz). First, whisker preprocessing is performed in order to get a smoothed and unbiased pressure value for each whisker, indicating how much the whisker is being bent or pushed inwards, i.e. how much a whisker deviates from its resting orientation. Then, pressures are generalised from whiskers to their whisker arrays on each side, considering the average and maximum pressures. In addition, an additional error considering the distribution of whisker pressure — towards the start or end of the array — is then calculated.

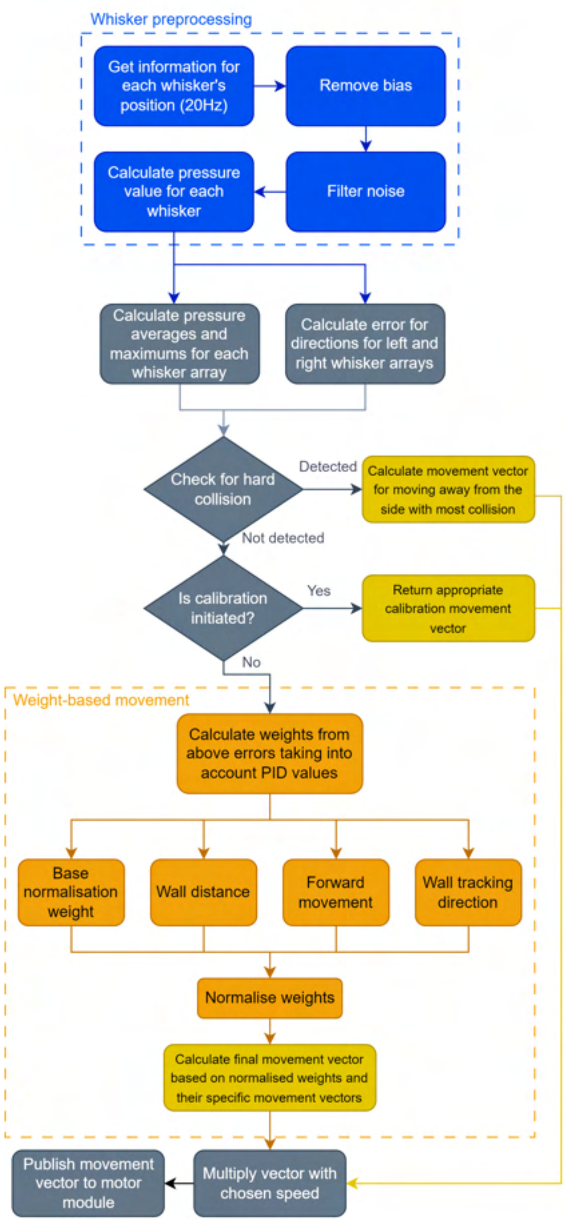

Whisker preprocessing
Get information for each whisker's position (20Hz)
Remove bias
Filter noise
Calculate pressure value for each whisker
Calculate pressure averages and maximums for each whisker array
Calculate error for directions for left and right whisker arrays
Check for hard collision
Detected
Calculate movement vector for moving away from the side with most collision
Not detected
Is calibration initiated?
Yes
Return appropriate calibration movement vector
No
Weight-based movement
Calculate weights from above errors taking into account PID values
Base normalisation weight
Wall distance
Forward movement
Wall tracking direction
Normalise weights
Calculate final movement vector based on normalised weights and their specific movement vectors
Publish movement vector to motor module
Multiply vector with chosen speed

Figure 12. *High-level working principle for wall-following.*

Then, if a whisker array is over a given threshold of either average or maximum pressure, hard collision avoidance is triggered, moving the robot in a reverse direction to the highest pressure whisker array. In addition, if calibration has been initiated — when the algorithm was selected or once per given time period — a separate recalibration algorithm is executed, outlined in Section 3.5.

If collision avoidance and recalibration are not triggered, the algorithm uses a weight-based movement system. This means that there are several predefined weights which depend on their specific defined errors. Each weight and its error also has a corresponding movement vector. After calculating the weights, they are then normalised and each weight factor's weight is multiplied with its respective movement factor, providing the final desired movement vector for the robot. That vector is then published to the motor module — not developed as part of this thesis — which transforms the vector into motor inputs and makes the screws move in the desired direction.

3.3.2 Navigation

The high-level working principles of the created navigation algorithm can be seen on Figure 13. Whisker preprocessing, whisker array average and maximum calculation, hard collision checking and calibration remain the same as in wall-following. However, as previously mentioned, the pose is also taken into account for the navigation. This gets transformed into a grid coordinate, which is the location of the robot in its internal map (stored as a graph). Using both the current location and the whisker contact points, collision points are then added relative to the internal map. If any collision points are in the given path, the path of course is recalculated.

The navigation movement part itself also uses weights, similar to the wall-following part. However, in order to speed up the navigation in unknown areas, wall-following is also incorporated where possible (outlined in Section 3.12). If wall-following is not applicable, weight-based movement is used, similar to the previous wall-following working principles, but with just the base normalisation weight, the next path node direction error weight and forward movement weight.

3.4 Processing whisker data

Each whisker sensor in every array allows the deflection of the whisker joint to be measured in three axes (see Figures 14 and 15). A ROS2 node developed for whisker data acquisition (based on the TLV493D-A1B6 C++ library for Arduino) calculates the orientation in

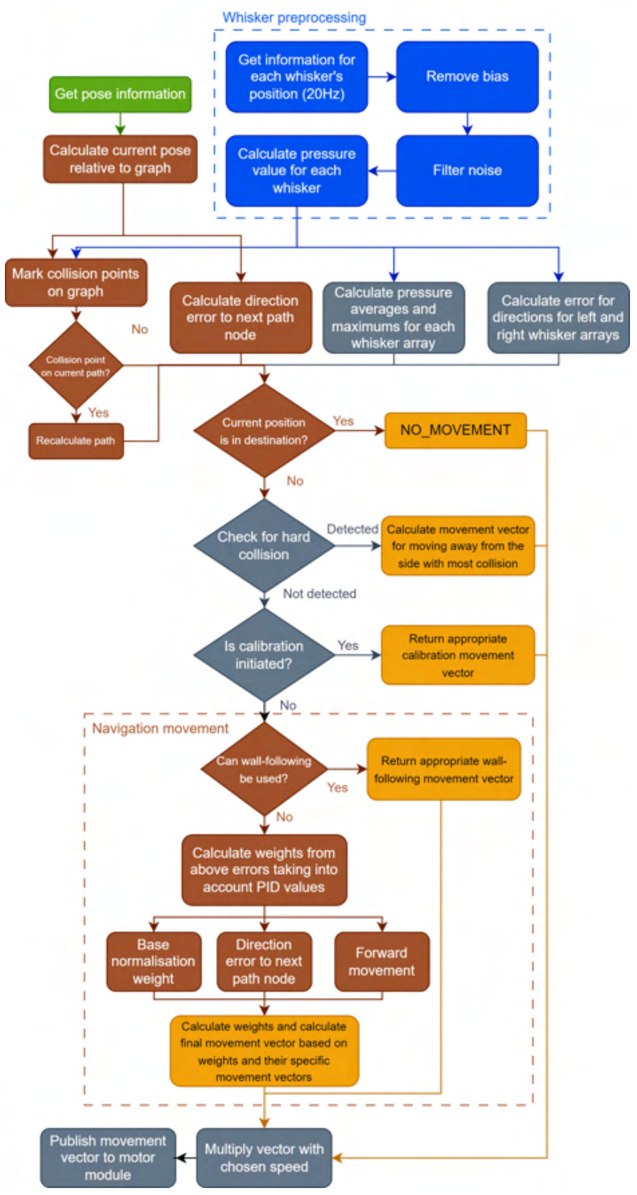

Whisker preprocessing
Get information for each whisker's position (20Hz)
Remove bias
Filter noise
Calculate pressure value for each whisker
Get pose information
Calculate current pose relative to graph
Mark collision points on graph
Calculate direction error to next path node
Calculate pressure averages and maximums for each whisker array
Calculate error for directions for left and right whisker arrays
Collision point on current path?
No
Yes
Recalculate path
Current position is in destination?
Yes
NO_MOVEMENT
No
Check for hard collision
Detected
Calculate movement vector for moving away from the side with most collision
Not detected
Is calibration initiated?
Yes
Return appropriate calibration movement vector
No
Navigation movement
Can wall-following be used?
Yes
Return appropriate wall-following movement vector
No
Calculate weights from above errors taking into account PID values
Base normalisation weight
Direction error to next path node
Forward movement
Calculate weights and calculate final movement vector based on weights and their specific movement vectors
Publish movement vector to motor module
Multiply vector with chosen speed

Figure 13. *High-level working principle for navigation.*

spherical coordinates, with ϕ being the azimuth, θ being the polar angle and r being unmeasurable, but remaining a constant depending on the chosen whisker. As in real-life some of the deflection is absorbed by the flexible stem, then these coordinates were entirely based on the direction of the lower part of the whisker, as that was the part of the whisker in direct contact with the sensor. Thus unless the whisker was in contact while remaining at or nearly at 90 degrees, the exact location of the end of the whisker was hard to determine, as the whisker itself was flexible. As the whiskers were rigid in the simulator, then this was not the case for simulation, causing some inconsistencies between simulation and real-world experiments.

Due to this the azimuth ϕ value was discarded for this thesis. Further reasoning for this was that the azimuth does not assist in finding the distance of an object or a wall, which

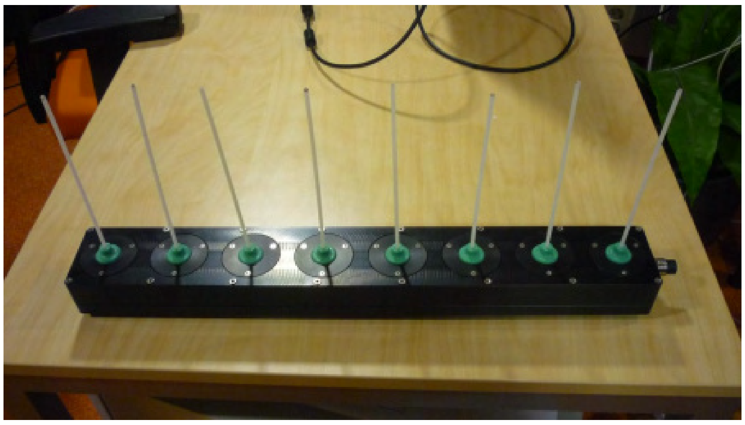


Figure 14. *Single whisker array consisting of 8 whisker sensors.*

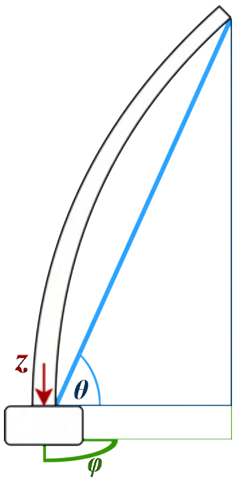


Figure 15. *Measurement axes of a whisker.*

was the main application of whiskers in the present thesis. However, this is one avenue which could be explored further in the future for refinement of the algorithm.

In addition to the azimuth ϕ and polar angle θ, the third axis that can be measured with the sensor is z. This axis stands for the perpendicularly applied pressure to the whisker, i.e. the force of pushing the whisker directly towards the sensor. Depending on the exact shape and material of the obstacle and its angle relative to the robot, both polar angle θ and perpendicular force z are useful for different purposes, and thus various ratios of their measurements were used — weights of w_θ and w_z respectively.

3.4.1 Introducing bias and noise in simulation

During real-world tests the whiskers generated some noise, which was not modeled in the simulator. In order to make simulation more realistic, experimental measurements were taken and then similar noise was added to the whisker measurements during preprocessing. This was a matrix of randomised values with Gaussian distribution (R), which was regenerated every time the whisker values were passed.

Another issue that needed handling was the bias of the whiskers. This refers to the whiskers not being in an exact orientation to give a reading of exactly zero when not in contact as they did in simulation. This issue derives from the whiskers being flexible, yet still somewhat stiff, and thus retaining some of the bend when deflected by a large amount. This could be improved in tests manually by bending the whiskers back to be entirely straight, however none of the whiskers were entirely perfect.

In order to simulate this behaviour, a matrix of randomised values with Gaussian distribution (B) is generated, and then added to the whisker sensor readings every time. So, in terms of simulation there are two random values applied to every whisker every time to simulate what happens in reality (see Equation (3.1)). Matrix W_{in} represents the values of the original whiskers and W represents the matrix of output values for the whiskers after adding noise and bias. The matrices are three-dimensional: first dimension being m whisker arrays, second dimension being n whiskers per array and the last dimension being the whisker's (θ, z) data measuring deflection and perpendicular pressure. For non-simulated use of course, no noise is added, i.e. $W = W_{in}$.

$$W = W_{in} + B + R \tag{3.1}$$

3.4.2 Handling initial bias and noise

In order to cancel out whisker bias B, a method is proposed of first putting the robot in an environment with a neutral state — i.e. no whiskers in contact with an object — and then measuring the average for whisker values for a given time period to reduce the noise on the bias reading itself. Then, for every reading the bias cancelling matrix B^- is subtracted from whisker matrix W. In Equation (3.2), $\theta_{ij}(t)$ represents the value of θ of whisker in position (i, j) at time t, where the robot is also in a neutral state. The same applies to $z_{ij}(t)$, but for the value of z, respectively.

$$\begin{aligned} b_{ij}^- &= \begin{bmatrix} \frac{\sum_{t=0}^{n-1} \theta_{ij}(t)}{n} & \frac{\sum_{t=0}^{n-1} z_{ij}(t)}{n} \end{bmatrix} \\ B^- &= \begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,n-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n-1,0} & b_{n-1,1} & \cdots & b_{n-1,n-1} \end{bmatrix} \end{aligned} \tag{3.2}$$

In order to filter out noise not resulting from bias, a simple method of averaging by a constant $r = 3$ was sufficient due to the relatively small amount of noise in the whisker sensor readings. In the preprocessing equation (see Figure (3.3)), $W'k$ represents the three-dimensional output matrix of the whiskers. The preprocessing equation seeks to reduce the bias and noise from whisker sensor readings which appeared in real-world use.

$$\begin{cases} W'_0 = [0]_{m \times n \times 2} \\ W'_k = W'_{k-1} + \dfrac{W_k - B^- - W'_{k-1}}{r} \end{cases} \tag{3.3}$$

3.5 Recalibration

One aspect that was not replicated in simulation and only discovered during real-world experiments was the occurrence of accumulating biases in whiskers during operation. An example of this can be seen on Figure 16. After the first two displacements, the resting point of the whisker remained very similar, however after a multitude of displacements, the resting point changed by about 3 degrees. As this skewed the perception of the robot into slightly invalid values — eventually resulting in a detected contact on the whisker's side — then a mechanism was needed to recalibrate the whiskers depending on certain conditions.

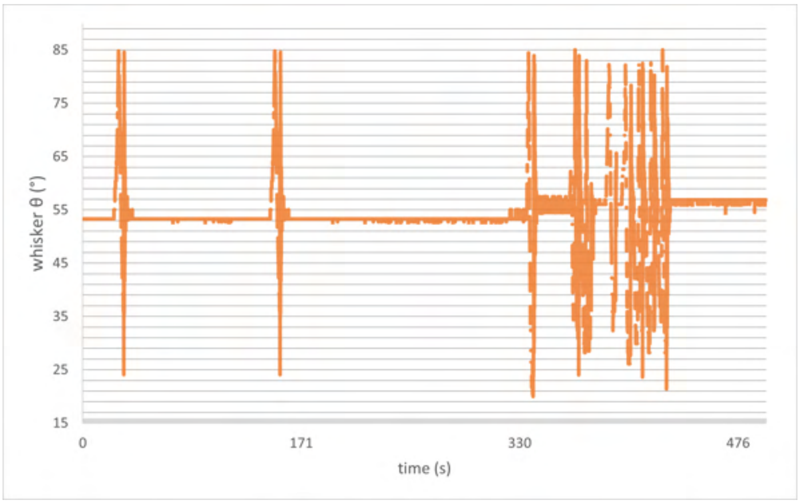


Figure 16. *Example of accumulating bias in whisker bias.*

A key problem is that at one given point in time it cannot be known whether the whisker is displaced due to bias or due to contact. Thus calibration has to consist of two parts, first navigating the whisker array away from any potential contact, and only then calibrating the array. The following strategy for calibration is used:

1. If calibration is initiated, create a set of uncalibrated directions $U = \{D_{left}, D_{right}, D_{front}, D_{rear}\}$.

2. If $U = \emptyset$, calibration is completed and regular navigation/wall-following movement can proceed. If $U \neq \emptyset$, choose a direction $d \in U$.
3. Navigating away from a potential contact:
   3.1. Create a deque q for calibrating d, i.e. a data structure which can hold a maximum of s elements, and if adding an element would go over the limit, then it deletes the first (i.e. oldest) element when adding a new element.
   3.2. Add the maximum pressure value in the array $P_{max}(d)$ (in more detail in Chapter 3.6.2) to deque q.
   3.3. If $|q| = s$, calculate the standard deviation σ of all elements in deque.
   3.4. If $|q| < s$ or $\sigma \geq \xi_{cal}$, move the robot in a direction opposite to d to reduce pressure on the whisker array that is being calibrated.
4. Calculate an updated matrix for the whisker array:
   4.1. Create a set M to store bias matrices and define the sample amount n with which to average the bias calculation (see Equation (3.2)).
   4.2. Store filtered whisker matrix W'_k in M. If $|M| \neq n$, wait for the next whisker data update and keep the robot still.
   4.3. If $|M| = n$, calculate the new bias matrix $B^{-'}$ (see Equation (3.4)), where new bias values $b'^{-}_{i,j}$ are added to the current whisker array's bias matrix values $b_{i,j}$. This can be achieved simply with matrix multiplication, for example for array $i = 1$ this can be achieved with Equation (3.5).
   4.4. Update $U = U - d$. Continue with Step 2.

$$\begin{aligned} b'^{-}_{ij} &= \begin{bmatrix} \frac{\sum_{t=0}^{n-1} \theta_{ij}(t)}{n} & \frac{\sum_{t=0}^{n-1} z_{ij}(t)}{n} \end{bmatrix} \\ B^{-'} &= \begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,n-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ b'_{i,0} & b'_{i,1} & \cdots & b'_{i,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n-1,0} & b_{n-1,1} & \cdots & b_{n-1,n-1} \end{bmatrix} \end{aligned} \tag{3.4}$$

$$B^{-'} = \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix} B^{-}_{old} + \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} B^{-}_{new} \tag{3.5}$$

For calibration, pressure sample amount s, bias sample amount n and standard deviation threshold ξ_{cal} have to be chosen, depending on the amount of sensor noise and the delay of motor movement between control inputs. For s and n, too low values provide inadequate bias compensation (due to not sufficient averages), and too high values take up too much time to calibrate. For ξ_{cal}, too high values mean the robot might still be in contact with a wall during calibration, and too low values mean the robot will never start calibration. Thus these values need to be calibrated for each type of robot separately. For the current robot, these values were chosen to be $s = 10$, $n = 20$ and $\xi_{cal} = 0.015$. Also, in Step 3.4. the velocity should be multiplied by a small constant to make calibration smoother and for the robot to travel a lesser distance during calibration.

In addition, of course a collision could be detected from the other side of the robot in case it is in a narrow gap. For this, the hard collision avoidance system (introduced in Section 3.10) would override the movement provided by the calibration. Thus calibration would occur in the place of hard collision, which would still be the best calibration location for the whisker array locally.

As to when to trigger calibration — in the present thesis it was triggered when a movement algorithm was selected and also every 1000 whisker cycles, i.e. once every 50 seconds. In the future, this could be optimised further to only trigger calibration once conditions occur in whiskers that are likely to move the bias point, i.e. high pressure values, and also to only trigger calibration for that specific direction.

3.6 Errors calculated from whisker pressures

In order to obtain information about the surroundings, the polar angle (θ) and perpendicular forces (z) of single whiskers are converted into one general value for each whisker and then into generalised errors over entire whisker arrays.

3.6.1 Individual whisker errors

A single whisker is always mapped to a single value, pressure p, which is normalized, i.e. $0 \leq p \leq 1$. A p value 0 means no contact for the whisker and a p value 1 means that the whisker effectively cannot bend any more without likely damage to the whisker. The normalization values $\theta_{max} = 0.7$ and $z_{max} = 25$ were thus chosen during experiments to take this into account. Weights w_θ and w_z can be varied depending on the experiment results to prioritise the robot to use more of the polar angle θ or the perpendicular force z. Due to measurement noise and the possibility of getting measurements beyond the maximum value, it is also necessary to explicitly cap p_θ and p_z values in the formulas to retain p normalization to $[0, 1]$.

The formula for p is presented in Equation (3.6). In the following chapters, p_{ij} will be used to refer to the pressure calculated for a whisker that is in whisker array i and is the whisker in the j'th position in the whisker array.

$$\begin{cases} p_z = max(1, abs(z_{rk}/z_{max})) \\ p_\theta = max(1, (\pi/2 - abs(\theta_{rk}))/(\pi/2 - z_{max})) \\ w_\theta \geq 0, w_z \geq 0, w_\theta + w_z = 1 \\ p = w_\theta * p_\theta + w_z * p_z \end{cases} \tag{3.6}$$

In addition, as the algorithm is designed to work for following a wall on both its right side or its left side, then the way of processing whiskers also changes when switching sides. As such, we define the value $sign_{wall}$ for use in further equations (see Equation (3.7)).

$$sign_{wall} = \begin{cases} 1, \text{if following right wall} \\ -1, \text{if following left wall} \end{cases} \tag{3.7}$$

3.6.2 Generalised whisker array errors

In order to do wall-following and navigation, generalising errors to whisker arrays is needed as well. For calculating the errors, both average and maximum pressure p values are first calculated for each whisker array i, marked by $P_{avg}(i)$ and $P_{max}(i)$ (see Equations (3.8) and (3.9)). In the equations, the total number of whiskers in an array is marked as n.

$$P_{avg}(i) = \frac{\sum_{j=0}^{n-1} p_{ij}}{n} \tag{3.8}$$

$$P_{max}(i) = \max_{j \in \{0,1,2,\ldots,n\}} p_{ij} \tag{3.9}$$

In addition, helper definitions are defined for upcoming equations: $D_{tracked}$ as the direction of the tracked wall and $D_{opposite}$ as the direction opposite of the tracked wall direction. Thus, for example if the indices of the left and right whisker arrays are respectively 3 and 4, and if the tracked wall direction is chosen to be the left side, then $D_{tracked} = 3$ and $D_{opposite} = 4$. Thus also $P_{avg}(3) = P_{avg}(D_{tracked})$ and $P_{avg}(4) = P_{avg}(D_{opposite})$. These helper definitions are defined because the tracked wall direction can change depending on configurations. In addition — for clarity — when only a whisker array in a certain direction is applicable, D_{left}, D_{right}, D_{front} and D_{rear} will be used instead of using a certain i value as above.

3.6.3 Directional error

The directional error was introduced for keeping the robot as parallel to the wall (or object) next to it as possible, primarily for the purpose of smooth wall-following. In addition, in cases where the robot was surrounded by walls on two or three sides, it was one way for the robot to orient itself correctly for continuing to follow the wall on a chosen side.

The basic principle of the directional error is to always try to keep the whiskers of an array at the same length if possible. So if on the right side the frontmost whisker is the only whisker to have contact, then the robot should turn left in the hopes of finding a wall with also the other whiskers and then having the whiskers at the same length if possible. However, this can be generalised for all whiskers. Thus if there is a tunnel that the robot can fit through, then it would average the directions on both left and right sides to find the optimal way through.

In addition, including the front and rear whiskers in the directional error also serves a valuable purpose. As can be seen on Figure 17, pressure on front whiskers can serve as another reason to rotate. If the desired wall to be followed is on the right, contact on the front should force the robot to steer left (as there's a wall or an impassable gap on the right and front). If the desired wall is on the left, it should steer right. For calculating

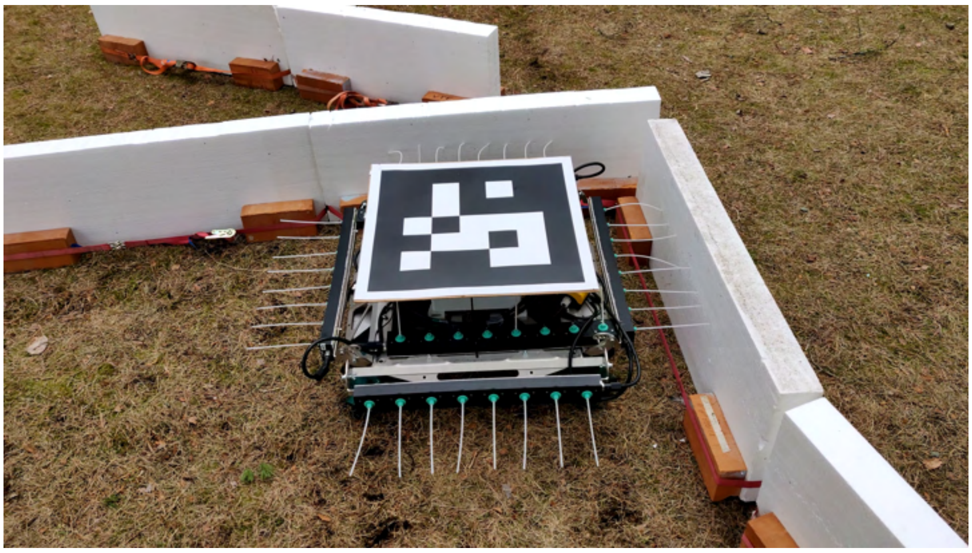


Figure 17. *Example of front whisker contact forcing rotation.*

the directional error, the concept of a directional whisker weight w_{dir} was introduced and generalised for any whisker array that has equidistant whiskers, regardless of the number of whiskers (see Figure 18 and Equation (3.10)). As a reminder, $\lfloor a/b \rfloor$ means integer division, i.e. a divided by b, with the fractional part discarded.

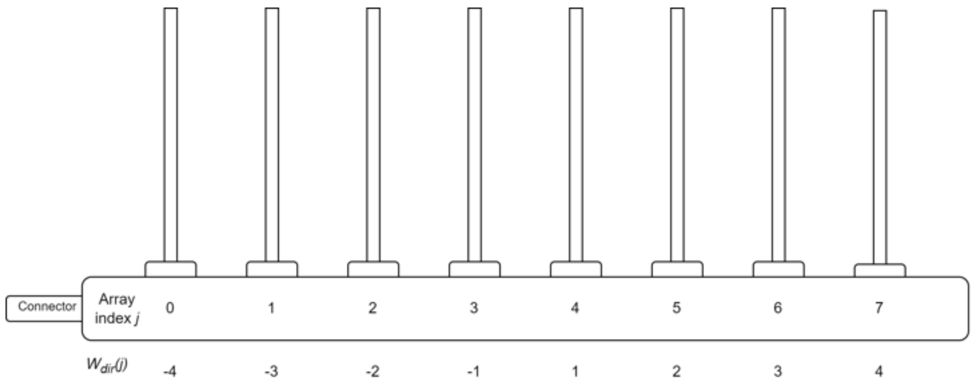


Figure 18. *Calculation of the directional whisker weight w_{dir} for $n = 8$.*

$$w_{dir} = \begin{cases} \lfloor (j-n)/2 \rfloor, \text{if odd} \\ \lfloor (j-n)/2 \rfloor, \text{if even and } j \geq \lfloor n/2 \rfloor \\ \lfloor (j-n)/2 \rfloor - 1, \text{if even and } j \leq \lfloor n/2 \rfloor \end{cases} \tag{3.10}$$

In addition, the resulting total directional error for the whisker array is also normalised to $[0, 1]$. Thus, the directional error for a given whisker array i is calculated with Equation (3.11).

$$Ec_{dir}(i) = \frac{\sum_{j=0}^{n-1} p_{ij} w_{dir}(j)}{2 \cdot \sum_{k=0}^{\lfloor n/2 \rfloor} k} \tag{3.11}$$

In order to create a correct directional error, knowledge of the placement of the whiskers is required (see Figure 19). In addition, it was necessary to choose either the left or the right side according to which other values will be set to either add or subtract from the error. In this thesis the right side was chosen. This means that above-zero values from the weight functions will come from right side sensors 4–7, which makes the robot turn left. To get the same behaviour from left-side sensors 0–3, $E_{dir}(D_{left})$ have to be negated. For front and rear whisker arrays, using the directional weights would not give additional benefit — for the front side the robot should steer left (or right in the case of following the left wall) regardless of the whiskers that are in contact. However, using both the P_{avg} and the P_{max} showed better adaptability than using just one, which led to using weights for them. As for the rear whiskers, not much benefit was observed in regards to having the rear array included in the directional error. Depending on the environment and the specific angles it would usually disrupt the wall-following regardless if it was added or subtracted from the error. This means that rear whiskers were excluded from the calculation of the final

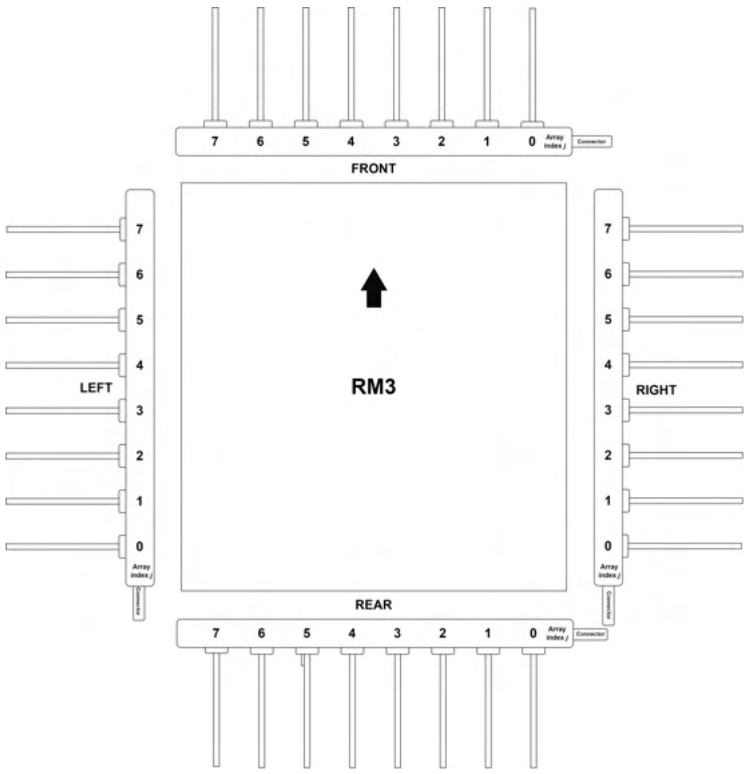


Figure 19. *Layout of the RM3 whiskers and their indices.*

directional error, E_{dir} (see Equation (3.12)).

$$\begin{cases} w_{avg} \geq 0, w_{max} \geq 0, w_{avg} + w_{max} = 1 \\ E_{dir} = Ec_{dir}(D_{right}) - Ec_{dir}(D_{left}) \\ \qquad + sign_{front}(w_{avg}P_{avg}(D_{front}) + w_{max}P_{max}(D_{front})) \end{cases} \tag{3.12}$$

3.7 Path planning

In order to follow a path, first the path has to be calculated after a destination has been entered through a control application (described in Section 3.14). In this section, the method of positioning, path planning and converting into an error of path following angle are described.

3.7.1 Grid structure

As exploration is part of the functions of the robot, then before navigating through an entire cave, the size of the cave cannot be known. Thus variable and fixed-size arrays were considered unsuitable as data types and the structure for the grid was chosen to be a dictionary, with the keys being (x, y) values, where x and y were signed integers. For future use, this can also easily be expanded to be a three-dimensional grid. Non-existing values in the dictionary are thus considered passable until contact is detected on that square. In addition, visited squares are marked separately to retain knowledge of the resulting map.

As localisation with whiskers was not a goal for the present thesis — will be developed and integrated separately — then the position of the robot was acquired with a camera and fiducial markers in real-world testing. For simulation, the coordinates of the robot in the simulated world were simply used. As the scale of these coordinates could be different, the resolution of the map also needed to be tuned based on the information or units of sensors used for localisation.

In order to tune the resolution of the grid, grid square size λ is defined. Lower values allow for more precise edges of obstacles, but too low values can increase the amount of time it takes for exploration significantly. In order to transform the location of the robot into the grid's dimensions, Equation (3.13) is used. This is used for the direction errors, for navigating to the next point. However, for graph-based path planning, integer values for grid square locations are needed. For this, the location that is transformed to the grid's dimension is rounded to the nearest integer (see Equation (3.14)). This gives the location of the grid intersection to which the robot is currently nearest to.

$$\begin{cases} x' = \dfrac{x}{\lambda} \\ y' = \dfrac{y}{\lambda} \end{cases} \tag{3.13}$$

$$\begin{cases} x'_{node} = \left\lfloor \dfrac{x}{\lambda} + \dfrac{1}{2} \right\rfloor \\ y'_{node} = \left\lfloor \dfrac{y}{\lambda} + \dfrac{1}{2} \right\rfloor \end{cases} \tag{3.14}$$

3.7.2 Marking collisions

In order to mark collisions based on the whiskers, the current orientation of the robot is needed. As with the position, this was also obtained through a camera and markers for real-world testing, or simply through simulator coordinates for simulation. For a collision to be marked, a threshold of $\xi_{mark} = 0.1$ was chosen. Thus, if any whisker pressure reaches over the given threshold, i.e. $p \geq \xi_{mark}$, a collision is marked using the whisker's position.

Table 2. *Collision angle parameters based on whisker array.*

Direction of whisker over the threshold	**Collision angle** α_{array}	**Whisker angle multiplier** $\lambda_{whisker}(i)$
D_{left}	90°	-1
D_{right}	-90°	1
D_{front}	0°	1
D_{rear}	180°	-1

Table 3. *Whisker index to $\alpha_{whisker}$ mapping.*

Whisker index	0	1	2	3	4	5	6	7
$\alpha_{whisker}$	-45°	-30°	-20°	-10°	10°	20°	30°	45°

The whisker position is converted into an angle relative to the robot using Tables 2 and 3. Table 2 first displays a_{array}, showing the angle of the direction of the whisker array relative to the front of the robot. It also displays $\lambda_{whisker}(i)$, which relates to the direction of the physical whisker array. Thus it allows to map the values of $\alpha_{whisker}$ in a unified direction such that higher whisker indices would be towards the left of the robot and lower indices towards the right.

The resulting total angle $\alpha_{collision}$ can then be used to mark a square on the grid as impassable, as displayed on Figure 20. This process is repeated for all whiskers during whisker preprocessing every time information about whisker states is received.

$$\alpha_{collision} = \alpha_{robot} + \alpha_{array} + \lambda_{whisker}(i)\ \alpha_{whisker} \tag{3.15}$$

3.7.3 Theta* implementation

Most of the Theta* algorithm could just be implemented as written in Section 2.6 using the graph that was derived above. However, a line-of-sight check also needed to be implemented. This is implemented by forming a line from point (x_1, y_1) to another point x_2, y_2, as can be seen in Figure 21. Thus the slope a of the line can be obtained. Next, all intersections of the line between the two points are calculated. This is accomplished by first choosing the x coordinate, increasing its absolute value towards the second point by 1 (as it could be negative as well) and marking down all the edges it passes through, marking the node position on both sides until $x = x_2$. Then the same process is repeated for the

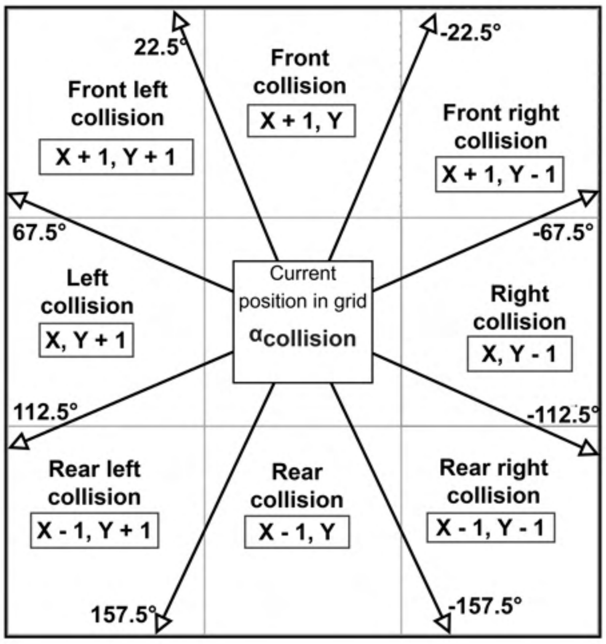


Figure 20. *Collision marking on the grid.*

y coordinate. In addition, special cases have to be added for $x_1 = x_2$ and $y_1 = y_2$, as these would have an invalid slope a of either 0 or $\pm\inf$, not allowing a correct calculation. This is a similar method of producing line-of-sight calculations as was used in the original Theta* proposition paper [31].

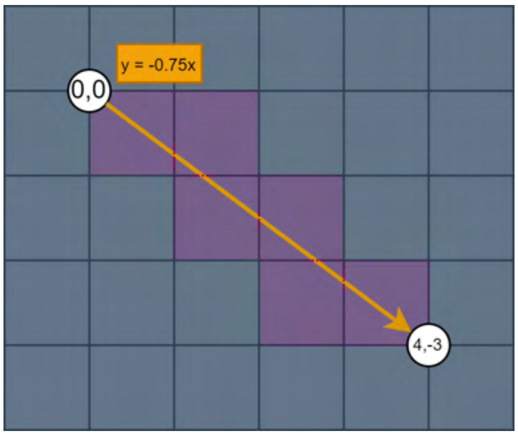


Figure 21. *Basic Theta* line-of-sight calculation.*

This sort of line-of-sight calculation however considers the robot to be a point without any dimensions. Thus a path between two squares could be planned to cross that would require an infinitely small robot to pass. As such, squares on the sides of these points should also be included, as seen in Figure 22. As seen on the figure, additional line-of-sight checks are performed, taking into account a constant c. Thus the space that is needed for the robot to pass through is enough. The same approach for considering the width of the robot was also used in [31].

As the algorithm states, this line-of-sight check is used between all nodes except neighbouring ones. If any of the nodes acquired using this way is marked as impassable, no line of sight is present between the start and end node. Thus, when executing Theta*, the performance of the line-of-sight check can be improved by performing a check for the square being an obstacle or not instead of adding the square to a set S, and immediately returning *False* if the node is an obstacle.

The algorithm for obtaining the nodes for the line-of-sight check is presented below:

1. Create set $S = \emptyset$ to store visited squares. Define $\delta x_{min} = min(0, x_2 - x_1)$, $\delta x_{max} = min(0, x_2 - x_1)$ and repeat for y_{min} and y_{max}.
2. For x collisions:
   2.1. Assign $\delta x = 0$ and $S_{pot} = \emptyset$.
   2.2. If $\delta x > x_2 - x_1$, then move to Step 3.
   2.3. Update $\delta x = \delta x + sgn(x_2 - x_1)$ and the resulting $x_k = x_1 + \delta x$. Then calculate the y part and transform it into a grid coordinate: $y_k = \lfloor y_1 + a\delta x + c \rfloor$. If $y_1 = y_2$, use $y_k = \lfloor y_1 + c \rfloor$.
   2.4. For points (x_{k-1}, y_k) and (x_k, y_k), if $x_1 + \delta x_{min} \leq x \leq x_1 + \delta x_{max}$ and $y_1 + \delta y_{min} \leq y \leq y_1 + \delta y_{max}$, add the point to S. Then go back to Step 2.2.
3. For y collisions:
   3.1. Assign $\delta x = 0$.
   3.2. If $\delta y > y_2 - y_1$, then finish the loop and return the visited square set S.
   3.3. Update $\delta y = \delta y + sgn(y_2 - y_1)$ and the resulting $y_k = y_1 + \delta y$. Then calculate the x part and transform it into a grid coordinate: $x_k = \lfloor x_1 + (y_k - c)/a \rfloor$. If $x_1 = x_2$, use $x_k = \lfloor x_1 + c \rfloor$.
   3.4. For points (x_k, y_{k-1}) and (x_k, y_k), if $x_1 + \delta x_{min} \leq x \leq x_1 + \delta x_{max}$ and $y_1 + \delta y_{min} \leq y \leq y_1 + \delta y_{max}$, add the point to S. Then go back to Step 2.2.

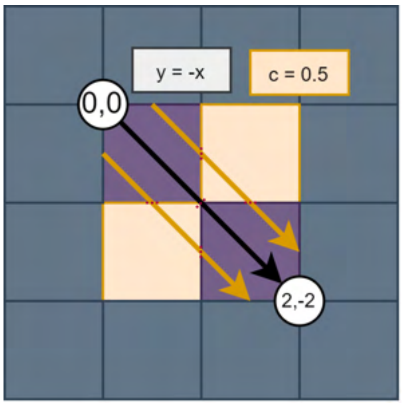


Figure 22. *Theta* line-of-sight calculation with robot size modification.*

3.7.4 Path following angle error

Theta* calculates the sequence of path nodes $p_0, p_1, ..., p_m$ that is required to reach the chosen destination. The sequence of path nodes is pruned at the start, i.e. path nodes that are closer to the current robot location in Euclidean distance than a given threshold are eliminated. This needs to be done as it is impossible to reach a given point precisely with floating point numbers, and also it is required for the robot to not try to navigate back in case the robot overshoots the initial goal by a small amount. This threshold was chosen to be half the grid square size of the graph, i.e. $\xi_{remove} = 0.5$. For the final destination this was chosen to be $\xi_{remove\_final} = 0.1$ for a more accurate final position. In addition, for the later aspect of utilising wall-following during path planning, then $\xi_{remove\_wf} = 3$ was utilised when wall-following was being used in order to encourage wall-following. Pruning the sequence of path nodes gives a new sequence: $p'_0, p'_1, ..., p'_n$.

Then, the required angle between the current unrounded translated position s' (floating point precision) — obtained using Equation (3.13) — and the translated destination position p'_0 (integer precision) is calculated using Equation (3.16). As in the algorithm all the values are in degrees, then the error is also converted to degrees. Then, the current position of the robot is used to obtain the error E_{path} for the path following regarding direction (see Equation (3.17)).

$$\begin{cases} s' = (x'_s, y'_s) \\ p'_0 = (x'_0, y'_0) \\ \alpha_{target} = \frac{180}{\pi} atan2(y'_0 - y'_s, x'_0 - x'_s) \\ \alpha_{temp} = (\alpha_{robot} - \alpha_{target}) \mod 360 \end{cases} \tag{3.16}$$

$$E_{path} = \begin{cases} \alpha_{temp} - 360, \text{ if } \alpha_{temp} > 180 \\ \alpha_{temp}, \text{ if } \alpha_{temp} \leq 180 \end{cases} \tag{3.17}$$

3.8 Weight-based movement system

In order to avoid complex decision trees and the need to smooth their movements, but still retain traceability of what was determining the movement of the robot, the algorithm is designed with the use of weights. A weight factor ϕ_i consists of a scalar *weight* w_i and a *movement vector* $\vec{v_i}$ (see Equation (3.18)). The way the parts of movement vector $\vec{v_i}$ impact movement is described in Table 4. In addition, in the algorithm a short description is also

Table 4. *Impact of v_x, v_y and $\dot{\psi}$ on motion.*

Parameter	**Positive values**	**Negative values**
v_x	Forward motion	Backwards motion
v_y	Movement left	Movement right
$\dot{\psi}$	Turn right	Turn left

added to the weight factor, which describes the purpose of the specific weight factor.

$$\begin{cases} w_i \geq 0, \quad -1 \leq v_x, v_y, \dot{\psi} \leq 1 \\ \vec{v}_i = \begin{bmatrix} v_x & v_y & \dot{\psi} \end{bmatrix} \\ \phi_i = (w_i, \vec{v}_i) \end{cases} \tag{3.18}$$

To determine the robot's final movement vector $\vec{v'}$, the weights are first normalised to a sum of 1 to keep values within expected limits. Then, the weights are multiplied with the respective weight factors' movement vectors to obtain the final movement vector. These two steps are combined into Equation (3.19). This final movement vector $\vec{v'}$ is then published to a separate module — not developed as part of this thesis — which is responsible for translating the movement vector to motor speeds.

$$\vec{v'} = \frac{\sum_{i=1}^{n} w_i \vec{v}_i}{\sum_{i=1}^{n} w_i} \tag{3.19}$$

Equation (3.19) and the overall logic can be considered similar to one of the most known fuzzy logic systems: the Takagi-Sugeno model introduced in [36]. In fact, the exact same equation is used in their method for defuzzification, with the vector being replaced by a scalar value. However, two major differences are that there is no strict rulebase for the method proposed in this thesis and also the inputs used are purely linear, without the need to go through a fuzzification process beforehand. Moreover, in the chosen method all the values affect each other due to normalisation. Though this is possible to implement in fuzzy logic, it would make the rulesets very complex. Thus this is considered as a separate method different to the Takani-Sugeno system.

3.9 Weight factors

In the following section all the weight factors created for the navigation and wall-following algorithms are presented. In addition, all weight factors — except the base normalisation and forward movement weight factors — also utilised a PID controller for easy tuning. The weight factors will be presented in the order they were tuned.

3.9.1 Base normalisation weight factor

In case the weights are small, then without having a baseline normalisation weight factor in place, the weights themselves would not matter, but only their size relative to each other. This means that for example if only a slight directional error is detected and the other errors are near zero, the direction weight factor would get exceedingly high values after normalisation. This in turn would prompt the robot to rotate at full speed, which is unwanted behaviour if the error is small. Thus a normalisation factor (see Equation (3.20)) is introduced, keeping small weights small in proportion.

$$\begin{cases} w_{base} = 1 \\ \vec{v}_{base} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \\ \phi_{base} = (w_{base}, \vec{v}_{base}) \end{cases} \tag{3.20}$$

3.9.2 Wall distance

The wall distance error E_y, i.e. the y-axis error, is used for keeping the robot at a fixed distance from the wall. The error is calculated (see Equation (3.21)) by adding together P_{avg} and P_{max} with specific weights w_{max} and w_{avg} determined during experiments.

$$\begin{aligned} E_y =& (w_{avg} P_{avg}(D_{tracked}) + w_{max} P_{max}(D_{tracked})) \\ &- (w_{avg} P_{avg}(D_{opposite}) + w_{max} P_{max}(D_{opposite})) \end{aligned} \tag{3.21}$$

During experiments it was noted that using just the $P_{avg}(i)$ disregarded some whiskers being very close to the wall or an object and thus the robot either eventually collided with the object or the whisker itself deformed too much. In addition, using just the $P_{max}(i)$ value did not provide a consistent performance for rough surfaces and resulted in too much oscillation even at low PID values. The resulting weight factor is brought out in Equation (3.22).

$$\begin{cases} vec(D_{tracked}) = \begin{cases} \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}, \text{if } D_{tracked} = D_{left} \\ \begin{bmatrix} 0 & -1 & 0 \end{bmatrix}, \text{if } D_{tracked} = D_{right} \end{cases} \\ w_{wall\_dist} = |E_y| \\ \vec{v}_{wall\_dist} = \begin{bmatrix} 0 & sgn(E_y) vec(D_{tracked}) & 0 \end{bmatrix} \\ \phi_{wall\_dist} = (w_{wall\_dist}, \vec{v}_{wall\_dist}) \end{cases} \tag{3.22}$$

3.9.3 Wall tracking direction

The wall tracking direction E_{dir} is used for keeping the robot as parallel to the wall as possible. As previously mentioned, some methods have used the least squares method for this, but the unpredictable environment RM3 is designed to work in makes it hard to apply. Instead, this weight factor directly uses E_{dir} derived in Equation (3.12), without any modification in between, except a PID. The directional weight factor is brought out in Equation (3.23).

$$\begin{cases} w_{dir} = |E_{dir}| \\ \vec{v}_{dir} = \begin{bmatrix} 0 & 0 & -sgn(E_{dir}) \end{bmatrix} \\ \phi_{dir} = (w_{dir}, \vec{v}_{dir}) \end{cases} \tag{3.23}$$

3.9.4 Path following angle

The path following angle error E_{path} is used for keeping the in the correct orientation to follow the calculated path. The path following angle weight factor also directly uses the E_{path} error, derived in Equation (3.17), without any modification in between, except a PID. The path following weight factor is brought out in Equation (3.24).

$$\begin{cases} w_{path} = |E_{path}| \\ \vec{v}_{path} = \begin{bmatrix} 0 & 0 & sgn(E_{path}) \end{bmatrix} \\ \phi_{path} = (w_{path}, \vec{v}_{path}) \end{cases} \tag{3.24}$$

3.9.5 Forward movement

For wall-following to happen, a robot must vary its forward momentum depending on the need to get closer to the wall so it would not overshoot the wall or collide with it. In addition, the robot should not go forward when it is not in contact with the wall it should follow or when the angle of the wall in relation to the robot is not close to being parallel. This is achieved by setting a weight for the forward movement weight factor that depends on w_{wall_dist} and w_{dir} (see Equation (3.25)). In order to select a value for the forward movement, first γ_{wall_dist} was tuned to be appropriate, i.e. to not cause large oscillations, but still be responsive. After this, γ_{dir} was tuned.

$$\begin{cases} \gamma_{wall\_dist} = max(1, min(0, 1 - 2|E_y|)) \\ \gamma_{dir} = max(1, min(0, 1 - 1.25|E_{dir}|)) \\ w_{forward\_wall} = \gamma_{wall\_dist}\gamma_{dir} \\ \vec{v}_{forward} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \\ \phi_{forward\_wall} = (w_{forward_wall}, \vec{v}_{forward}) \end{cases} \tag{3.25}$$

As navigation does not use E_y or E_{dir}, then a different weight calculation is also needed for forward movement. The purpose of the forward weight in this case is to slow the robot down when approaching an obstacle on the front. For navigation, using the maximum pressure in the front was chosen (see Equation (3.26)). The value was chosen such that $w_{forward\_path}$ would be 1 with no pressure on the front and 0 if the hard collision avoidance system would activate.

$$\begin{cases} w_{forward\_path} = max(1, min(0, 1 - 1.25P_{max}(D_{front}))) \\ \vec{v}_{forward} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \\ \phi_{forward\_path} = (w_{forward\_path}, \vec{v}_{forward}) \end{cases} \tag{3.26}$$

3.10 Hard collision avoidance

As there can occur cases where the weight-based algorithm (or any algorithm for that matter) could potentially cause damage to the robot's components, especially for high velocity values, then a hard collision avoidance system was implemented. The goal is to have a predefined threshold for when to invoke hard collision avoidance, upon which the robot would stop and slowly move back from the side with the most pressure. The implemented hard collision avoidance is presented below. The chosen thresholds were $\xi_{avg} = 0.4$ for $P_{avg}(i)$ and $\xi_{max} = 0.8$ for $P_{max}(i)$.

In addition, the need for an additional more complex mechanism arose in order to avoid perpetual oscillations between either front-rear or left-right directions. This can occur for example if the robot moves into an appropriately-sized hole on which both sides have a pressure above the threshold or very close to it. In addition, a threshold reduction $\xi_r = 0.3$ was defined for avoiding this oscillation. If a danger of perpetual oscillation is detected and if wall-following is currently being performed (i.e. $D_{tracked}$ is set), then the movement direction is simply the opposite of the tracked direction, as we can be sure there's a wall on the other side. However, if hard collisions are detected for example on the left and right, then inclination information needs to be used to obtain an appropriate direction, as can be seen in Step 6.

1. Set $dir_{prev} = 0$ if not set, for tracking previous hard collision movements that did not result from avoiding perpetual oscillation.
2. If $\max_{i \in D} (P_{avg}(i)) \geq \xi_{avg}$ or $\max_{i \in D} (P_{max}(i)) \geq \xi_{max}$, then hard collision was detected and maximum hard collision direction i was obtained (taking P_{avg} with a higher priority as is written). If no hard collision was detected, then regular movement can proceed.
3. Check for perpetual oscillation:
   - 3.1. Check which directions have $P_{avg}(i) \geq \xi_{avg} - \xi_r$ or $P_{max}(i) \geq \xi_{max} - \xi_r$. If both directions on an axis (i.e. both D_{left} and D_{right} or D_{front} and D_{rear}) were over a threshold, then skip to Step 5.
   - 3.2. Check if dir_{prev} is the opposite direction of array i. If yes, skip to Step 5.
4. If checks for perpetual oscillation were false:
   - 4.1. Move the robot in the opposite direction (see Table 5) of the high-pressure direction in order to move away from the hard collision. This should be performed at a slow pace with a fairly low value to ease transition back into regular weight-based movement.
   - 4.2. Set dir_{prev} to the direction the robot moved towards.
5. If $D_{tracked}$ is set:
   - 5.1. Move the robot opposite from $D_{tracked}$ at a slow pace. In addition, slight rotation towards the $D_{tracked}$ direction should be added to aid wall following.
6. If $D_{tracked}$ is not set (thus wall-following not currently enabled):
   - 6.1. If directions were front-rear, calculate inclination $inc = Ec_{dir}(D_{front}) + Ec_{dir}(D_{rear})$. If $inc \geq 0$, movement direction should be D_{right}. Otherwise, it should be D_{left}.
   - 6.2. Otherwise (i.e. if directions were left-right), calculate inclination $inc = Ec_{dir}(D_{left}) + Ec_{dir}(D_{right})$. If $inc \geq 0$, movement direction should be D_{rear}. Otherwise, it should be D_{front}.
   - 6.3. Move the robot towards the chosen movement direction at a slow pace, without any rotation added.

3.11 Wall-following algorithm

This section describes the final algorithm for wall-following in detail. A higher level overview was presented in Section 3.3.1 and on Figure 12. First, basic safety and calibration checks are performed, in order to ensure the safety and correct calibration of the robot. Then, contact with the tracked wall is checked, and the robot is moved with a constant

Table 5. *Movement vectors for hard collision avoidance.*

Whisker index	**Desired movement direction**	**Published no-weight movement vector**
D_{left}	Right	$\begin{bmatrix}0 & -1 & 0\end{bmatrix}$
D_{right}	Left	$\begin{bmatrix}0 & 1 & 0\end{bmatrix}$
D_{front}	Backward	$\begin{bmatrix}-1 & 0 & 0\end{bmatrix}$
D_{rear}	Forward	$\begin{bmatrix}1 & 0 & 0\end{bmatrix}$

speed towards the wall if no contact with the wall is detected. This prevents the weight-based movement from having integral windup when not in contact with the wall and allows higher PID gains and thus more responsive PID output values. Once the movement vector is returned, it is also multiplied with a velocity factor chosen in the control application (described in 3.14) to either increase or decrease robot velocity.

1. Check for hard collision using the algorithm defined in Section 3.10 using predefined thresholds for P_{max} and P_{avg}. If hard collision exists, then move the robot according to the appropriate movement vector.
2. If calibration has been initiated, then use the calibration algorithm described in Section 3.5. If the calibration algorithm has not been completed and thus returns a valid movement vector, move the robot according to the movement vector and deactivate the PID controllers to avoid integral windup.
3. If $P_{avg}(D_{tracked}) < \xi_{track}$, i.e. no contact is detected on the tracked wall with a given threshold ξ_{track}, then deactivate PID controllers and move the robot slowly towards the $D_{tracked}$ direction (0.1 coefficient on max speed in the present thesis, with a $\xi_{track} = 0.05$).
4. Weight-based movement:
   4.1. If PID controllers are deactivated, then re-activate them and reset the integral values to 0.
   4.2. Calculate the movement vector based on the weight-based movement method described in Equation (3.19) with set $S_{wall} = \{\phi_{base}, \phi_{wall\_dist}, \phi_{forward\_wall}, \phi_{dir}\}$.

3.12 Navigation algorithm

The navigation algorithm — outlined on a higher level in Section 3.3.2 and on Figure 13 — uses hard collision avoidance and calibration in the same way as wall-following. First, a check is done to halt movement if the robot has arrived at the destination. Then, basic checks are done just as in the wall-following algorithm to handle hard collisions and calibration. Then, a check is done to see whether wall-following could be used during navigation, to speed up navigation in unknown areas (introduced in Section 3.12.1). If all the mentioned algorithms do not return a movement vector, then weight-based movement is used to move the robot directly to the path node.

1. If the current position is the destination or no destination has been supplied, hold the robot still.
2. Check for hard collision using the algorithm defined in Section 3.10 using predefined thresholds for P_{max} and P_{avg}. If hard collision exists, then move the robot according to the appropriate movement vector.
3. If calibration has been initiated, then use the calibration algorithm described in Section 3.5. If the calibration algorithm has not been completed and thus returns a valid movement vector, move the robot according to the movement vector and deactivate the PID controllers to avoid integral windup.
4. If it is possible to use wall-following during navigation (see algorithm in Section 3.12.1), then use the movement given by the algorithm.
5. Weight-based movement:
   5.1. Deactivate wall-following PID controllers.
   5.2. Calculate the movement vector based on the weight-based movement method described in Equation (3.19) with set $S_{path} = \{\phi_{base}, \phi_{forward\_path}, \phi_{path}\}$.

3.12.1 Utilising wall-following

In order to improve navigation speed in unknown areas with obstacles, wall-following is utilised when applicable. The way this is achieved can be seen in Figure 23. If a wall is detected on the left or right side and the next path node is within the thresholds, wall-following can start.

For the thresholds, the open side threshold serves the purpose of not going too far with the wall-following if the path node is on its (hopefully) free side. The wall side threshold serves the purpose of redirecting the robot if the desirable path node is in the other direction. These thresholds can be tuned to achieve optimal performance, but for the present thesis the open side threshold was set to $\xi_{open} = 40°$ and the wall side threshold to $\xi_{wall} = 150°$.

An example of both the open side and wall side thresholds can be observed in Figure 24. Also, keep in mind that this is designed for times when the map is still unknown and true line-of-sight values are not known yet. First, on Figure 24a the robot has detected a collision on its right with the whiskers and thus can start following the right wall. Then it

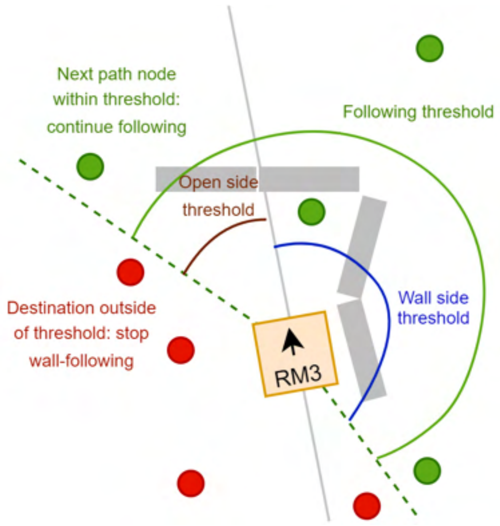
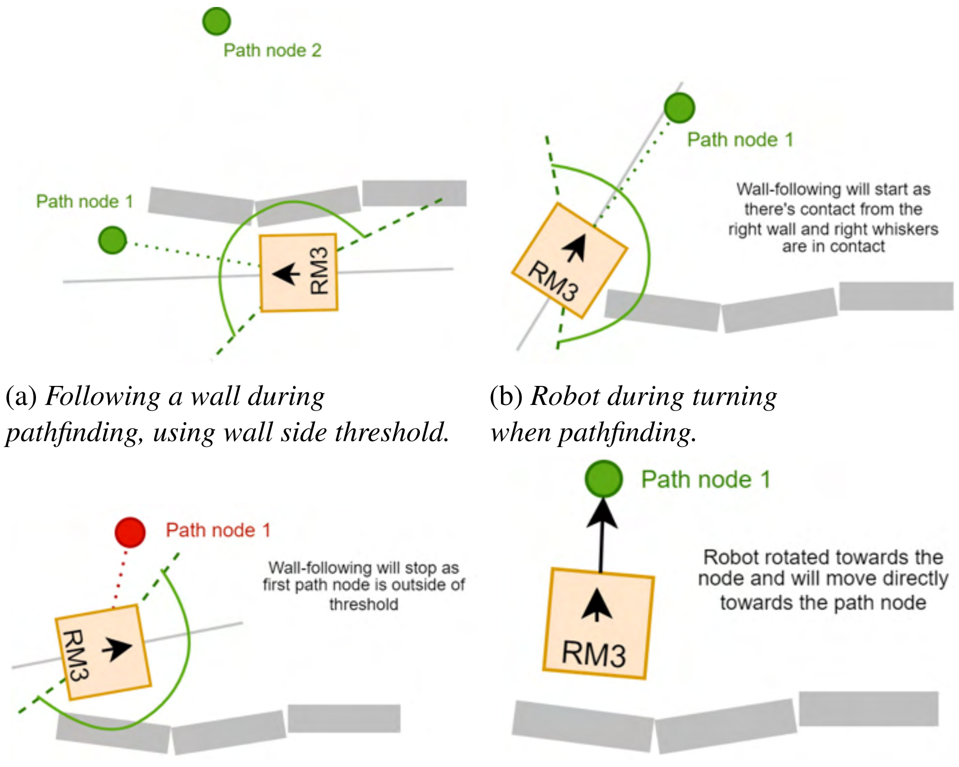


Figure 23. *Utilisation of wall-following during navigation for right-wall following.*

reaches the first node — causing the first node to get deleted and node 2 getting the index of 1 — and starts to rotate around the path node on Figure 24b.


Path node 2
Path node 1
RM3
Path node 1
Wall-following will start as there's contact from the right wall and right whiskers are in contact
RM3
Path node 1
Wall-following will stop as first path node is outside of threshold
RM3
Path node 1
Robot rotated towards the node and will move directly towards the path node
RM3

(a) *Following a wall during pathfinding, using wall side threshold.*

(b) *Robot during turning when pathfinding.*

(c) *First path node going outside the threshold after path recalculation.*

(d) *Stopping of wall-following due to no wall contact.*

Figure 24. *Example of utilising wall-following for navigation.*

Then, the robot goes past the threshold of the path node on its left on Figure 24c. Thus, according to our rules, wall-following can not continue and regular path planning movement is used, i.e. weight-based movement to reach the path node directly. The robot then rotates to its left and continues on to the path node, seen on Figure 24d. The full algorithm is the following:

1. If uninitialised, initialise $q_{path} = false$, $q_{wall\_follow} = false$, $cycles = 0$.
2. If no path can be calculated and there is side contact ($P_{avg}(D_{left}) \geq \xi_{avg}$ or $P_{avg}(D_{right}) \geq \xi_{avg}$), perform wall following, setting $D_{tracked}$ as the side direction with a larger $P_{avg}(i)$ value.
3. If $q_{align} = true$:
   3.1. If $|E_{path}| > \xi_{nav}$, then proceed with regular navigation.
   3.2. If $P_{avg}(D_{left}) \geq \xi_{avg}$ or $P_{avg}(D_{right}) \geq \xi_{avg}$, then also proceed with regular navigation, in order not to trigger wall following again until a transition to path following is complete (rotation and movement away from initial point).
   3.3. Otherwise, set $q_{path} = false$ and continue with the next step.
4. Calculate whether the left side is within the wall-following threshold, ($-\xi_{wall} < E_{dir} < \xi_{open}$), and whether the right side is within wall-following threshold ($-\xi_{open} < E_{dir} < \xi_{wall}$).
5. If $q_{wall\_follow} = true$ and the current tracked side $D_{tracked}$ is within wall-following threshold, then continue wall-following with the same $D_{tracked}$ direction.
6. If either left or right side have contact and the side with a higher $P_{avg}(i)$ value is within wall-following threshold, then wall-following can start. Set $D_{tracked}$ to the side with the higher $P_{avg}(i)$ value. Also set $q_{wall\_follow} = true$ and temporarily deactivate path planning PIDs to avoid integral windup. Then execute the regular wall-following algorithm.
7. If $q_{wall\_follow} = true$ or $D_{tracked} \neq 0$ and there is side contact, then cancel wall-following and rotate towards the next path node, i.e. set $D_{tracked} = 0$, $q_{wall\_follow} = false$ and $q_{path} = true$. Then execute weight-based navigation.
8. If $D_{tracked} = 0$, there is front contact ($P_{max}(D_{front}) \geq \xi_{max}$) and no contact on the sides, then an attempt to rotate the robot to a suitable wall-following position is made. If $E_{dir} \geq 0$, set $D_{tracked} = D_{right}$, else set $D_{tracked} = D_{left}$.
9. If $D_{tracked} \neq 0$:
   9.1. If $cycles >= \xi_{cycles}$, then the threshold amount of cycles of direct movement has been exceeded. Clear α_{goal} if set, and set $cycles = 0$.
   9.2. If α_{goal} is not set, then set $\alpha_{goal} = \alpha_{robot} + 30$ if $D_{tracked} = D_{right}$, else set $\alpha_{goal} = \alpha_{robot} - 30$.
   9.3. Get the current error between the temporary goal orientation: $\Delta_{goal} = \alpha_{goal} - \alpha_{robot}$.
   9.4. If $|\Delta_{goal}| \geq \xi_{nav}$, then set $cycles = 0$, deactivate all PIDs and turn the robot to reduce the error, i.e. towards left if $\Delta_{goal} < 0$ and right otherwise.
   9.5. If $|\Delta_{goal}| < \xi_{nav}$, update $cycles = cycles + 1$ and run the regular wall-follower

algorithm. Thus it will try to find a wall in the $D_{tracked}$ direction until it finds one or the amount of cycles run out.

10. If previous steps did not return a movement vector, then wall-following cannot be currently performed and regular path following is used.

Several thresholds are defined for the algorithm. The temporary goal orientation $\xi_{nav} = 5°$, the $P_{avg}(i)$ pressure threshold for a contact $\xi_{avg} = 0.05$, the $P_{max}(i)$ pressure threshold for a contact $\xi_{max} = 0.05$ and the amount of cycles $\xi_{cycles} = 10$ to attempt to find a wall (by performing wall-following). In addition, q_{path} is a variable to transition from wall-following to path following when the angle of the robot is not in the wall-following threshold any more.

The full algorithm can be considered relatively complex compared to the logic of weight-based movement. The main reason behind this is that there are quite many corner cases that can arise when transitioning from wall-following to path following and back, usually resulting in perpetual loops. One example of this is when the robot is rotating away from the wall in order to follow a path and then encountering a wall again, forcing it to rotate back towards the wall (as now it is within wall-following threshold). This example was solved with Step 3.2.

3.13 Tuning PID controller parameters

There are three PID controllers used in the algorithms in total: for E_y, E_{dir} and E_{path}. For these errors, having a PID controller in between the input and the output of the error increases the stability significantly compared to using a proportional controller with gain 1. In addition, for E_y a setpoint can be set to ensure contact with the wall is achieved.

For tuning PID controller parameters, a manual approach based on oscillation was used, however the starting parameters were taken with the approach mentioned in Section 2.7, i.e. increasing proportional gain until the system starts to oscillate and then setting values based on a formula presented in [33, p. 160]. As the PID controls are linked and also the value itself modified on each cycle due to the weight-based approach, then these values were always reduced even further.

As the PID controls affected each other, each error was first tuned separately, without involving any other movement. Then, the movements were combined one-by-one. So, for wall-following, first the combination of ϕ_{base}, ϕ_{wall_dist} and $\phi_{forward\_wall}$ were tuned together before adding ϕ_{dir} and tuning all PID controllers together depending on the resulting behaviour. The resulting PID controller values can be seen in Table 6. The setpoint for E_y was either 0.2 or -0.2 depending on whether the tracked side was left or right.

Table 6. *Tuned PID controller values.*

Error type	K_p	K_i	K_d	**Setpoint**	**Saturation**
E_y	1.0	0.5	0.2	± 0.2	$[-0.5, 0.5]$
E_{dir}	7.5	1.5	0.5	0.0	$[-5, 5]$
E_{path}	0.5	0.1	0.2	0.0	$[-40, -40]$

It was also observed that the PID controller values that were tuned in simulation were slightly suboptimal in real-world tests. As only one real-world environment for tests was used, then different terrains could also have different optimal PID controller values. Thus this was considered as one avenue for more research in the future.

3.14 Control application

For the control mechanism, a terminal-based application was created to allow changing the parameters of the pathfinding algorithm in real-time, for testing both in simulation and real life. The application allowed to:

- select (and also change in real-time) one of the following pathfinding algorithms:
  - left wall following;
  - right wall following;
  - waypoint-based A* algorithm;
  - waypoint-based Theta* algorithm.
- change the destination point of the navigation algorithm (for waypoint-based algorithms);
- change the velocity of the navigation;
- turn the navigation algorithm off (killswitch).

4. Results and discussion

In the following chapter the results of simulation and real-world testing are described, including the method with which the algorithm was tested, the measurements taken and the changes the tests resulted in.

4.1 Overview of experiments

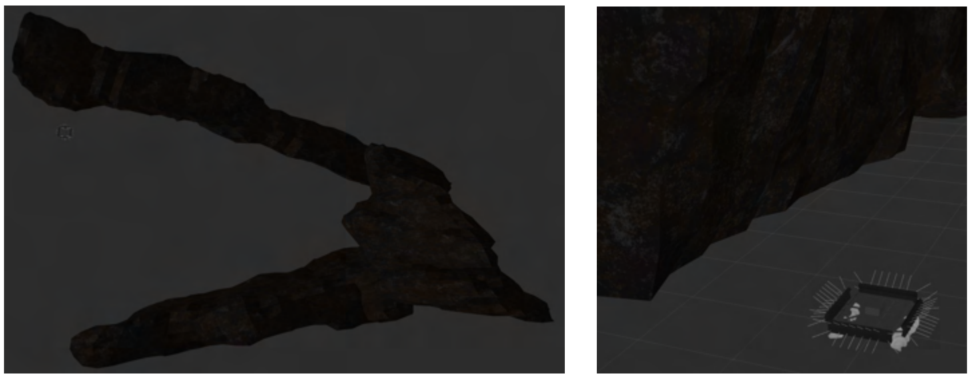
Continuous testing in the simulator was done during the development of the algorithm and three real-world experiments were carried out for validation. This section will explore the methodology with which the simulated and real-world experiments were carried out and analysed.

4.1.1 Simulation with Gazebo

As robots are often easier to test and develop in simulations, then a simulation environment for the RM3 had been previously built. The simulator that was used was Gazebo. Gazebo is an open-source 3D simulator with a physics engine that also supports relatively realistic sensor simulation, including cameras and laser range finders. With additional programming it is also possible to create custom sensors, as was previously done for the whiskers as part of the ROBOMINERS project. In addition, Gazebo interfaces very well with ROS 2, the chosen operating system for the RM3 robot.

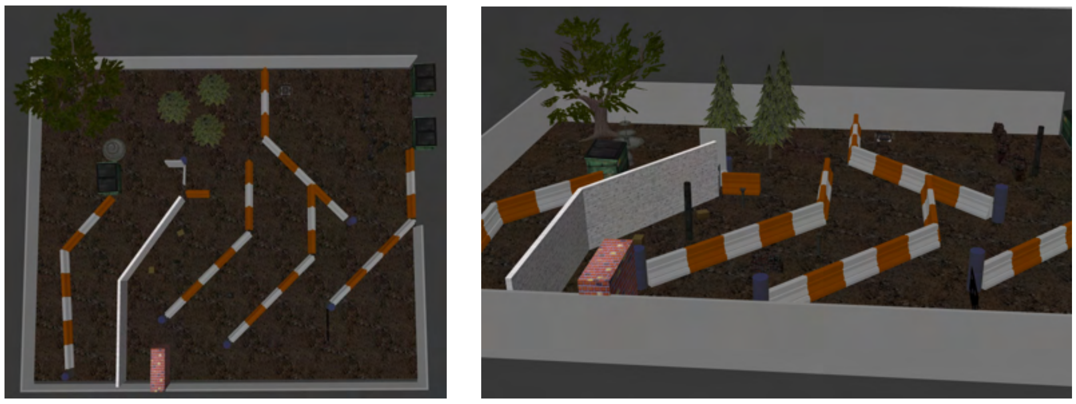
In Gazebo, the RM3 could be placed in different environments to allow testing the performance of the algorithms in various conditions. The environments used in the thesis were created using Blender. Environment 1 (see Figure 25) was a cave-like environment with walls that were curved and had many different facets. Environment 2 (see Figure 26) was an unpredictable city-like environment containing sharper, more defined objects like barriers, trees, water fountains and garbage containers.

Initial tests while developing the algorithms were done on environment 1 due to more cave-like features. However, with complex objects like the cave walls with a complex layout, problems started to appear with collision of small objects like the whiskers. The main issue was the whiskers going through walls without colliding and thus passing faulty information to the robot as a result. As environment 2 had very few such issues, then environment 2 was the main environment used for tests, and environment 1 was mostly



(a) *Top-down image of simulated environment 1.* (b) *Closer view of the simulated environment 1.*

Figure 25. *Layout of simulated environment 1.*



(a) *Top-down image of simulated environment 2.* (b) *Simulated environment 2 at an angle.*

Figure 26. *Layout of simulated environment 2.*

used to verify PID values for cave-like walls.

Due to computation constraints in the simulator, the whiskers were stiff, unlike their real-world counterparts. Thus no amount of the recorded whisker's deflection was lost to the flexible whisker in simulation, which slightly modified the whisker pressure results. In addition, no detailed simulation of the surface was done, meaning that if in real-world tests the locomotion screws could dig up some of the ground and thus hinder locomotion, then in simulation this was not a factor. In addition, as described earlier, artificial noise and bias was added to the whiskers in simulation in order to simulate noise that would be present in real-life measurements. Measurements were taken from real-life tests to provide the best simulation possible, as filtering the noise impacts the response time of the algorithm, which in turn require different PID values for weight factors.

4.1.2 Real-world experiments

In order to test the suitability of the algorithm in real-world use for the RM3 robot, three experiments were conducted. Different environments were created for testing the algorithm, taking into account various challenges that were encountered in simulation. The environments were created using available objects: bricks, styrofoam walls and a large flower pot. External camera-based pose estimation was used for measuring the pose and analysing the trajectories of the algorithms. In addition, for navigation the same pose information was also passed as input, as described in Section 3.2.

The environments (see Figures 27-31) were created specifically to take into account different failures that initially occurred during testing in simulation. This includes areas for testing undershoot, i.e. when the turn is too sharp and the robot overshoots into empty space. These can be seen on the left side of the environment. Also another situation is the potential of driving into a wall at a sharp angle, which can be seen at the top half part of the environment.

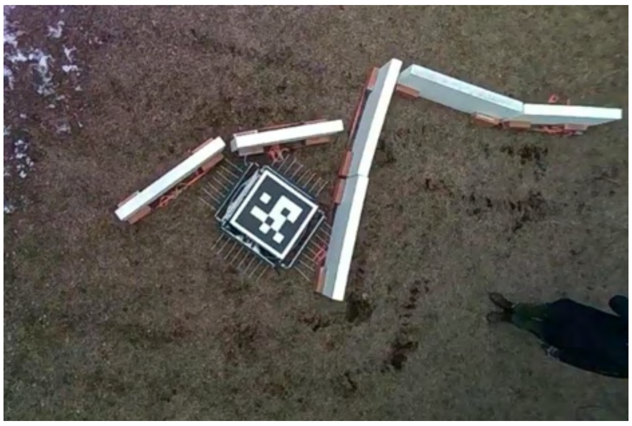


Figure 27. *Top-down angle of the environment used for the first test.*

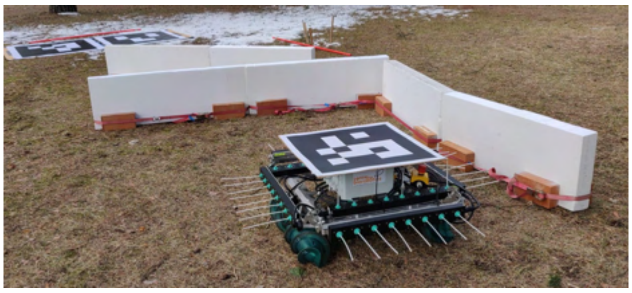


Figure 28. *Close-up angle of the environment used for the first test.*

4.1.3 Analysis

For validating the performance of the algorithm, data originating from either the RM3 object in the simulator or the data originating from the external camera-based pose estimation

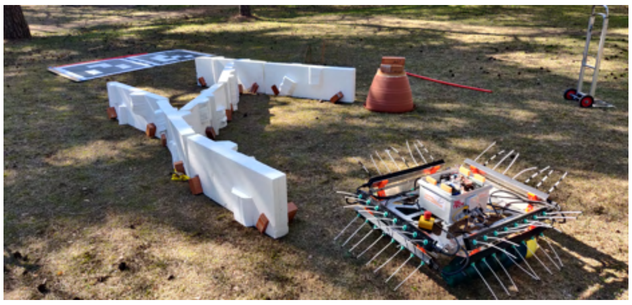


Figure 29. *Close-up angle of the environment used for the second test.*

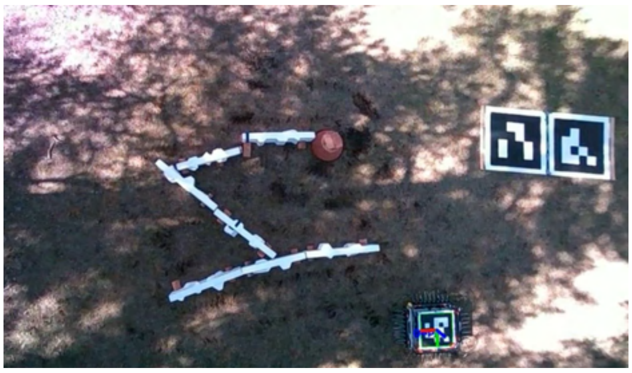


Figure 30. *Top-down angle of the environment used for the third test.*

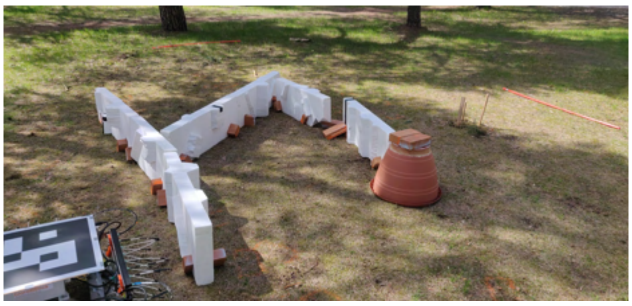


Figure 31. *Close-up angle of the environment used for the third test.*

was used. Then, the traversal time and total distance were calculated. For calculating theoretical fastest times, the fastest forward speed and the fastest turning speed were obtained by having the robot only do the specific movement, and then by taking the average of the movement speed of 100 samples. The obtained maximum speeds are presented in Table 7.

Table 7. *Forward and angular speeds used for calculating the theoretical best traversal time.*

Movement type	**Simulation**	**Real-world experiment**
Moving $\left(\frac{m}{s}\right)$	0.18	0.053
Turning $\left(\frac{\text{deg}}{s}\right)$	22.50	6.45

The theoretical fastest traversal for a given path itself was calculated by taking the path of the robot — averaged over 100 samples for real-world experiments, not filtered for simulation — and then dividing it with the maximum forward speed of the robot, presented in Table 7. In addition, turning was also included by calculating the amount of turning needed to pass the course and dividing it by the maximum turning speed. As this calculation does not include physical factors like acceleration or surface conditions changing, then this is of course realistically unachievable. However, comparable results can be achieved. These results could then be compared with the results of the algorithm to provide a metric for the algorithm's performance.

In addition, for navigation, an additional theoretical best was calculated where the whole environment had been explored (and thus the theoretical best path was also known). In some cases, especially for simpler cases, the algorithm could find a similar path without the map explored. However, in worse cases, the differences could be larger. There is a theoretical limitation to this when the map cannot be known in advance. However, algorithms can be tuned to be more likely to choose the correct path, especially if some parameters of the environment and likely paths are already known.

4.2 Modifications for real-world use

In total, three separate real-world experiments were carried out in order to verify the algorithm working in a complex environment. Experiment 1 encountered issues due to different aspects not being considered. These issues are outlined in this section. Experiment 2 encountered issues relating to the robot's locomotion hardware. As the robot is a prototype and novel in several ways, then this was however to be expected, and will not be covered due to the focus of the present thesis being on the algorithm implementation.

The present section also confirms the thought mentioned in Section 2.4 for the work the author considers the closest current research to the present topic — that there are often major differences between simulation and real-world use that also require changes in the overall principles of the algorithm. An example of this is whisker bias, which if not dealt with in the algorithm could make the algorithm unusable.

4.2.1 Whisker configuration

One issue that was confirmed during experiment 1 was the issue of blind spots for the robot. Although this at times also appeared as an issue in simulation, then in simulation this issue was not as severe. One major reason for this is that the cables required for connecting the whisker arrays to the main electronics compartment are not simulated. As the connector cables have been engineered to be at the end of the whisker arrays, then blind spots being at the same place was a major problem for wall-following and navigation (see Figure 32). Though the cables used were relatively rigid, then these would not stand up to long-term use. In addition, the connectors would of course ideally not be on the side in the future.

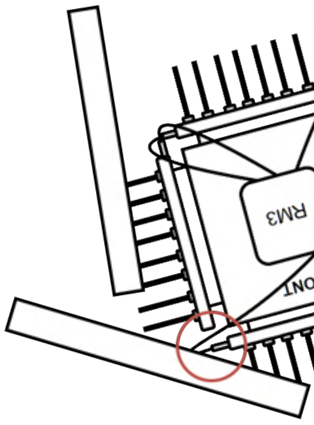
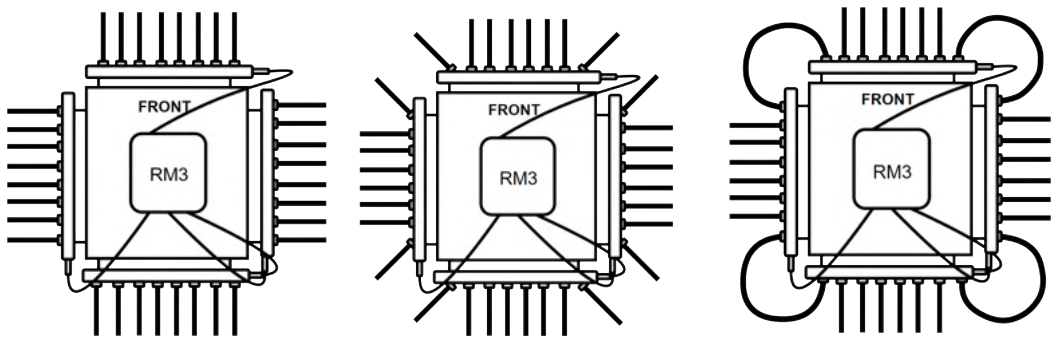


Figure 32. *Blind spot for the default whisker configuration.*

Three different approaches were tried for whiskers in regards to the blind spots, as seen on Figure 33. The default whisker configuration is the configuration with the posed issue. The diagonal edge whiskers were used only in simulation (with good results), as the currently existing whisker arrays do not support having the whisker sensors at an angle. In this case the edge whiskers themselves in theory could have been bent at such an angle that it would have given a similar outcome, but this was not tried due to the modified whiskers likely having more issues with bias in such a case.



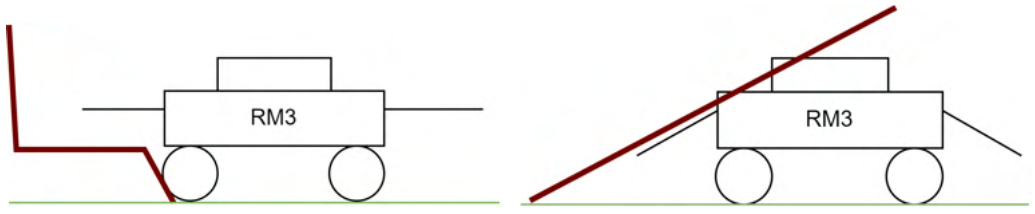
(a) *Default whisker configuration.* (b) *Diagonal edge whiskers used in simulation.* (c) *Connected edge whiskers used in real-world experiment.*

Figure 33. *Whisker configuration approaches to mitigate blind spots.*

The third variant — connected edge whiskers — was not used in simulation due to the required effort of producing such a whisker modification in simulation. However, these were tried out in the experiment 1 by using heat-shrink tubes to connect a regular whisker to a much longer whisker on the other side. This configuration needed no change in the algorithm due to the way bias is accounted for, as described in Section 3.4.2.

In theory, the connected edge whiskers configuration should have performed well, especially with walls, as they provided full coverage of the blind spots on the plane they were situated on. In tests however, this configuration introduced new issues, as the friction between the styrofoam wall and the whiskers' grooved plastic material was high enough that following a wall in a stable manner was not possible. For other whiskers, mostly just the ends of the whiskers touched the styrofoam walls, thus it was not a major problem. Despite this, the connected edge whisker configuration was the best candidate of covering the blind spots without large modifications to the whisker arrays. Thus this approach was used for experiment 3, with the connected edge whiskers covered in tape to reduce friction.

In addition, the vertical placement of the whiskers was also explored. Different configurations were first first tested out in simulation. If the demands are to only follow a wall with the environment being very controlled (no debris, straight walls and even floors), then one array could be considered enough. However, if there are uncertainties (see Figure 34), then there are blind spots on the vertical side as well, which need to be taken into consideration. In case of just having level or ascending whisker arrays, lower obstacles would cause issues for navigation. When only having descending whisker arrays however, then a ceiling with a small slope could damage the main electronics compartment, as no obstacle is detected in that case.



(a) *Main blind spot for a level single-array setup.* (b) *Main blind spot for a descending single-array setup.*

Figure 34. *Whisker configuration approaches to mitigate blind spots.*

As was observed, just one whisker array is not enough for unpredictable environments. Thus a more universal whisker setup is proposed in Figure 35. It avoids the blind spots of level and descending single array setups by incorporating two arrays: one ascending and one descending. There remain two relatively unlikely and small blind spots. For solving these, different variants can be proposed, with the easiest theoretical solution being to add more whisker arrays. Another way of solving this would be to have every whisker pair (i.e. whiskers equally spaced from the center of the array) at a slightly different angle to cover the blind spots vertically. One issue with this approach would be whisker bias, changing the angles over time. Due to the blind spots being small and improbable, any additional mitigation of this was not explored.

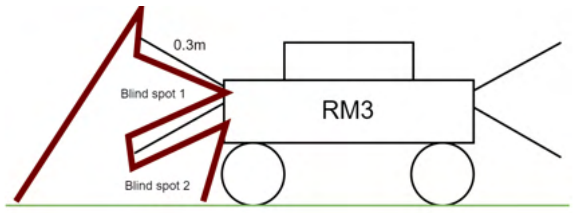


Figure 35. *Proposed universal whisker setup with its potential blind spots.*

For real-world testing, initially only one array was used on each side, with the possible angle between $-30°$ and $0°$. Later, custom mounting brackets for the arrays were manufactured with a 3D printer in order to attach multiple whiskers at an angle (see Figure 36). Initially, a $-30°$ and $30°$ configuration was used for all sides, as in Figure 36. However, as the walls used for testing were too short, then final testing was done with the whisker arrays at $-30°$ and $0°$ instead to obtain more realistic results.

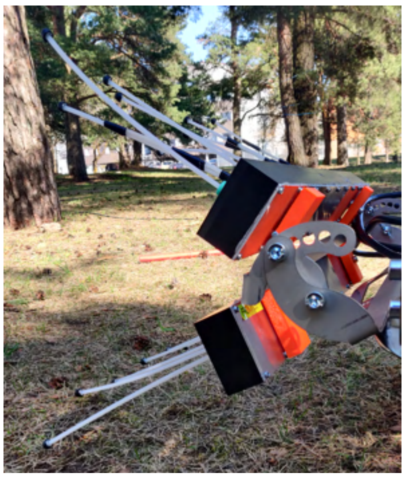
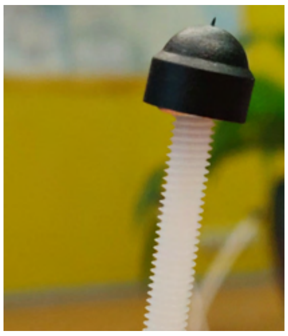


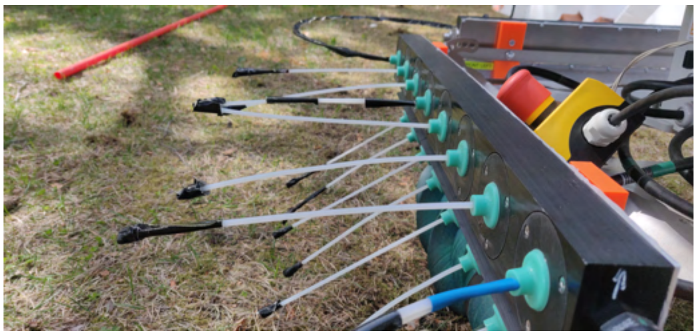
Figure 36. *Method of attaching whiskers at an angle.*

4.2.2 Whisker friction

In Section 4.2.1, the connected edge whiskers were described to have a large amount of friction with the styrofoam wall. Although not as big of an issue, then regular whiskers had some issues with friction as well. As the tips of the whiskers were cut with wire cutters and did not receive any further treatment, they were relatively sharp. Combined with the fact that the whisker material itself was relatively stiff and thus required quite a large amount of force to bend (and thus detect pressure), the friction for the tips of whiskers was also relatively large. This in turn hindered smooth wall-following, as the whisker could usually bend further due to the large friction force holding it against the wall.

For experiments 2 and 3, this was partly mitigated by gluing a plastic end cap onto the whisker (see Figure 37a). The grooves of the whiskers can also be seen on the same figure, which are the main culprit in the large amount of friction. The oversized end cap depicted is however not ideal either, as it could get stuck on some types of obstacles due to the sharp edge. In order to improve upon this even further, gaffer tape was thus applied to the ends of whiskers (see Figure 37b) in order to smooth out the edge between the end cap and the whisker. As the connected edge whiskers were the most affected by friction, tape was added for most of the entire part as well (see Figure 37c). Despite the connected edge whiskers still showing signs after this of slightly getting stuck due to friction, it was a major improvement.


(a) *Whisker grooves and end cap.*


(b) *Tape applied to the ends of whiskers.*

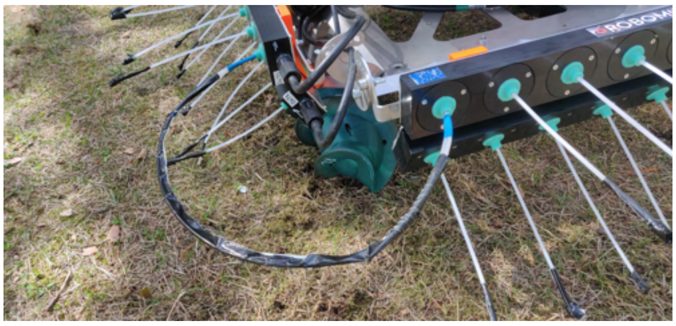

(c) *Tape applied to connected edge whiskers.*

Figure 37. *Mitigation of friction between whiskers and walls.*

These measures are not a long-term solution by itself, only a temporary improvement due to the nature of the present whiskers. However, these grooves are also not unnecessary — they are present as they have the ability to be screwed into the whisker array. This experiment however shows that the grooved main body of the whisker causes issues. Thus in the future these grooves should be eliminated from the outer part of the whisker. One option would be to cast the entire whisker with the base included, instead of having to connect the whisker with grooves. As there have also been issues with the grooves in the base wearing out, this would be one way of solving that problem.

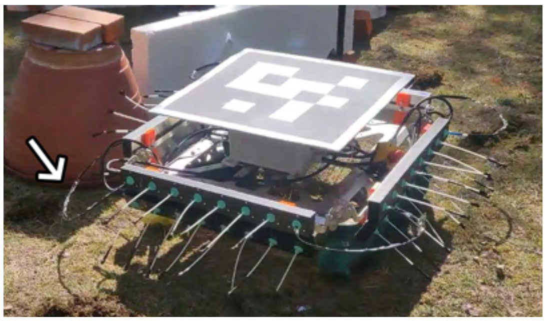
In relation to the shape of the whisker, different materials and shapes of the whisker itself could also prove beneficial. The whiskers used for the robot were of uniform thickness, however a tapering whisker was used in the cockroach-inspired wall-follower [19] with good results. In addition, whisker-based sensing research covered in Section 2.3 also seemed to use significantly more flexible whiskers that were also smaller in size. Of course, stiffer whiskers have the benefit of producing less noise, in addition to being less affected by the robot's own movements and the movements of wind or water. Thus, this is still an additional avenue for improvement and optimisation.

4.2.3 Whisker bias

As mentioned in Section 3.4.2, whisker bias had to be handled. As seen on Figure 37b, the whiskers always have bias to some extent. However, in cases the whisker is displaced to a larger extent, then the bias can also change during the execution of the algorithm, especially so if there are lots of hard collisions. This is accounted for in the algorithm through recalibrating, which for the final tests was done every 60 seconds. However, this also means degrading performance of the whisker in most cases, as it cannot detect objects anymore from as far away from the wall as possible.

This bias was most noticeable on the connected edge whiskers, especially in front on the side which the robot tracked. As the connected edge whisker was constantly being displaced up and down in different cases of obstacles, the bias accumulated very quickly. An example of such a connected edge whisker can be seen in 38a.

In order to reduce the accumulating bias, a setup of two small strings of fishing line were used in order to attach the centre of the connected edge whisker to the robot's frame (see Figure 38b). One of the strings was attached onto a higher part and the other onto a lower part of the frame than the whisker. This was done to limit the motion range of the connected edge whisker to desired positions. The fishing lines were pulled to be slightly more than taut. This enabled the whisker bases to be at a higher angle to the whisker array,


(a) *Accumulated bias in connected edge whisker on the left.*

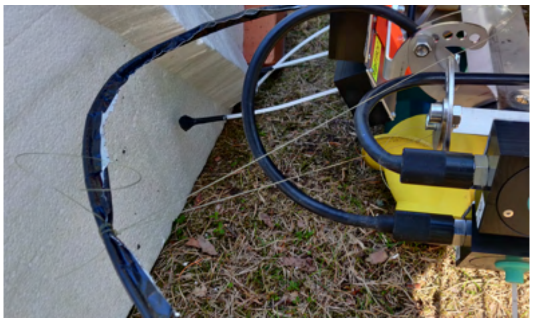

(b) *Fishing lines used for mitigating bias.*

Figure 38. *Mitigation of accumulating bias in connected edge whiskers.*

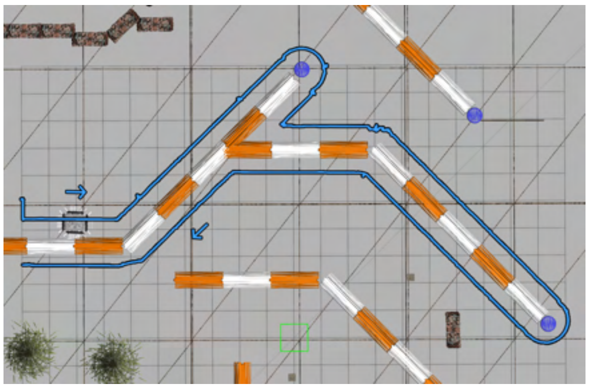
thus increasing the sensitivity to pressure for the connected edge whisker.

4.3 Wall-following results

In the following section the results of testing the final wall-following algorithm in both simulation and real-world experiments are presented and discussed. As previously mentioned, final results only consider the final algorithm, i.e. the algorithm outlined in Chapter 3. Thus the outlined results originate either from the latest simulation tests or from experiment 3.

4.3.1 Simulation

First, the results of wall-following are presented, being the simplest algorithm of the three. As can be seen on Figure 39a, the trajectory of the Path 1 can be considered relatively optimal in simulation. Most of the small bumps in the trajectory are spots where calibration took place. Though this is not required for simulation (as accumulating bias was not simulated), it was still done for the results to be more comparable to real-world cases. However, this is a small part of the optimality difference.


(a) *Full path.*

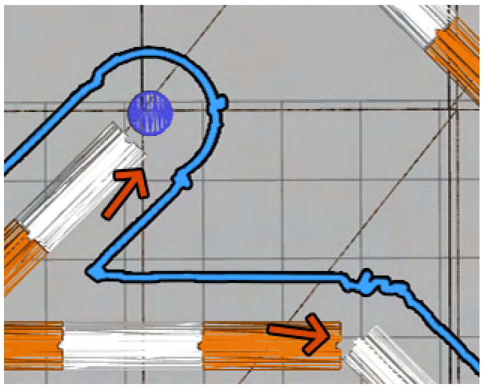

(b) *Areas with gaps in the wall.*

Figure 39. *Path 1 of the wall-following algorithm in simulation.*

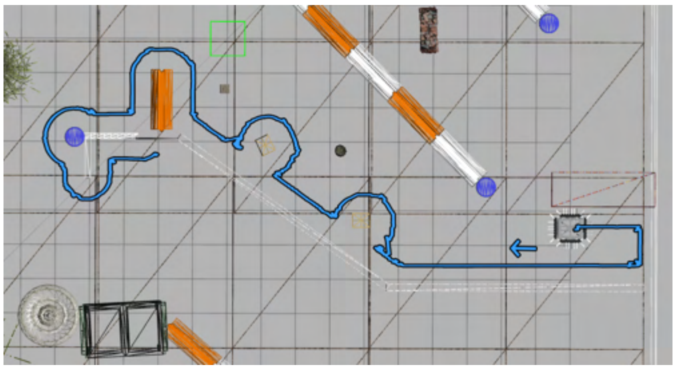
Most of the difference between the algorithm's traversal time and the theoretical best traversal time (see Table 8) came down to the spots where there were gaps in the wall, such which in this path were entirely in the spots outlined in 39b. In addition, the rotation in very sharp corners also causes quite a bit of losses between the two. Decreasing the amount of time the robot takes to move through these obstacles is a balance between speed, reliability and smoothness. The slower the robot moves, the smoother the path, as the chance of losing track of the wall is smaller. Thus the traversal time could likely be reduced at the cost of a less defined path with more oscillation.

Path 2 for wall-following was chosen to involve slightly more complex objects (see Figure

Table 8. *Performance of the regular wall-following algorithm for Path 1 compared to the theoretical best case.*

Description	**Traversal time (s)**	**Traversal time (%)**	**Distance (m)**
Theoretical fastest traversal for the wall-follower path	363.04	100.00%	59.71
Wall-follower	505.40	139.22%	59.71

40a). This included right angle turns, some less thick walls and also dynamic boxes which moved when a considerable amount of pressure was applied on its sides. In addition, wall-following was performed on the left. Though theoretically the same algorithm can be used for both sides (with the small setpoint modification in E_y and change in $D_{tracked}$), this was done to showcase that the algorithm is indeed able to follow walls on both sides.


(a) *Full path.*

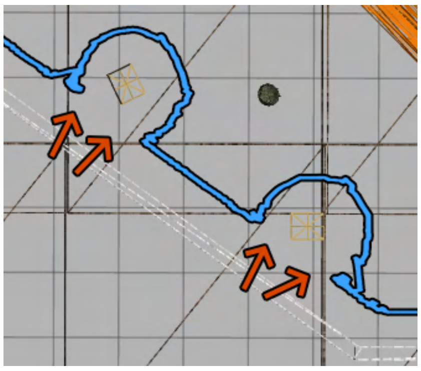

(b) *Areas with sharp corners.*

Figure 40. *Path 2 of the wall-following algorithm in simulation.*

As can be seen from the traversed path, the wall-follower followed the wall and obstacles in a relatively smooth way. However, a lot of the time difference compared with the theoretical fastest traversal of the path (see Table 9) came down to the sharp corners. For the first box on the right (see Figure 40b), the robot moved towards the wall ignoring the small pressure on its front-right corner initially. Then, it took some time to turn, changing between turning towards its right and avoiding the occurring hard collisions on its left, rear and front sides.

For manual control and for cases where the location of the robot, the wall and the obstacle are precisely known, the robot could just reverse to a given distance, turn right and continue following the box. However, the challenge here is to do this blind and avoid hard collisions. Thus, close contact is needed with the tracked direction or on the two sides perpendicular to it.

Table 9. *Performance of the regular wall-following algorithm for Path 2 compared to the theoretical best case.*

Description	**Traversal time (s)**	**Traversal time (%)**	**Distance (m)**
Theoretical fastest traversal for the wall-follower path	290.77	100.00%	42.22
Wall-follower	565.15	194.36%	42.22

If these situations would be frequent, the rectangle outer shape of the robot or the whiskers could be changed to be more rounded. An ideal case for turning would be for the outer shape to be a circle. Thus turning would not induce additional pressure and only change the locations of the pressures around the robot, making it ideal for turning. However, there are of course other tradeoffs for this, e.g. smooth wall-following becoming more difficult with the mentioned configuration, and likely increased costs and complexity in manufacturing.

Another issue that was also present for Path 2 was that at the termination point was an obstacle that was uncrossable due to the combination of the algorithm and the whisker setup (see Figure 41). As the algorithm prioritises a certain distance from the wall and also a total direction sum of 0, then in this case the robot gets stuck. This is due to the large hole similar to the length of the whisker arrays and the thin wall. Thus, the robot started to slowly oscillate between directions.

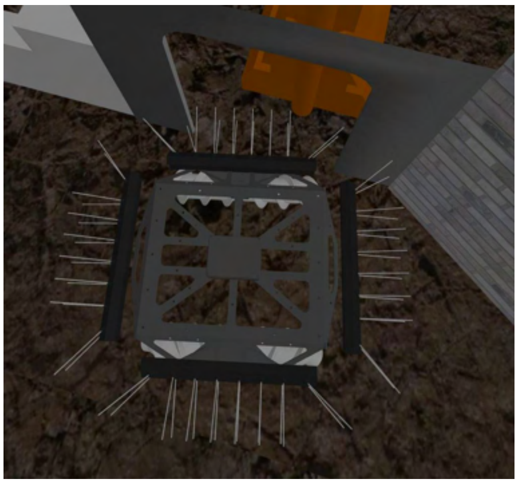


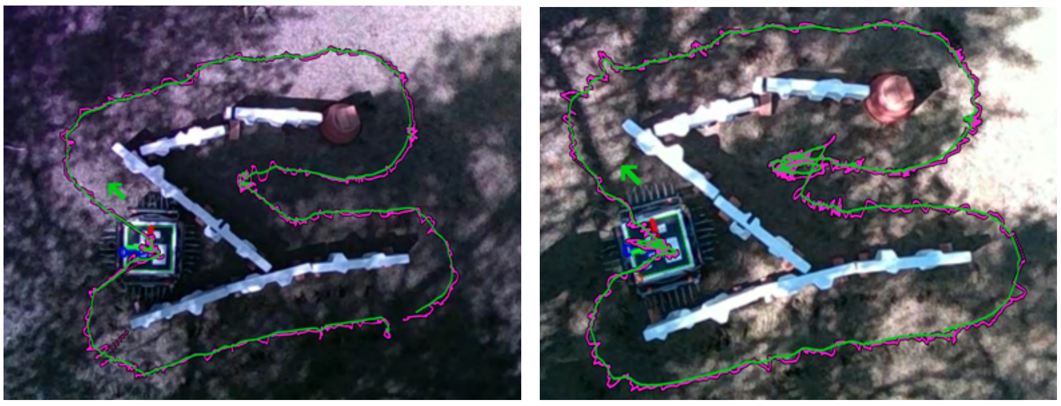
Figure 41. *Example of an uncrossable obstacle for the wall-follower.*

There is reason to believe this would not occur with the connected edge whisker configuration that was used for real-world experiments. Instead of the wall on the top-right being stuck between two edge whiskers, the wall would instead apply pressure to the connected edge whisker and also provide the E_y threshold for the robot to move forward.

Based on the above, the simulator tests confirmed that overall the wall-following algorithm was reliable, but inoptimal at times. In addition, the simulations showed that issues with the simulated whisker configuration can occur.

4.3.2 Real-world experiments

For the real-world experiments, difficult cases from the simulator were carried over into real-world environments for verification in real use. Path 3 was navigating around the whole environment. The tests were carried out with two speed settings of the algorithm. This meant that all of the published movement vectors were multiplied with a constant of either 1.0 or 1.5. The resulting trajectories the robot moved in for both cases can be observed on Figure 42. As there was significant noise in measuring compared to simulation, the pink line signifies the path with positions averaged over 10 readings, and the green identifies positions averaged over 100 readings. The same is true for all real-world experiments.



(a) *With speed setting 1.0.* (b) *With speed setting 1.5.*

Figure 42. *The path of the wall-following algorithm in experiment 3 with two different speed settings.*

The real-world experiment highlights the same as the previous simulation tests: that the main factor in a lower traversal time compared to the theoretical best (see Table 10) is navigating around sharp corners. As can be seen for both, the main positions with less smooth movements are in the sharp corners, especially so for the 1.5 speed setting variant. This is a result of the same behaviour as seen in simulation for Path 2: the need to rotate around while avoiding hard collision, but still keeping in contact with the tracked side or in its absence the two sides perpendicular to it.

In addition to the inoptimality pointed out in the simulator, the robot also struggled a bit with locomotion. This was especially visible on smooth turns, like the one around the upside down flower pot visible on the top-right in Figure 42. For these movements there were situations where two of the locomotion screws stood still and only acted as breaks

Table 10. *Performance of the wall-following algorithm at two speeds compared to the best theoretical result.*

Description	**Traversal time (s)**	**Traversal time (%)**	**Distance (m)**	**Distance (%)**
Theoretical fastest traversal	531.49	100.00%	19.42	100.00%
Right wall, speed 1	1304.60	245.46%	21.40	110.19%
Right wall, speed 1.5	1416.30	266.48%	25.59	131.80%

for the other two screws, slowing down the overall movement and also causing more damage to the top part of the surface than usual. This was not strictly an issue with the present algorithm, but instead with the locomotion algorithm that converted the published movement vectors to motor speeds. However, it still added to the increased traversal time of the algorithm.

Overall, the real-world experiments confirmed much of the same as the simulator: that overall the algorithm was reliable, but inoptimal for sharp angles. In addition, issues with the locomotion algorithm converting the algorithm movement to motor speeds also increased the traversal time for real-world experiments.

4.4 Navigation results

In the following section the results of testing the final navigation algorithm in both simulation and real-world experiments are laid out and discussed. As previously mentioned, then final results only consider the final algorithm, i.e. the algorithm outlined in Chapter 3. Thus the outlined results originate either from the latest simulation tests or from experiment 3.

4.4.1 Simulation

Firstly, simulation results are presented. One of the main hypotheses raised in the Chapter 3 was that of the wall-following Theta* offering better performance compared to regular Theta* for maps with an unknown state. Thus, this was tested out in simulation as well. For Destination 1, as seen on Figures 43 and 44, this indeed seems to be the case when looking at the paths both algorithms took.

The wall-following variation results in a smooth path, needing to rotate very little against an unknown wall, whereas the regular Theta* algorithm needs to turn $180°$ for every traversed square for a diagonal wall as seen on Figure 44. However, this also depends on

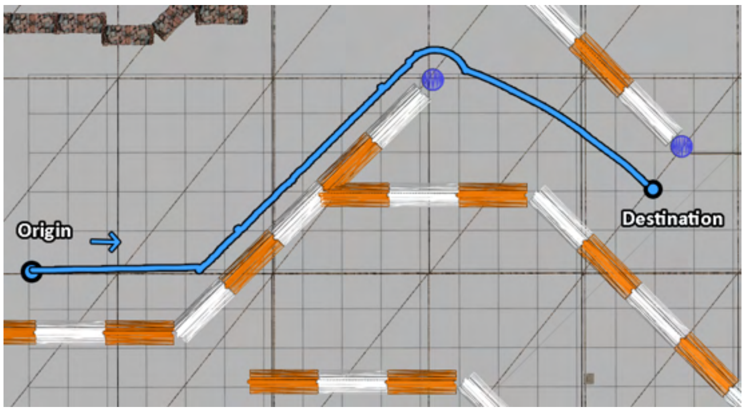


Figure 43. *Navigation for Destination 1 in simulation using the final Theta*-based algorithm that utilises wall-following.*

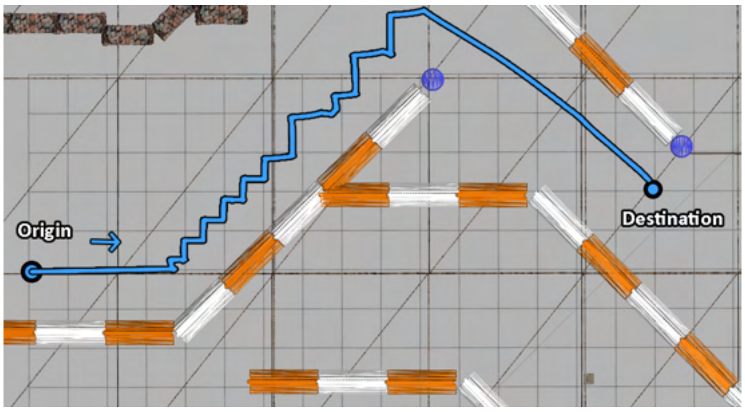


Figure 44. *Navigation for Destination 1 in simulation using the final Theta*-based algorithm that does not utilise wall-following.*

having an optimal algorithm in place to engage and disengage the wall-following. For Destination 1 this algorithm looks optimal: the line is smooth when following the wall and also the wall-following is stopped almost as soon a direct line-of-sight appears between the robot's location and the destination.

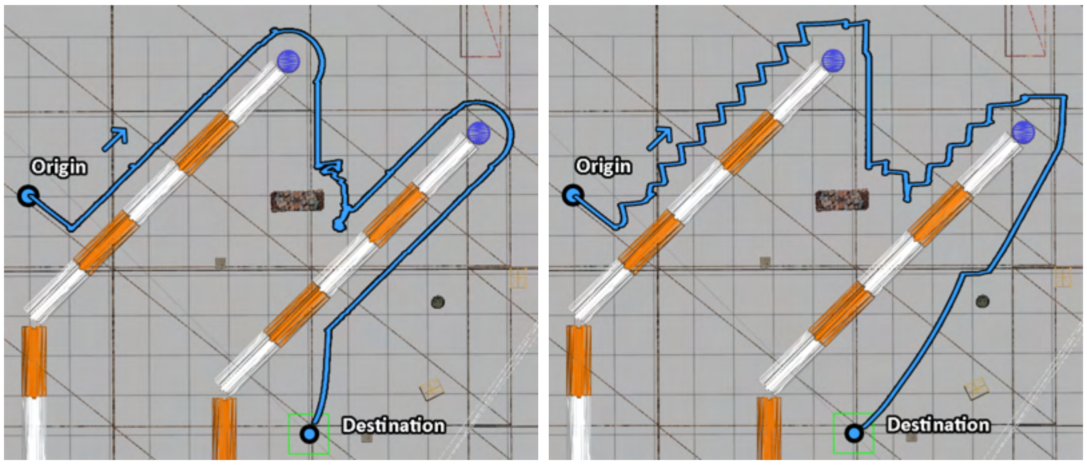
When looking at the traversal times and distances (see Table 11), then a similar conclusion can be made as when looking at the paths qualitatively. When not utilising wall-following, the distance increased by only 14.77%, however the traversal time improved by 42.97%. As turning is time-consuming, then this is an expected outcome.

For Destination 2 (see Figure 45), a point was chosen to require a more complex trajectory. Though both start against a diagonal wall similar to the last example, the navigation gets more complex after doing the turn, as there is an rectangular obstacle in the way in addition to a sharp angle right next to it.

Similar to Destination 1, following the diagonal walls was one reason for a longer trajectory

Table 11. *Performance of Theta* vs Theta* wall-following in simulation for Destination 1 compared to the theoretical best cases.*

Description	**Traversal time (s)**	**Traversal time (%)**	**Distance (m)**	**Distance (%)**
Theoretical best (map explored)	97.75	83.13%	16.90	83.57%
Theoretical fastest traversal for the Theta* wall-following path	117.59	100.00%	20.22	100.00%
Theta* wall-following	164.55	139.94%	20.22	100.00%
Theta* without wall-following	288.35	245.22%	23.21	114.77%



(a) *Utilising wall-following.* (b) *Not utilising wall-following.*

Figure 45. *Navigation for Destination 2 in simulation using the final Theta*-based algorithm.*

and a longer traversal time for the algorithm not utilising wall-following. However, this time there was also very small potential benefit — it did not try to go between the rectangular obstacle and the wall and it happened that the fastest traversal path was not to go through. However, in cases where the robot is just small enough to fit through this, then the algorithm utilising wall-following could provide different solutions to the algorithm not utilising wall-following.

Looking at the traversal times and distances (see Table 12), the results could be considered similar to the results for Destination 1. For Destination 2, utilising wall-following improves the traversal speed by 26.39%. However, due to the increase in the amount of sharp turns, the difference in time between the Theta* wall-following variant and the theoretical fastest time for the path was now significantly higher. This was also seen in tests in the wall-following algorithm overall in Section 4.3.

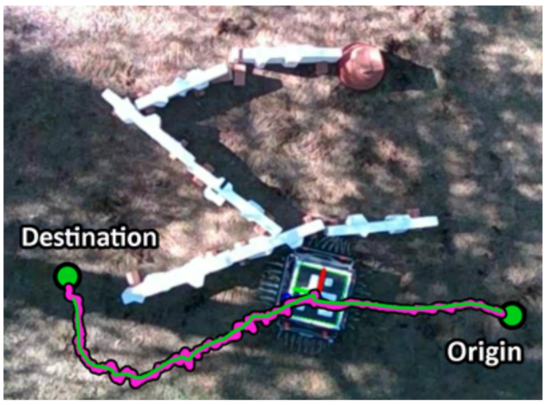
Table 12. *Performance of Theta* vs Theta* wall-following in simulation for Destination 2 compared to the theoretical best cases.*

Description	**Traversal time (s)**	**Traversal time (%)**	**Distance (m)**	**Distance (%)**
Theoretical best (map explored)	149.53	64.87%	24.67	72.55%
Theoretical fastest traversal for the Theta* wall-following path	230.52	100.00%	34.00	100.00%
Theta* wall-following	420.85	182.57%	34.00	100.00%
Theta* without wall-following	571.70	248.01%	36.21	106.50%

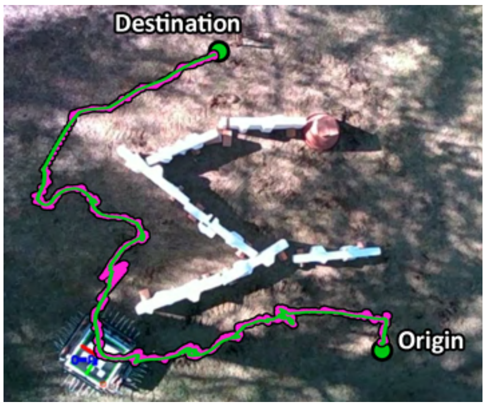
4.4.2 Real-world experiments

Due to time constraints and the significantly better performance of the wall-following variant of Theta* in simulations, only the wall-following navigation algorithm was tested in real-world experiments. Three different trajectories that were used in experiments are presented.

Firstly, two successful navigations are presented, Destinations 3 and 4 (see Figure (see Figure 46). Destination 3 was similar to the simulated Destination 1, and performed similarly as well when observing the path. Though there are some smaller deviations, these are mostly seen in the areas of the wall where additional curvature was added to make it more cave-like.



(a) *Destination 3, a simpler trajectory.*



(b) *Destination 4: a longer, more complex trajectory.*

Figure 46. *The paths of two experiments for the navigation algorithm utilising wall-following.*

Destination 4 on the other hand was a longer and more complex trajectory. Overall, it performed better than wall-following as well, allowing to skip the sharp corner on its right midway through. However, towards the end of the trajectory on the top left there was an inoptimality — the robot switched to regular navigation and went to a grid square, when it would have been more optimal to continue wall-following.

As previously mentioned, the optimality of the Theta* algorithm utilising wall-following depends largely on the algorithm that switches between wall-following and regular Theta*. Thus the thresholds, grid square size and a plethora of other variables could be altered for better performance that is tuned to specific environments. Further optimisations to the switching system are thus definitely an additional avenue for research.

Results for Destination 3 (see Table 13) show that the traversal time of the algorithm was 205.94% of the theoretical fastest traversal time. For the simulated Destination 1, the corresponding value for the same algorithm was 139.94%. Thus it can be concluded that the unsimulated real-world factors described at the end of Section 4.3.2 — the locomotion system and the addition of a surface material — increase the traversal time quite a bit when compared to simulation, even when no sharp angles were present as in Section 4.1.2.

Table 13. *Performance of Theta* wall-following algorithm for destination 3 compared to the theoretical best cases.*

Description	**Traversal time (s)**	**Traversal time (%)**	**Distance (m)**	**Distance (%)**
Theoretical best (map explored)	113.93	93.59%	5.83	94.73%
Theoretical fastest traversal for the Theta* wall-following path	121.74	100.00%	6.15	100.00%
Theta* wall-following	250.70	205.94%	6.15	100.00%

For Destination 4, the difference in values (see Table 14) was even higher, just like for the more complex Destination 2 in simulation. In addition, Destination 4 was the first example where the theoretical best (for when the map is explored) was not chosen as the path, as the shortest path would have been to go from the right of the structure. The traversal time for Destination 4 was nearly four times higher than the theoretical fastest traversal time. This shows that even if the path itself could be considered relatively optimal for an unknown map, then there are improvements that can be made for a quicker traversal of the path.

So far mostly successful traversals have been shown. However, a more complex destination — Destination 5 — got stuck in a local loop (see Figure 47). As regular wall-following did not encounter issues in the same place, then it can be deduced that the current system of utilising wall-following is still susceptible to local loops of a specific size. One aspect that likely contributed to the issue was that the Theta* path planning algorithm could not find a

Table 14. *Performance of Theta* wall-following algorithm for destination 4 compared to the theoretical best cases.*

Description	**Traversal time (s)**	**Traversal time (%)**	**Distance (m)**	**Distance (%)**
Theoretical best (map explored)	116.35	55.20%	5.87	58.37%
Theoretical fastest traversal for the Theta* wall-following path	210.76	100.00%	10.06	100.00%
Theta* wall-following	810.40	384.51%	10.06	100.00%

way out, as the square sizes were too large and thus all squares nearby had been marked as occupied. In these rare cases, the algorithm uses wall-following until it finds a valid path. Thus it was thought that this behaviour might need additional logic to not get stuck in a loop.

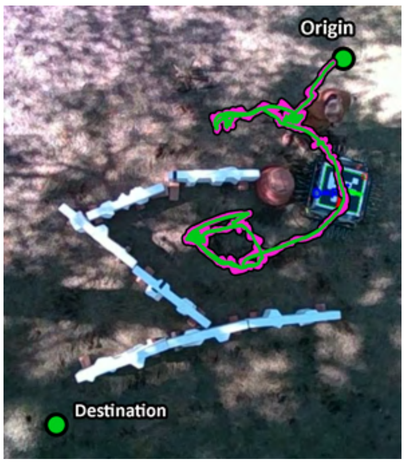


Figure 47. *Failed real-world traversal with Theta* due to local loop in utilising wall-following.*

Overall, though there are still a few cases (one that we know of) which require more thorough testing, both algorithms proved to be reliable for reaching the destinations in an unknown map, both in simulation and in real-world experiments. In order to reach more optimal results in terms of traversal time, the locomotion system needs to be improved. In addition, there definitely exist optimisations to enable quicker navigation in sharp corners.

4.5 Potential avenues for future research

As this is relatively novel research into the possibility of using tactile sensors for navigation outdoors, there are a plethora of avenues for future research that the thesis opens. These can be split into five categories: unifying systems, improved simulation, locomotion system improvements, whisker configuration improvements and navigation algorithm improvements.

For unifying systems, a way of not needing the external camera-based pose estimation is necessary to be able to have the navigation algorithm in actual use. This could be added for example by unifying it with a SLAM algorithm, as was stated before. This could also be provided by exploring the mine with several robots at once and having them communicate for providing pose, as was done in [21]. For unifying systems, another potential avenue would be the development of a control application with more features and a proper graphical user interface. For example, this could display the discovered map and also the location of all robots. In addition, of course in the long term an avenue related to the robot would be to perform mining with it.

A major improvement would also be to have improved simulation for the robot. One part of this would be to properly simulate the flexible whiskers and the perpendicular force z. This would allow for improved tuning of the robot. In addition, the simulation of the terrain and the locomotion system could also be improved to be more realistic. These in turn would significantly speed up the development of all other avenues of research for the robot.

Regarding the locomotion systems, it was observed that the locomotion system produces inoptimal motor speeds for some movements, especially for turns with a wide radius. If a more optimal locomotion system could be created, then this would speed up the navigation of the robot significantly.

As seen in Section 4.2, several changes were made to the whisker configuration and the whiskers themselves to enable real-world use. In addition, many avenues of further research were proposed within the chapter as well. One major avenue of research is the optimal material and shape of the whiskers. For example one major improvement would be to get rid of the grooves in the whiskers to reduce friction. Additionally, other whisker configurations are worth exploring, both for horizontal and vertical axes. One proposition of this was to have a circular configuration for more efficient turning.

For the navigation algorithm itself, there are also a number of additional research avenues. The most pertinent is the overall speed optimisation for traversing sharp angles. In addition, a more thorough analysis could be performed of the optimality of which side to follow when encountering a wall in unmapped positions. Another development aspect would be also to allow any side of the robot to follow a wall, not just the left or right as given now — and this could also be dynamic, i.e. follow the wall on the side that is most optimal. This optimality could also take into consideration which way it would be more efficient to traverse the surface, i.e. straight or sideways.

One more avenue for research in regards to the navigation algorithm would also be to enable it to perform navigation in 3D, in multilevel mines and caves. In addition, a worthy avenue of exploration would be to test it in a large mine, or to have a larger sample size of tests for determining any other shortcomings of the algorithm. For maps, an occupancy map could also be used instead of a discrete grid, or other ways of mapping, in order to provide increased precision when path planning. And finally, of course the optimisation of the computation speed of the algorithm is also worth exploring in order to run it in big mines.

5. Summary

The main aim of the thesis was to validate the ability to perform navigation using the novel sensing method of whiskers. For doing this, a control application was created, along with three algorithms: a wall-follower, a Theta* navigation algorithm and an algorithm combining the two for increased navigating efficiency in unknown maps. For both the regular wall-follower and the algorithm that combined wall-following with navigation, the act of following a wall took between 139.22% and 245.45% of the theoretical fastest traversal, thus not losing an inordinate amount of speed due to wall-following.

For navigation, the hypothesized greater efficiency of utilising wall-following was shown to be true in simulations, improving the traversal time by between $26 - 43\%$. In addition, relatively efficient paths were found by the algorithm considering the missing knowledge about the environment for the algorithm. Though the paths were relatively optimal, then the speed for traversing them could be improved significantly. The reasons for these are mostly inoptimalities in the locomotion system and the turning in sharp angles being slow due to the constraint of constantly being in contact with walls on the robot's sides.

Overall, it was validated that the ability to perform navigation using whiskers certainly exists. Due to the state-of-the-art techniques being difficult to use in mine and cave environments, then these results are not meant to be compared with existing methods using LiDAR and cameras. This thesis displays that tactile sensing can be used in complex environments to perform wall-following. In the event that a relatively accurate method of estimating pose is unified with the algorithm, then tactile sensing can also be used as an efficient aid to perform navigation.

The work done in the scope of this thesis opens up a plethora of potential avenues for further research. This includes improving the locomotion system, whisker configurations, the navigation algorithm itself and also developing more realistic simulation environments. An additional avenue of research is unifying the robot with other robots and interfaces for use in mine and cave environments, for industrial and research applications.

References

[1] Yusra Alkendi, Lakmal Seneviratne, and Yahya Zweiri. “State of the Art in Vision-Based Localization Techniques for Autonomous Navigation Systems”. In: *IEEE Access* PP (May 2021), pp. 1–1. DOI: 10.1109/ACCESS.2021.3082778.

[2] Luís Lopes et al. “ROBOMINERS – Developing a bio-inspired modular robot-miner for difficult to access mineral deposits”. In: *Advances in Geosciences* 54 (Oct. 2020), pp. 99–108. ISSN: 1680-7359. DOI: 10.5194/adgeo-54-99-2020.

[3] *Resilient Bio-inspired Modular Robotic Miners*. Mar. 2023. URL: https://cordis.europa.eu/project/id/820971/news.

[4] *Robominers - resilient Bio-inspired Modular Robotic Miners*. URL: https://robominers.eu.

[5] Steven Macenski et al. “Robot Operating System 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66 (May 2022). DOI: 10.1126/scirobotics.abm6074. URL: https://doi.org/10.1126%5C%2Fscirobotics.abm6074.

[6] Asko Ristolainen, Maarja Kruusmaa, and Simon Godon. *New Bio-Inspired Locomotion Strategies Concepts For Mining Environments*. Tallinn University of Technology, June 2021, pp. 9–10. DOI: https://doi.org/10.3030/820971.

[7] Andreas Nagel. “Blind Mapping and Localisation for Small-Scale Mining Robots”. MA thesis. Tallinn: Tallinn University of Technology, 2021.

[8] Teresa A. Kent et al. “WhiskSight: A Reconfigurable, Vision-Based, Optical Whisker Sensing Array for Simultaneous Contact, Airflow, and Inertia Stimulus Detection”. In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 3357–3364. ISSN: 23773766. DOI: 10.1109/LRA.2021.3062816.

[9] Asko Ristolainen et al. “Hydromast: A bioinspired flow sensor with accelerometers”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9793. Springer Verlag, 2016, pp. 510–517. ISBN: 9783319424163. DOI: 10.1007/978-3-319-42417-0{_}55.

[10] Martin J Pearson and Mohammed Salman. “Active Whisker Placement and Exploration For Rapid Object Recognition”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 672–677. DOI: 10.1109/IROS40897.2019.8968517.

[11] Chenxi Xiao et al. "Active Multiobject Exploration and Recognition via Tactile Whiskers". In: *IEEE Transactions on Robotics* (Dec. 2022). ISSN: 19410468. DOI: 10.1109/TRO.2022.3182487.

[12] Charles Fox et al. "Tactile SLAM with a biomimetic whiskered robot". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., 2012, pp. 4925–4930. ISBN: 9781467314039. DOI: 10.1109/ICRA.2012.6224813.

[13] Oliver Struckmeier et al. "ViTa-SLAM: A Bio-inspired Visuo-Tactile SLAM for Navigation while Interacting with Aliased Environments". In: *2019 IEEE International Conference on Cyborg and Bionic Systems, CBS 2019* (Sept. 2019), pp. 97–103. DOI: 10.1109/CBS46900.2019.9114526.

[14] A. Karambakhsh et al. "Robot navigation algorithm to wall following using fuzzy Kalman filter". In: *IEEE International Conference on Control and Automation, ICCA*. 2011, pp. 440–443. ISBN: 9781457714757. DOI: 10.1109/ICCA.2011.6138043.

[15] T Yata, L Kleeman, and S Yuta. "Wall following using angle information measured by a single ultrasonic transducer". In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. Vol. 2. 1998, pp. 1590–1596. DOI: 10.1109/ROBOT.1998.677372.

[16] Tey Wee Teng et al. "Vision based wall following framework: A case study with HSR robot for cleaning application". In: *Sensors (Switzerland)* 20.11 (June 2020), pp. 1–24. ISSN: 14248220. DOI: 10.3390/s20113298.

[17] Manuel Mucientes et al. "Evolutionary learning of a fuzzy controller for wall-following behavior in mobile robotics". In: *Soft Computing* 10.10 (Aug. 2006), pp. 881–889. ISSN: 14327643. DOI: 10.1007/s00500-005-0014-x.

[18] Xin Wei et al. "A wall-following algorithm based on dynamic virtual walls for mobile robots navigation". In: *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*. 2017, pp. 46–51. DOI: 10.1109/RCAR.2017.8311834.

[19] Jusuk Lee et al. "Templates and anchors for antenna-based wall following in cockroaches and robots". In: *IEEE Transactions on Robotics* 24.1 (Feb. 2008), pp. 130–143. ISSN: 15523098. DOI: 10.1109/TRO.2007.913981.

[20] Xianchao Long. "Tactile-Based Mobile Robot Navigation". MA thesis. WORCESTER: WORCESTER POLYTECHNIC INSTITUTE, May 2013. URL: https://web.wpi.edu/Pubs/ETD/Available/etd-061313-134710/unrestricted/xlong.pdf.

[21] Himangshu Kalita et al. “Path planning and navigation inside off-world lava tubes and caves”. In: *2018 IEEE/ION Position, Location and Navigation Symposium, PLANS 2018 - Proceedings*. Institute of Electrical and Electronics Engineers Inc., June 2018, pp. 1311–1318. ISBN: 9781538616475. DOI: 10.1109/PLANS.2018.8373521.

[22] Mihir Dharmadhikari et al. “Autonomous Cave Exploration using Aerial Robots”. In: *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2021, pp. 942–949. DOI: 10.1109/ICUAS51884.2021.9476870.

[23] Wennie Tabib et al. “Autonomous Cave Surveying With an Aerial Robot; Autonomous Cave Surveying With an Aerial Robot”. In: *IEEE Transactions on Robotics* 38.2 (2022). DOI: 10.1109/TRO.2021. URL: https://doi.org/10.1109/TRO.2021..

[24] Héctor Azpúrua et al. “Towards Semi-autonomous Robotic Inspection and Mapping in Confined Spaces with the EspeleoRobô”. In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 101.4 (Apr. 2021). ISSN: 15730409. DOI: 10.1007/s10846-021-01321-5.

[25] Jesus Aleman et al. “Autonomous navigation for a holonomic drive robot in an unknown environment using a depth camera”. In: SPIE-Intl Soc Optical Eng, Aug. 2020, p. 1. DOI: 10.1117/12.2568163.

[26] Christian Zammit and Erik Jan van Kampen. “Comparison between A* and RRT algorithms for UAV path planning”. In: *AIAA Guidance, Navigation, and Control Conference, 2018*. American Institute of Aeronautics and Astronautics Inc, AIAA, Jan. 2018. ISBN: 9781624105265. DOI: 10.2514/6.2018-1846.

[27] João Braun et al. “A comparison of A* and RRT* algorithms with dynamic and real time constraint scenarios for mobile robots”. In: *SIMULTECH 2019 - Proceedings of the 9th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. SciTePress, 2019, pp. 398–405. ISBN: 9789897583810. DOI: 10.5220/0008118803980405.

[28] Song Luu. *Comparing the motion planning methods Hybrid A* and RRT for autonomous off-road driving of bicycle vehicles*. Tech. rep. 2021.

[29] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[30] W. Zeng and R. L. Church. "Finding Shortest Paths on Real Road Networks: The Case for A*". In: *Int. J. Geogr. Inf. Sci.* 23.4 (Apr. 2009), pp. 531–543. ISSN: 1365-8816. DOI: 10.1080/13658810801949850. URL: https://doi.org/10.1080/13658810801949850.

[31] Kenny Daniel et al. "Theta*: Any-Angle Path Planning on Grids". In: *J. Artif. Intell. Res. (JAIR)* 39 (Jan. 2014). DOI: 10.1613/jair.2994.

[32] Alex Nash and Sven Koenig. "Any-Angle Path Planning". In: *AI Magazine* 34.4 (Sept. 2013), pp. 85–107. DOI: 10.1609/aimag.v34i4.2512. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2512.

[33] Karl J. Åström and Tore Hägglund. *Advanced PID Control*. International Society of Automation (ISA), 2006. URL: https://app.knovel.com/hotlink/toc/id:kpAPIDC001/advanced-pid-control/advanced-pid-control.

[34] Olof Garpinger and Tore Hägglund. "Modeling for optimal PID design". In: *IFAC Proceedings Volumes (IFAC-PapersOnline)*. Vol. 19. Apr. 2014.

[35] *Driver PID Settings*. URL: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=9013.

[36] Tomohiro Takagi and Michio Sugeno. "Fuzzy identification of systems and its applications to modeling and control". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-15.1 (1985), pp. 116–132. DOI: 10.1109/TSMC.1985.6313399.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I, Tanel Kossas:

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Tactile Navigation Algorithm For Autonomous Underground Mining Robots", supervised by Mohamed Walid Remmas, Roza Gkliva and Asko Ristolainen
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

09.05.2023

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.