

EPR400/2 Project Report

Final Report

M.B. Hanekom

1807432

Submitted as partial fulfilment of the requirements of Project EPR400
in the Department of Electrical, Electronic and Computer Engineering
University of Pretoria

November 2021

Study leader: Prof. I.K. Craig

Part 1. Preamble

This report describes work that I have done in my final year project, developing a robotic vacuum cleaner to vacuum a room autonomously.

Project proposal and technical documentation

This main report contains an unaltered copy of the approved Project Proposal (as Part 2 of the report).

Technical documentation appears in Part 4 (Appendix).

All the code that I developed appears as a separate submission on the AMS.

Project history

This project makes use of the kinetic model and obstacle avoidance algorithm by Yiping et al. [1], the mapping algorithm by Wickramarachchi et al. [2] and the navigation algorithms by Hasan et al. [3]. These algorithms have been modified extensively and adapted to the requirements of this project. Where other authors' work has been used, it has been cited appropriately, and the rest of the work reported on here, is entirely my own.

Language editing

This document has been language edited by a knowledgeable person. By submitting this document in its present form, I declare that this is the written material that I wish to be examined on.

My language editor was _____.

Language editor signature

Date

Declaration

I, _____ understand what plagiarism is and have carefully studied the plagiarism policy of the University. I hereby declare that all the work described in this report is my own, except where explicitly indicated otherwise. Although I may have discussed the design and investigation with my study leader, fellow students or consulted various books, articles or the Internet, the design/investigative work is my own. I have mastered the design and I have made all the required calculations in my lab book (and/or they are reflected in this report) to authenticate this. I am not presenting a complete solution of someone else.

Wherever I have used information from other sources, I have given credit by proper and complete referencing of the source material so that it can be clearly discerned what is my own work and what was quoted from other sources. I acknowledge that failure to comply with the instructions regarding referencing will be regarded as plagiarism. If there is any doubt about the authenticity of my work, I am willing to attend an oral ancillary examination/evaluation

about the work.

I certify that the Project Proposal appearing as the Introduction section of the report is a verbatim copy of the approved Project Proposal.

M.B. Hanekom

Date

TABLE OF CONTENTS

Part 1. Preamble	i
Part 2. Project definition: approved Project Proposal	vi
Part 3. Main Report	xv
1 Literature study	1
1.1 Positioning and Mapping	1
1.2 Navigation	2
1.3 Sensors	4
2 Approach	6
3 Design and implementation	8
3.1 Design summary	8
3.2 Theoretical Analysis & Design	10
3.3 Hardware Design & Implementation	33
3.4 Software Design & Implementation	37
3.5 Simulations	38
3.6 Visual and statistical analysis	40
4 Results	41
4.1 Summary of results achieved	41
4.2 Qualification tests	42
5 Discussion	43
5.1 Interpretation of results	43
5.2 Critical evaluation of the design	43
5.3 Design ergonomics	43

5.4	Health, safety and environmental impact	43
5.5	Social and legal impact of the design	43
6	Conclusion	44
6.1	Summary of the work completed	44
6.2	Summary of the observations and findings	44
6.3	Contribution	44
6.4	Future work	44
7	References	45
	 Part 4. Appendix: technical documentation	 47
	HARDWARE part of the project	48
	Record 1. System block diagram	48
	Record 2. Systems level description of the design	48
	Record 3. Complete circuit diagrams and description	48
	Record 4. Hardware acceptance test procedure	48
	Record 5. User guide	48
	 SOFTWARE part of the project	 48
	Record 6. Software process flow diagrams	48
	Record 7. Explanation of software modules	48
	Record 8. Complete source code	48
	Record 9. Software acceptance test procedure	48
	Record 10. Software user guide	48
	 EXPERIMENTAL DATA	 48
	Record 11. Experimental data	48

LIST OF ABBREVIATIONS


UGV	Unmanned ground vehicle
SLAM	Simultaneous localization and mapping
RRT	Rapidly-expanding random tree
SRT	Sensor-based random trees
GPS	Global positioning system
IMU	Inertial measurement unit
MARG	Magnetic angular rate gravity
AWGN	Additive white Gaussian noise
EKF	Extended Kalman filter
UKF	Unscented Kalman filter
FOV	Field of view
AI	Artificial Intelligence

Part 2. Project definition: approved Project Proposal

This section contains the problem identification in the form of the complete approved Project Proposal, unaltered from the final approved version that appears on the AMS.

For use by the Project lecturer	Approved	Revision required
---------------------------------	----------	-------------------


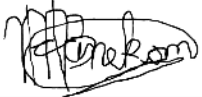
Feedback


Approved

Good project and thorough document, but scoped too large for Project.
 You don't have any requirements for the robot itself.
 You may scale requirements 4 to 7 down to minimal requirements and ensure that the project focus is with requirements 1 to 3.
 Approval is conditional on this: that you understand that the core focus is on Requirements 1 to 3, and the design and building of the platform.
 A revision is not required, but if preferred you may submit one in which these requirements are scaled appropriately.

Symbol awarded: A

To be completed by the student							
PROJECT PROPOSAL 2021				Project no	IKC7	Revision no	0
Title	Surname	Initials	Student no	Study leader (title, initials, surname)			
Mr	Hanekom	MB	18007432	Prof IK Craig			
Project title							
Robotic vacuum cleaner							

Language editor name	Language editor signature
MB Hanekom	
<u>Student declaration</u> I understand what plagiarism is and that I have to complete my project on my own.	<u>Study leader declaration</u> This is a clear and unambiguous description of what is required in this project
Student signature	Study leader signature and date
	21 May 2021 

1. Project description

What is your project about? What does your system have to do? What is the problem to be solved?

Domestic chores such as cleaning the floors takes considerable time and energy. People have to sacrifice work, leisure and family time to keep their houses clean. This project will aim to lessen the domestic burden by designing a autonomous vacuum cleaner, that can clean the whole or specified areas of the house with as little human intervention as possible.

The robot will map out a specified floor area, estimate the time required to clean the area and commence cleaning, which involves vacuuming dirt everywhere in the room. The thoroughness of the cleaning operation will depend on the amount of dirt present. The autonomous robot will also be able to manage its own power and return to the source when it needs to recharge. The robot also needs to be smart enough to avoid falling off stairs or leaving the specified room via an open doorway.

2. Technical challenges in this project

Describe the technical challenges that are *beyond* those encountered up to the end of third year and in other final year modules.

2.1 Primary *design* challenges

- An efficient mapping algorithm needs to be designed that is capable of mapping a room of any shape to a certain size accurately.
- A thorough navigation algorithm and driving control system needs to be implemented which will guide the robot everywhere safely.
- A dirt sensing evaluation algorithm needs to be developed to enable dirt dependent cleaning.
- An obstacle avoidance algorithm needs to be designed that successfully avoids colliding with static objects.

2.2 Primary *implementation* challenges

- The docking station and way to dock with the power source needs to be built that allows the robot to easily and consistently connect to its own power.
- A powerful vacuum pump and brush technique needs to be constructed to suck large and small dirt particles.
- A dirt sensor, which will be mixture of off the shelf and custom sensing components needs to be correctly built and positioned for optimal efficiency.
- The optimal positioning and amount of sensors needs to be chosen to see the environment clearly and accurately.

3. Functional analysis

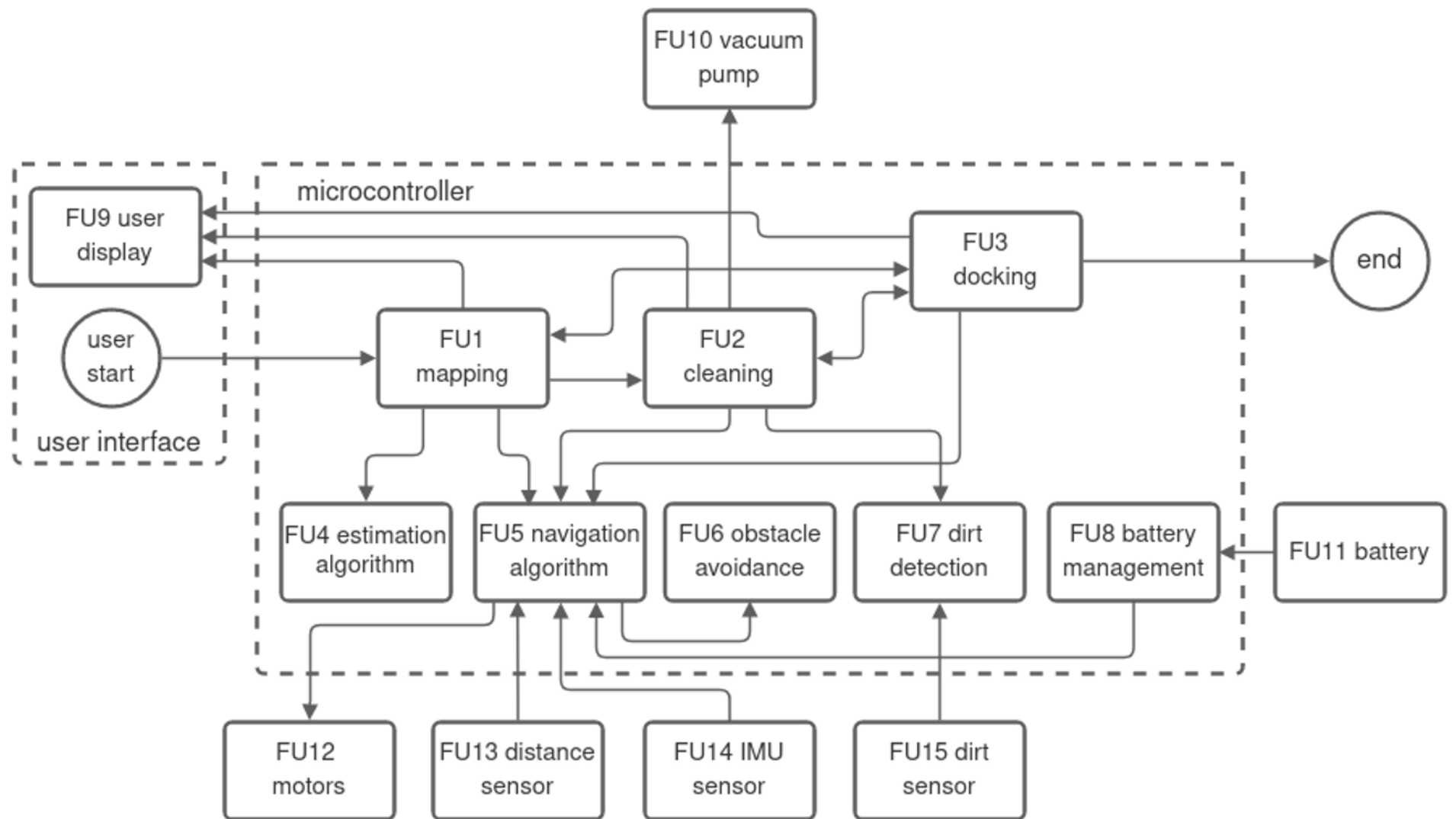
3.1 Functional description

Describe the design in terms of system functions as shown on the functional block diagram in section 3.2. This description should be in narrative format.

The robot starts by first mapping the available floor space (FU1). This state uses the navigation algorithm (FU5) to move around and when it finishes uses the measured floor space to estimate the time required to clean the area (FU4) and displays both the measured space and the estimated time on the display (FU9). The robot then proceeds to cleaning (FU2), which involves driving the vacuum pump (FU10), navigating the area (FU5) and detecting the dirt (FU7), as well as displaying its progress on the display (FU9). After cleaning is complete, or during mapping or cleaning if the battery level is too low, the robot will initiate the docking sequence (FU3) after navigating to the power source (FU5). If the robot is finished, it will remain docked at power, otherwise it will continue the current task when its battery is recharged. The battery level and percentage cleaned is also displayed on the user display (FU9).

The navigation algorithm (FU5) takes as input the current distance to the nearest object from the distance sensors (FU13), the acceleration from the IMU sensor (FU14) and the battery level from the battery management unit (FU8), which it measures from the battery power level (FU11). The algorithm then guides the robot through the room by driving the motors (FU12) and calling the obstacle avoidance algorithm (FU6) to avoid colliding with objects. The dirt detection algorithm (FU7) takes as input the dirt level from the dirt sensor (FU15) to determine how thorough the robot should clean a given area.

3.2 Functional block diagram



4. System requirements and specifications

These are the core requirements of the system or product (the mission-critical requirements) in table format IN ORDER OF IMPORTANCE. Requirement 1 is the most fundamental requirement.

	Requirement 1: the fundamental functional and performance requirement	Requirement 2	Requirement 3
1. Core mission requirements of the system or product. Focus on requirements that are core to solving the engineering problem. These will reflect the solution to the problem.	Autonomous navigation in an area: the robot should be able to navigate a known area thoroughly and completely.	Autonomous 2D mapping of an area: the robot should be able to map an unknown area accurately.	Obstacle avoidance: the robot should not bump into objects, but rather navigate around them intelligently.
2. What is the target specification (in measurable terms) to be met in order to achieve this requirement?	The robot should cover at least 95% of the mapped floor space while cleaning.	The robot should map the specified area with a 10cm ² tolerance in two dimensions. It should map an area smaller than 20m ² completely to within 10cm ² tolerance.	The robot should not collide with static objects, stopping at least 1cm before the object and going around it.
3. Motivation: how or why will meeting the specification given in point 2 above solve the problem? (Motivate the specific target specification selected)	The robot can only clean where it walks, thus in order to clean properly everywhere, it needs to go everywhere. <div>why 95%?</div>	Mapping an area before cleaning will allow the robot to know the bounds of the specified room, calculate the best cleaning route and give an estimate of its workload, which will enable it to clean the floor more efficiently. <div>where do the numbers come from?</div>	The robot can injure itself or others if charging into objects. It can also get stuck in loose clothes, which would render it unable to clean the floor. <div>Where does 1 cm come from?</div>
4. How will you demonstrate at the examination that this requirement (point 1 above) and specification (point 2 above) has been met?	100 rice grains will be scattered evenly over a known floor area and allow the robot to clean. It should clean at least 95 of the grains, thus there should be 5 or less remaining grains when it is finished.	Place the robot in an unknown space no greater than 20m ² . The robot should map the area and give its size, which should be accurate to 10cm ² .	Place one or more static objects in the room, the robot should navigate around it without bumping into any of them.
5. Your own design contribution: what are the aspects that you will design and implement yourself to meet the requirement in point 2? If none, remove this requirement.	navigation algorithm, sensor data processing, spatial sensing and motor control system.	mapping algorithm, sensor data processing, user interface, spatial sensing and motor control system.	sensor data processing, object collision algorithm, spatial sensing and motor control system
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate "none"	microcontroller, distance sensor, DC motors, wheels.	microcontroller, output display, distance sensor, DC motors, wheels.	microcontroller, distance sensor, DC motors, wheels.

these are good for a commercial product, but may be scaled down for Project

The core focus will be requirements 1 to 3

System requirements and specifications page 2

	Requirement 4	Requirement 5	Requirement 6
1. Core mission requirements of the system or product. Focus on requirements that are core to solving the engineering problem. These will reflect the solution to the problem.	Dirt detection and removal: the robot should clean more or less thoroughly depending on the amount of dirt present at the current location.	Autonomous power management: the robot should return to docking station when necessary and never run out of power.	User feedback: the robot should estimate its cleaning time accurately.
2. What is the target specification (in measurable terms) to be met in order to achieve this requirement?	The robot should be able to clean three different levels of dirty, which is measured in the amount of rice grains spread over 10cm ² . The robot should clean at least 90% of the rice without wasting time (continue cleaning when it is already clean).	The robot be should aware of its power levels and return to the station when battery falls below 20%. It should dock itself to the charge port before battery falls below 5%.	After mapping the room the robot should estimate the cleaning time accurately to within 5 minutes, taking in consideration the size of the room, dirt level at samples and obstacles present. It should also display what it is busy doing (cleaning, mapping, returning to dock) visibly for the user.
3. Motivation: how or why will meeting the specification given in point 2 above solve the problem? (Motivate the specific target specification selected)	The robot needs to clean the floor thoroughly and efficiently. It should not waste energy by overcleaning a relatively spotless floor and it should also not miss dirt in a particularly dirty environment. An adaptive system would meet the goal the best.	If the robot fails to charge itself before its power runs out it will need human assistance, which prohibits the robot from being a fully autonomous device.	Accurate feedback and estimation on its tasks allows the user to do other things while the robot is cleaning and add cleaning to the weekly planner.
4. How will you demonstrate at the examination that this requirement (point 1 above) and specification (point 2 above) has been met?	Demarcate three 10cm ² areas. Add 1 teaspoon of rice to Area 1, 1 tablespoon to Area 2 and 3 tablespoons to Area 3. The robot should spend more time in each successive area. Afterwards collect remaining grains, which should be less than 1 full teaspoon.	The robot should start cleaning with 30% battery capacity. At 20% power it should return to the power source and dock before it has 5% power remaining.	The robot will map an unknown space no greater than 20m ² , and then give an estimation of the cleaning time. A timer will run during the subsequent cleaning operation. Afterwards the difference should be no more than 5 minutes.
5. Your own design contribution: what are the aspects that you will design and implement yourself to meet the requirement in point 2? If none, remove this requirement.	dirt detection algorithm, navigation algorithm, sensor data processing, motor control system.	power management algorithm, navigation algorithm, battery input data processing, power source docking station .	mapping algorithm, navigation algorithm, estimation algorithm, user interface, spatial sensing.
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate "none"	microcontroller, dirt detection sensor hardware, DC motors, wheels.	microcontroller, battery, output display, DC motors, wheels, power source (wall plug).	microcontroller, output display, timer, LEDs.

System requirements and specifications page 3

same comments as for
Req 4 to 6

	Requirement 7	Requirement 8	Requirement 9
1. Core mission requirements of the system or product. Focus on requirements that are core to solving the engineering problem. These will reflect the solution to the problem.	Mission focus: the robot should not leave the current room during cleaning nor fall down stairs.		
2. What is the target specification (in measurable terms) to be met in order to achieve this requirement?	The robot should stop at least 1cm before stairs or similar altitude drops, during a cleaning operation it should not exit the room via a doorway.		
3. Motivation: how or why will meeting the specification given in point 2 above solve the problem? (Motivate the specific target specification selected)	If the robot falls from a height it might get damaged or break apart, and not be able to continue cleaning. If it leaves the assigned room it might continue cleaning, but not where the user wants it to, thus not fulfilling the mission.		
4. How will you demonstrate at the examination that this requirement (point 1 above) and specification (point 2 above) has been met?	Put the robot on a table and watch that it does not come closer than 1cm to the edge with its wheels when cleaning. Clean in a room with a door that the robot should not go into.		
5. Your own design contribution: what are the aspects that you will design and implement yourself to meet the requirement in point 2? If none, remove this requirement.	mapping algorithm, navigation algorithm, sensor data processing, spatial sensing and motor control system.		
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate "none"	microcontroller, distance sensor, DC motors, wheels.		

System requirements and specifications page 4

	Requirement 10	Requirement 11	Requirement 12
1. Core mission requirements of the system or product. Focus on requirements that are core to solving the engineering problem. These will reflect the solution to the problem.			
2. What is the target specification (in <i>measurable</i> terms) to be met in order to achieve this requirement?			
3. Motivation: <i>how or why</i> will meeting the specification given in point 2 above <i>solve the problem</i> ? (Motivate the <i>specific</i> target specification selected)			
4. How will you demonstrate at the examination that this requirement (point 1 above) and specification (point 2 above) has been met?			
5. Your own design contribution: what are the aspects that <i>you will design and implement yourself</i> to meet the requirement in point 2? If none, <i>remove this requirement</i> .			
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate "none"			

5. Field conditions

These are the REAL WORLD CONDITIONS under which your project has to work and has to be demonstrated.

	Field condition 1	Field condition 2	Field condition 3
Field condition requirement. In which field conditions does the system have to operate? Indicate the one, two or three most important field conditions.	The robot should operate indoors on a relatively flat surface.	The robot should be able to operate in rooms with static obstacles.	
Field condition specification. What is the specification (in measurable terms) for this field condition?	The floor should not have sharp steps or troughs having a sudden incline of more than 8 mm - this also includes the incline to get onto a carpet.	There may be a mixture of static (motionless) objects, such as chairs, boxes and tables, but no more than 10 per room.	

6. Student tasks

6.1 Design and implementation tasks

List your primary design and implementation tasks in bullet list format (5-10 bullets). These are *not* product requirements, but *your* tasks.

- Design the hardware layout of the robot.
- Calculate the optimal amount and position of the distance and dirt sensors.
- Acquire hardware and build prototype.
- Build a docking station for the robot to draw power from.
- Design a navigation algorithm and iteratively improve it by testing on the prototype.
- Design a mapping algorithm and iteratively improve it by testing on the prototype.
- Design a docking algorithm and iteratively improve it by testing on the prototype.
- Add obstacle avoidance functionality to the robot.
- Add dirt dependent cleaning functionality to the robot.

You may not write in the imperative form

6.2 New knowledge to be acquired

Describe what the theoretical foundation to the project is, and which new knowledge you will acquire (beyond that covered in any other undergraduate modules).

- The student will need to learn about Lidar distance sensors and implementation.
- The student will need to learn how 2D space mapping and modelling works.
- The student will need to learn about image filtering used in the dirt detection techniques.
- The student will need to understand the control systems involved in navigating a robot with wheels.
- The student will need to learn about battery management and charging.

Part 3. Main Report

1. Literature study

The world is a vast and mysterious place, full obstacles, dynamic elements and unforeseen circumstances. Robots that have to navigate the world autonomously face a difficult and complex challenge, and require sophisticated solutions to address the uncertainty in sensor data, the danger of the environment and the comprehensive nature of the problem. Autonomous mapping and navigation of a UGV is the cornerstone of mobile robotics. Without a system capable of mapping the world and its location in it, as well as planning paths effectively, a robot relies on human interference to move around.

1.1 Positioning and Mapping

A UGV has to create and maintain an internal representation of the world to navigate around and operate intelligently. The model needs to be at the right level of granularity to interpret the world accurately and efficiently. It consists of a map of the environment as well as an estimation of the robot's position and orientation in it, updated by a process called SLAM.

Various SLAM algorithms exist, among which Particle Filter (PF)-SLAM and Graph-SLAM are most prominent. Sugiura and Matsutani [4] investigate the performance of both methods during mapping. PF-SLAM estimates the map and robot position by distributing samples throughout the search space, which are weighted according to the degree of similarity between the sample's estimate of the environment and the sensor's measurement. Larger weights will propagate to the next sample, while smaller ones are removed. Graph-SLAM builds a node-graph while traversing the area, which optimizes the map around closed loops created upon arrival in a previously visited place.

An important aspect of both methods is the concept of scan-matching, where local maps built at different timesteps are compared to see if the map and position of the robot can be optimized in such a way that the maps correlate better. Sugiura and Matsutani interpret the map as a grid and evaluate four algorithms: Hill-Climbing, Gauss-Newton, CSM and Branch-and-Bound for optimization. The former two approaches are subject to local maxima, whereas the latter two are computationally expensive. The researchers finally implement CSM, starting with a coarse search map, which iteratively increases in granularity and decreases in size as the proximity of the optimal location is identified.

Zooming in on the mapping itself, there are many ways to model the map information obtained from the various sensors measurements, but it is important to choose a model that accurately describes the environment and facilitates quick and easy positioning and navigation. One implementation involves creating a quadtree, as proposed by Han et al. [5], which partitions the known environment into occupied and unoccupied blocks hierarchically, stopping at the coarsest resolution with a uniform occupancy state. This effectively reduces the memory required to store the map and can ease path planning calculations in some applications.

An alternative approach involves creating an occupancy map, either bit-wise, where "0" represents open and "1" represents blocked, or probabilistic, where each cell stores the probability that its area is occupied. Chen et al. [6] describes a probabilistic occupancy grid map that undergoes optimization via a Bayes Filter.

1.2 Navigation

Autonomous navigation is a broad field, and many ingenious algorithms have been developed to address the challenge. The algorithms can be classified into two groups according to their goal, in other words the objective that the method is trying to achieve. The first group aims to determine an optimal path to a certain position in a known environment. In essence, these methods try to plan a path to a certain position in the least amount of time and distance covered, without walking into obstacles. The second group of algorithms aim to maximize information gain in an unknown environment, exploring the landscape as efficiently as possible.

A-star search, optimized for navigation in the paper by Ju et al. [7], is the most famous of the first group of path planning algorithms. A-star is a search algorithm with a heuristic, typically provided as the distance to the goal, and a cost, usually the distance travelled to the position, which tries to minimize the sum of the heuristic and the cost at each iteration. The algorithm always chooses the path with the lowest combined heuristic and cost and is thus optimal, even though complete knowledge of the relevant surroundings is needed. The paper adds an improvement to elegantly avoid obstacles similarly to how a human agent would, by heading towards the goal in a straight line while swerving around small obstacles. There is also potential to combine small sequential obstacles to form a larger obstacle that can be navigated more effectively.

A path planning algorithm that works on an entirely different principle is RRT. RRT and its improvement RRT*, is described by Chen et al. in [8] as a sampling-based alternative to the slower and more complex algebraic path planning methods. The algorithm randomly chooses the next location in the state space and proceeds in the location's direction by a set or varying amount, exploring new territory and evaluating if the goal has been reached. These class of algorithms are very simple to implement, and can be used in a wide variety of problems, due to the flexibility and generality of the approach. The methods also biases exploration into unknown domains, which helps to rapidly converge on a solution, if one exists. Whereas RRT connect the newly explored node to the geometrically closest node in a tree, the RRT* algorithm seeks to optimize the solution by joining nodes that correspond to the closest distance to that point from the origin. This will produce an optimal result as the number of nodes go to infinity, but suffers from increased complexity and computational burden.

The implementation specifically proposed in Chen's paper describes a novel adaptation to the dual tree RRT (DT-RRT), which constructs two trees, one starting at the state and one starting

at the goal to simultaneously add nodes in the state space, aiming to converge on a solution sooner. Their paper not only combines dual trees with RRT* rewiring, but also samples in a Gaussian sampling cloud, which is dependent on the state of the system and the context of the problem. They also include methods to deal with boundary problems. This algorithm, being extremely sophisticated, will find use in large and complex path planning problems.

The second category of navigation algorithms focuses on exploration and maximizing information gain in the environment. Path planning in an unknown area is inherently more difficult than in known areas, but have seen a lot of research in the past two decades. Groundbreaking work was done by Yamauchi [9] on frontier-based exploration. He proposes an method that seeks to explore an unknown environment by identifying and moving towards areas of highest potential information gain. This is achieved by constructing an occupancy grid map of the environment, specifying open, unknown and obstructed areas. The frontiers between open and unknown spaces are identified and if a frontier has sufficient size, it will be explored by the robot. The algorithm makes room for dynamic obstacle avoidance in the case that the world changed a bit since the last visit, and will explore all reachable areas in time.

Despite the algorithms intuition and completeness appeal, the actual implementation is a lot more daunting. Between the map construction, frontier identification and path planning and obstacle avoidance to potential frontiers, this algorithm has a lot of loosely coupled components, each critical to the success of the operation. Calculating solutions analytically or using search algorithms are exhaustive and potentially burdensome. With this in mind Oriolo et al. [10] divided a new method for exploring frontiers, with inspiration from the RRT approach, called Sensor-based Random Tree (SRT).

SRT uses the sensor measurements to construct a safe zone around the robot, either as a star, assuming accurate sensor readings in the respective directions, or as a ball, assuming inaccurate measurements and regarding only the closest distance to an obstacle. In either case, a random angle is selected together with a distance in that angle's direction near the edge of the safe zone to obtain a new candidate position node. The position is evaluated to verify that it is far enough away from the current position to justify the move and to make sure it does not fall within an already explored safe zone of a previously explored node. If the candidate fails these checks, a new candidate is chosen until the maximum allowed attempts are exceeded, in which case the robot moves back to its parent node. This process ensures that the robot will continuously explore new areas until none are remaining, in which case the robot will return to its original starting position.

Combining the random sampling system first seen in RRT with the bias towards frontiers allows the SRT algorithm to implement a simple and effective sampling based exploration mechanism, capable of navigating an unknown environment intuitively with adjustable parameters depending on the sensor specifications of the robot. This makes SRT a simple and extremely flexible navigation tool with a wide range of possible uses and applications.

1.3 Sensors

To navigate around and interact with the environment intelligently, a UGV has to perceive the world sufficiently. Various types of sensors exist measuring various aspects of the environment at different accuracies, sample rates and costs. In mobile robotics it is often necessary to obtain information about the environment, such as the location of obstacles, as well as information about the robot itself, such as its measured speed or angular rate. Chan et al. [11] recognizes the ubiquity of laser rangefinder in modern UGV systems, which maps the environment as a set of points to the nearest obstacle in a specific direction. The rangefinder is very accurate and its observation data can be used to reconstruct the world map quite effectively.

Another common sensor used for obstacle detection is a monocular camera, first proposed as part of the LSD-SLAM method by Caruso et al. [12]. Photos taken by a camera are extremely accurate and full of detail that can be used to reconstruct the environment precisely. The downside is that photos by themselves do not contain depth-information, so a simple analysis of the data do not yield useful results. Instead, the researchers design an intricate system of depth estimation Kalman filters, map optimization and context tracking across frames to analyse the input data. Not only is the process complex, but the photos themselves are large and obnoxious to work with. A laser rangefinder will output an array of distances to the nearest obstacle in the respective directions, totalling a few dozen data points at most. The photos in the paper, on the other hand, has a resolution of 480x480px, accumulating to 230400 data points arriving at 40 Hz. A system would require large computational and memory resources to deal with such a large influx of data.

Tracking the robots movement in the environment usually comes down to three sensors: IMUs, GPS and odometers. GPS navigation localizes a user on the ground using positioning data from nearby GPS satellites, and is extremely accurate. However these sensors only work well if there is a clear line of sight between the robot and the satellites, and is generally not suitable for use indoors.

IMUs contain a gyroscope, accelerometer in three dimensions, and often comes with a magnetometer as well, then known as a MARG sensor, enabling it to estimation orientation with high accuracy if a suitable filter is used to fuse to sensor data. Harindranath and Arora [13] compare four commonly used filters to measure their statistical error and relative performance. Two statistical filters, the Extended Kalman Filter (EKF) and Two stage EKF (TKF-Q), as well as two non-statistical filters, the Mahony and Madgwick filters are compared. The paper aims to analyze the filters in different operating environments, simulating various levels of Gaussian noise, random walk bias and orientation perturbations. While the TKF-Q filter performed well in high levels of noise, the Mahony filters was the winner all around with the lowest average error for static and dynamic simulations with various levels of noise. This filter also benefits from fast execution times.

Odometers measure the rotation speed of the wheels attached to the motors, and can be used to determine distance travelled and orientation. Unfortunately wheels can slip, leading to inaccurate measurements, thus in general odometer data is fused with either IMU sensor data or corrected with environment perception optimization. Milijas et al. [14] proposes an interactive SLAM-based algorithm, where laser rangefinders are used not only to obtain information of the environment but also to adjust for bad odometry, by viewing both sets of input data in the context of an optimization problem.

2. Approach

Designing and constructing an robotic vacuum cleaner capable of autonomous positioning, mapping, exploration and navigation is a rich and complex problem, with many subsystem that need to work together flawlessly for the end product to function. The first important step upon encountering such a large project is to set up a general roadmap and timeline for the design and implementation of the project. The roadmap starts out by evaluating the system requirements and identifies the main problems that the solution needs to address, which are autonomous navigation and floor cleaning in this case.

These high-level abstract problems are systematically broken down to smaller and more tangible subsystems. Even though the cleaning is a core requirement of the project, it does not entail as much engineering as the navigation, as the project is aimed at computer and not mechanical engineering students, and thus relatively little has to be calculated and designed for the subsystem to function. Autonomous navigation, on the other hand, requires a holistic solution to the problem of robot positioning, mapping and navigation, each with its own sets of challenges and methods, working together in a complex dance. The subsystem's aim is to simultaneously allow the robot to perceive its environment to a sufficient degree that it can interact intelligently with it, and move around in a useful, timely and energy-efficient manner.

Apart from the software design side, the system has a major hardware component as well. There is a need to identify components and design a layout that contains the necessary sensors, processing power and memory to achieve the goal and to integrate the parts efficiently so that it can be used in a mobile application. There is a bit of a design conundrum here, as the precise implementation of the software is dependent on the hardware components and the hardware requirements is dependent on the software perceived problems. This issue is addressed by identifying the high-level needs of the system software and thereafter searching for available and relevant hardware components (quite a challenge with the global supply chains disrupted). After the first few months of shopping and soldering the first iteration of the robot consisted of an chassis, motors and wheels, a H-bridge and a low-cost processor.

Next, the basic startup firmware, including timers and peripheral drivers were implemented. The sensor peripherals used to measure distance, angular rate and wheel odometry are adding one-by-one and tested. The use of a Microchip PIC32 chip prohibited the use of standard functions as every single peripheral is initialized manually with register manipulation. This enables the system to run as efficiently as possible, only using exactly what it needs and how it needs it. Optimization of the available real estate on the robot leads to various iterations of the soldered veroboard in this time, with each iteration becoming smaller and using less pins.

Finally the system is at the point where it can start interacting with the world. Although the software subsystem work together, there is a clear dependency hierarchy, as the mapping depends on accurate real-time positioning and the navigation depends on accurate, real-time mapping. Thus, these subsystems are mostly designed sequentially, iterating back to a previous step once a problem is uncovered. However, before implementation, it was discovered that these systems would be virtually impossible to test objectively using the current evaluation methods. As the reliance on large created data structures and measurement dependent parameters grow, there is no way of knowing why the robot reacts in the way it

does without looking into its brain, which would be difficult with a wired serial connection. Instead an OLED screen and Bluetooth serial connection were added to navigate the thought process of the device.

The positioning subsystem benefits from five sensor measurements: the accelerometer, gyroscope, magnetometer, right and left wheel odometers to update the state of the robot, which includes its position, relative to its internal Cartesian representation of the world. The perceived value and nature of each measurement is evaluated in an elaborate complementary filter. Starting from the systems own estimate of its state given its inputs, which are the PWM duty cycles of the wheels, the estimate is sequentially corrected by the measurements. This approach is favoured due to its simplicity and low computational requirements, whereas the abundance of sensor measurement data aims to bring balance into the equation.

The mapping subsystem faces the challenge of converting the range-based ultrasonic sensor data into an accurate enough representation of the environment so that the robot will not bump into obstacles and clean thoroughly. Three sensors give an idea of the distance to the nearest object in their respective directions, which are offset by 40° around the front of the robot. The robot creates a probabilistic map of a certain cm^2 resolution to represent the world. Considering that the measurement data contains noise and cannot actually determine the position of an obstacle, just the distance to it in a certain wide field of view, the sequential incoming data is plotted on the probabilistic map as a multivariate Gaussian distribution, with the perceived obstacle positions as the means. Applying a highly efficient multiplication algorithm to the evolving map provides a simple and relatively low cost solution of the perception problem. Of course, there is no use in evaluating parts of the global map that is not currently in range, and thus a smaller local map is updated in the immediate area surrounding the robot. To save space, this probabilistic map is regularly saved back into the global map as a 2 bit-map, with the four values representing open (00), unknown (01), cleaned (10) and obstructed (11). This can be effectively utilized by the navigation algorithm to plan its trajectory.

The navigation subsystem has to use the robot's position and map to effectively explore the world, avoid bumping into obstacles and clean an area thoroughly. This is achieved by splitting the robot objectives into local and global groups. Local goals include calculating the immediate next position the robot has to move to in order to reach a global goal or avoid an obstacle. This algorithm uses a depth-first search scanning the surrounding map tiles for an open path, biasing towards front-facing, unexplored and uncleaned areas. Global goals usually entail following a certain pattern, such as an S-like or wall-following path around the room, but also has the capacity to detect areas missed during the preliminary cleaning cycle, which could happen in rooms with peculiar shapes and obstacles. The local navigation focuses on reaching the next few decimeters harmless, while the global planner focuses on the bigger picture, thoroughly sweeping a room. The use of specific Cartesian point destinations, instead of general directions, such as "go forward" or "turn left" for navigation allows for the precise control of the inputs to the motors depending on the error angle and destination to the desired location. This also generalizes the controller to follow its path, find its charging station and dodge obstacles with the same mechanics.

3. Design and implementation

3.1 Design summary

This section summarises the project design tasks and how they were implemented (see table 1).

Deliverable or task	Implementation	Completion of deliverable or task, and section in the report
Determine sensor input requirements	It was determined that the system needs 3 ultrasonic sensors to perceive distance, 2 infrared sensors to detect fall hazards, and a 9-DOF IMU and 2 light odometers to calculate the vehicle's position	Completed. Section 3.3.1
Design the hardware layout on the robot's motherboard	The original PCB design was replaced with a veroboard, which was iteratively optimized through first principles to design and build a small and efficient board	Completed. Section 3.3.2
Design and construct robot body	The robot chassis, motors, wheels, vacuum pump, board and sensors were integrated together in a small autonomous vehicle.	Completed. Section 3.3.3
Design and construct vacuum pump	The vacuum pump itself is an off-the-shelf hand vacuum, but the integration with the system, including current limiting breakout circuit was designed to control its operation during cleaning.	Completed. Section 3.3.5.
Initialize and test peripherals	Peripheral drivers were developed from first principles for the PIC32 microcontroller, for the most efficient implementation and lowest possible latency.	Completed. Section 3.4.3
Design and implement positioning algorithm	A positioning algorithm was implemented from first principles using complementary sensor fusion to position the robot on an internal 2D cartesian map.	Completed. Section ??

Design and implement navigation algorithm	A navigation algorithm was implemented from first principles using ultrasonic proximity detection to reach specific goals and avoid obstacles, as well as control movement intelligently.	Completed. Section 3.2.5
Design and implement mapping algorithm	A mapping algorithm was implemented from first principles to store visited and obstacle information from the environment to plan routes more effectively.	Completed. Section 3.2.2
Implement fall-protection algorithm	A simple maneuver to avoid large was developed with mixed success and will not be demoed extensively for the vehicle's safety.	Incomplete.
Implement dirt-detection algorithm	The proposed dirt detection solution would have been too complex to incorporate into the system design, without adding any real value, and thus a design choice was taken to not implement this aspect of the system.	Incomplete.
Implement autonomous power management algorithm	An power management algorithm was implemented so that the robot returns to its starting position once it detects low power, but the docking station and mechanics of docking precisely was not implemented.	Incomplete.
Integrate in floor-cleaning system	The integration of the positioning, navigation and mapping algorithms cleans a given floor area.	Completed. Section ??
Test and verify results	A range of simulations and experiments were carried out to verify that the robot fulfills the specified requirements, including Python simulations, visual inspection and offline statistical analysis of positioning and navigation data.	Completed. Sections 3.5 & 3.6

Table 1.
Design summary

3.2 Theoretical Analysis & Design

The mathematical structure of the robot's intelligent interaction with the environment can be divided into 5 subsystems: localization, mapping, control, path planning and exploration, working together to sense and perceive the environment, plan a goal and act to attain the goal.

3.2.1 Localization

Filter motivation

It is important for the robot to localize itself accurately in its internal representation of the world, as the success of the mapping and collision-free navigation depends on this. Localization is performing through the non-linear sensor fusion of the IMU's measured acceleration and gravity, as well as the measured wheel velocity from the odometers. A design choice was taken not to implement a SLAM algorithm, as multiple localize-only sensors are available to correct for noise and drift, and the sensors used for mapping, the sonar sensors, are quite inaccurate and do not produce a high quality map of the environment, which would make it difficult to use the mapping data successfully in localization.

Instead a double Unscented Kalman Filter (UKF) is implemented complementing the nature of the sensors and their operation. The IMU can provide a quick succession of angular rate and acceleration values, and due to the transient nature of acceleration, these readings are also most accurate if obtained as continuously as possible. On the other the odometer needs time to measure the rotation of the wheel, and preemptively extrapolating the wheel spin leads to inaccuracies.

Combining the strengths and understanding the shortfalls of the two sensors, we propose a double UKF localization solution. The first filter, known as the "fast" filter, operates at 20 Hz, with its velocity estimate as the process model and incoming IMU data as the measurement model. The second "slow" filter operates at 4 Hz, using the last fast filter estimate as the process model and the incoming odometer readings as the slow filter. The double UKF data flow model is shown in Figure 1.

Every real world system deals with uncertainty in its processes and measurement data, which makes it difficult to accurately estimate a system's state, i.e. the important variables that describe a system in the world. Kalman filters solve the uncertainty problem with statistical probability representations of its state. Assuming that a state's uncertainty is approximately normally distributed, the essence of the state's expected values can be described by only keeping track of its mean and covariance. The Kalman filter updates the mean and covariance of the state over time using a dynamic Bayesian filter. Not only that, but if the process and measurement models are linear, the Kalman filter is the least squares approximator of the system, leading to optimal estimations of the state.

In reality most systems are non-linear, and thus the standard Kalman filter has to be expanded to deal with process or measurement non-linearities. Two common approaches are model linearization, also known as the Extended Kalman filter (EKF), and the unscented transforma-

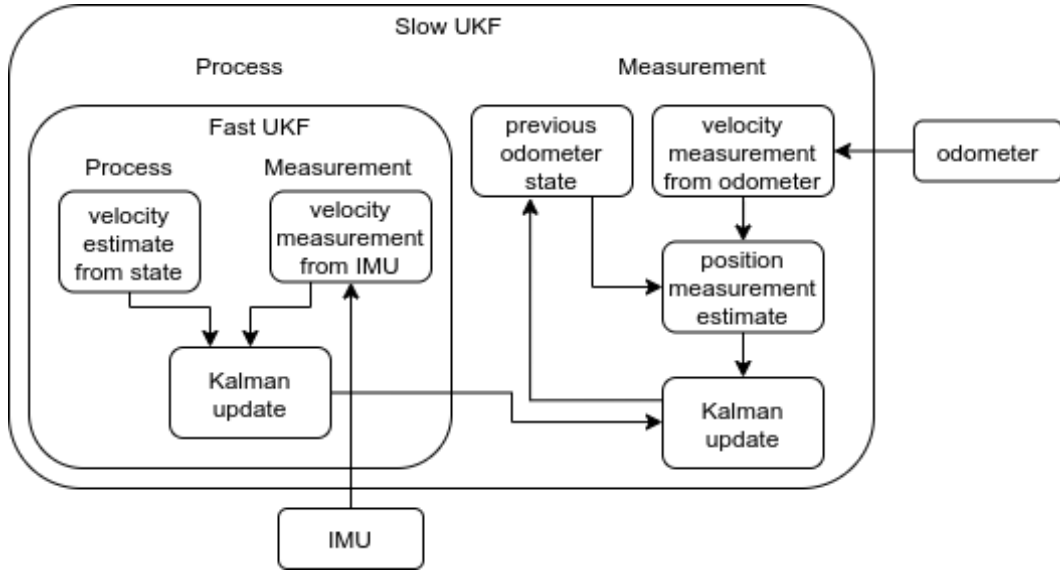


Figure 1.
Double UKF data flow model

tion of specific state estimates, also known as the Unscented Kalman filter (UKF). This paper implements a UKF to solve system dynamics, as it is generally more accurate, but also because the UKF is well suited to the nature of the problem. The localization problem is framed in such a way to require two semi-independent filters, one fast and one slow. Yet the data should still be comparable between the models. By creating the sigma points in the slow filter and updating the systems belief based on those sigma points in the fast filter, the slow filter will have access to both the original data from the previous iteration and the update estimate from the fast filter. This simplifies the design process and enables the UKF to work correctly.

UKF theory

A UKF filters data by creating a set of sigma points from the state, which are processed to obtain an estimate of the current state of the system. This is statistically combined with an estimate from the measurement data to produce a new state estimate. The sigma points are used to reconstruct the distribution of the state after the non-linear transform, and thus each point is offset from the mean by a specific amount. The deviation, also known as weights W are obtained by the Cholesky decomposition of the covariance P with noise Q of the state x .

The Cholesky decomposition is effectively the square root of a matrix, and determines L , where $A = L^T L$, $L = \sqrt{A}$. Each position (i, j) of L can be calculated as

$$L_{ij} = \begin{cases} \sqrt{A_{ij} - \sum_{k=0}^{j-1} L_{jk}^2} & \text{if } i = j \\ \frac{1}{L_{jj}} \sqrt{A_{ij} - \sum_{k=0}^{j-1} L_{jk}^2} & \text{otherwise} \end{cases} \quad (3.1)$$

The set of sigma points $\{X_i\}$ are obtained as

$$W_i = \text{columns}(\sqrt{2n \cdot (P_{k-1} + Q)}) \quad i = 1 \dots n \quad (3.2)$$

$$W_{i+n} = \text{columns}(-\sqrt{2n \cdot (P_{k-1} + Q)}) \quad i = n + 1 \dots 2n \quad (3.3)$$

$$X_i = \hat{x}_{k-1} + W_i \quad i = 1 \dots 2n \quad (3.4)$$

where n is the dimensionality of the state vector. The sigma are transformed through the non-linear process model $A(x, q)$ to obtain the processed points $\{Y_i\}$, as well as the measurement model $H(x, r)$, to obtain the measured points $\{Z_i\}$. The noise variables q and r are set to zero, as noise has already been added into the points as Q .

$$Y_i = A(X_i, 0) \quad (3.5)$$

$$Z_i = H(X_i, 0) \quad (3.6)$$

The mean and covariance of $\{Y_i\}$ and $\{Z_i\}$ are calculated as the weighted sum and correlation of the sets. The measurement noise R is added to the measured covariance P_{zz} .

$$x_k^- = \sum_{i=0}^{2n} W_i^{(m)} Y_i \quad z_k^- = \sum_{i=0}^{2n} W_i^{(m)} Z_i \quad (3.7)$$

$$P_k^- = \sum_{i=0}^{2n} W_i^{(c)} \{Y_i - x_k^-\} \{Y_i - x_k^-\}^T \quad P_{zz,k} = \sum_{i=0}^{2n} W_i^{(c)} \{Z_i - z_k^-\} \{Z_i - z_k^-\}^T \quad (3.8)$$

$$P_{vv} = P_{zz} + R \quad (3.9)$$

The Kalman gain K_k represents the weighted degree of confidence the system has in the process and measurement models respectively, and is calculated with the cross-correlation of $\{Y_i\}$ and $\{Z_i\}$, P_{xz} , which aids in understanding how the process and measurement data influence each other.

$$P_{xz} = \sum_{i=0}^{2n} W_i^{(c)} \{Y_i - x_k^-\} \{Z_i - z_k^-\}^T \quad (3.10)$$

$$K_k = P_{xz} P_{vv}^{-1} \quad (3.11)$$

Finally the mean and covariance of the system can be updated as the Kalman weighted sum of the process and measurement estimate.

$$x_k = x_k^- + K_k (z - z_k^-) \quad (3.12)$$

$$P_k = P_k^- - K_k P_{vv} K_k^T \quad (3.13)$$

Process and measurement models

The Fast UKF filter is responsible for estimating the robot's position given its controller inputs and correcting the estimate with IMU sensor data. The motion model of the system with state

$\mathbf{x}_t^- = \{x_t^-, y_t^-, \theta_t^-, v_t^-, \omega_t^-\}$ can be defined as

$$\mathbf{x}_t^- = A(\mathbf{x}_{t-1}) \quad (3.14)$$

$$x_t^- = x_{t-1} + v_{t-1} \Delta t \cos(\theta_{t-1} + \omega_{t-1} \Delta t) \quad (3.15)$$

$$y_t^- = y_{t-1} + v_{t-1} \Delta t \sin(\theta_{t-1} + \omega_{t-1} \Delta t) \quad (3.16)$$

$$\theta_t^- = \theta_{t-1} + \omega_{t-1} \Delta t \quad (3.17)$$

$$v_t^- = v_{t-1} \quad (3.18)$$

$$\omega_t^- = \omega_{t-1} \quad (3.19)$$

assuming constant translational and angular velocity v and ω for short timespans Δt . The fast filter has to operate at relatively high frequencies (20 Hz in this case) for the process model's estimate to be theoretically accurate. What the motion model implies is that the present position and orientation of the robot can be accurately estimated over short timespans, given the previous position and orientation, as well as the forward and angular velocities. Not only that but if the velocities $\{v, \omega\}$ are corrected by the measurement model, and then used to calculate the position, the position will also be corrected.

This might seem straightforward, but it simplifies our approach to sensor data, as accelerometers, gyroscopes and odometers naturally work with velocities or can easily be converted to it, leading to the fast measurement model with state $\mathbf{z}_t^- = \{z_{x,t}^-, z_{y,t}^-, z_{\theta,t}^-, z_{v,t}^-, z_{\omega,t}^-\}$ and accelerometer and gyroscope sensor data $\mathbf{z}_t = \{a_x, g, \psi\}$:

$$\mathbf{z}_t^- = H(\mathbf{x}_{t-1}, \mathbf{z}_t) \quad (3.20)$$

$$z_{v,t}^- = v_t^- + a_x \Delta t \quad (3.21)$$

$$z_{\omega,t}^- = g \psi \quad (3.22)$$

$$z_{x,t}^- = x_{t-1} + z_{v,t}^- \Delta t \cos(\theta_{t-1} + z_{\omega,t}^- \Delta t) \quad (3.23)$$

$$z_{y,t}^- = y_{t-1} + z_{v,t}^- \Delta t \sin(\theta_{t-1} + z_{\omega,t}^- \Delta t) \quad (3.24)$$

$$z_{\theta,t}^- = \theta_{t-1} + z_{\omega,t}^- \Delta t \quad (3.25)$$

The slow filter incorporates the same form for the measurement model, with a different method for calculating forward and angular velocity. Intuitively, it makes sense that forward velocity will be a function of the combined wheel velocity, and angular velocity will in some way be a function of the difference in wheel velocities. This can be expressed as:

$$z_{v,t} = \frac{v_{r,t} + v_{l,t}}{2} \quad (3.26)$$

$$z_{\omega,t} = \frac{v_{r,t} - v_{l,t}}{l} \quad (3.27)$$

for right and left wheel velocities v_r and v_l and robot chassis length l . The position elements are calculated in the same way as the fast measurement model in Equations 3.23 to 3.25. To compare apples to apples, the previous state \mathbf{x}_{t-1} used in the fast and slow filter is different.

The fast filter uses the estimate obtained in the previous fast filter iteration, $\tau_f = \frac{1}{20} = 0.05$ seconds ago, while the slow filter uses the estimate obtained in the previous slow filter iteration, $\tau_s = \frac{1}{4} = 0.25$ seconds ago. Thus, the subsystem has to store the previous estimate of both iterations.

As hinted to earlier, the sigma points $\{X_i\}$ used in the respective filters are generated once every iteration of the slow filter, as this allows the final estimate $\{Y_i\}$ from the fast filter's 5 iterations on the same sigma points $\{X_i\}$ to be used directly as the process model estimate in the slow filter. It is crucial that the slow process model does not try to estimate the new state from $\{X_i\}$, as the quality of the estimate degrades rapidly with increasing timespan Δt . Figure 2 describes the flow of states in the system.

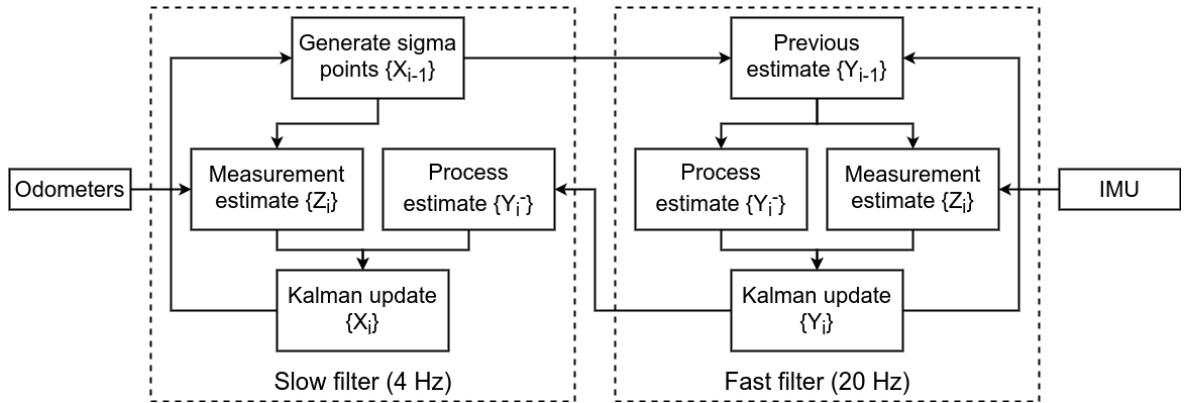


Figure 2.
Sigma points generations and transformation between the slow and fast filters

Slow UKF Model

3.2.2 Mapping

Localization by itself does not present the robot with enough information to act intelligently. The robot should have an idea about how the environment looks and works, not necessarily in extreme detail, but at least clear enough so that the robot can plan paths to specific goals and prevent collisions with obstacles in the environment. The robot needs a map, either as a point-based representation of obstacles or as a planar grid map describing the occupancy of each position.

The latter representation is known as an occupancy grid map, a two dimensional grid of cells, where each cell has a value between 0 (free) and 1 (occupied). Even though the graph represents binary occupancy, i.e. it assumes that each cell is either completely free or completely occupied, a cell's value will typically be a decimal value between the two extremes. This is necessary to account for the uncertainties in the sensor readings perceiving the environment. Sonar range technology as employed in this project can be quite inaccurate and is prone to outlier vulnerabilities. By storing the probability that a cell is occupied rather

than just a zero or one, the system is more flexible to deal with uncertainties by updating the map based on recurrent consensus of the occupancy of a cell over time. The probabilistic representation of the map is also desirable as it can be processed with Bayesian filtering.

Mapping large areas can quickly become large, complex problems. Therefore, to simplify the problem, we assume three things about the nature of the world and the map. First, it is assumed that every cell is either completely occupied or completely free, which is known as the occupancy assumption. Secondly, it is assumed that the world is static, no dynamic object except for the robot traverses the state space, which is known as the static assumption. Thirdly it is assumed that each cell is independent of one another, which is known as the independence assumption. It is clear that these assumptions do not always hold, a person can easily walk through the mapped area and it is quite reasonable to believe that some cells are partially covered or that the presence of an obstacle in one cell affects the probability of an obstacle in the next one. Fortunately the inaccuracies introduced by these assumptions are usually negligible, while the incorporation of these assumptions into the model produces a vastly simplified problem to solve.

Static Binary Bayes Filter

Given sensor data $z_{1:t} = \{z_1, z_2, \dots, z_t\}$ and input controls $u_{1:t} = \{u_1, u_2, \dots, u_t\}$, the objective of the mapping algorithm is to calculate the most likely map m^* , where m^* is the maximum likelihood estimate of map given the input data:

$$m^* = \operatorname{argmax}_m p(m|z_{1:t}, x_{1:t}) \quad (3.28)$$

using the independence assumption $p(m_j|m_i) = p(m_j)$ for cells i and j in the map, the problem reduces to a joint probability or product of the individual cells, which can be expressed using Bayes rule,

$$p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i|z_{1:t}, x_{1:t}) \quad (3.29)$$

$$p(m_i|z_{1:t}, x_{1:t}) = \frac{p(z_t|m_i, z_{1:t-1}, x_{1:t})p(m_i|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1}, x_{1:t})} \quad (3.30)$$

which describes the probability that a cell is occupied given sensor data proportional to the probability of the sensor data given that the cell is occupied. A useful property of Bayesian network is the Markov assumption, which states that the future state of the system is conditionally independent of past states, given the current state. Thus, equation 3.30 can be simplified by removing all past sensor data, $z_{1:t-1}$, given current data z_t and by disregarding future input controls, x_t , given the current map m_i .

$$p(m_i|z_{1:t}, x_{1:t}) = \frac{p(z_t|m_i, x_t)p(m_i|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1}, x_{1:t})} \quad (3.31)$$

Bayes rule can be applied once again to the first term in the numerator, and multiplied back into the equation

$$p(z_t|m_i, x_t) = \frac{p(m_i|z_t, x_t)p(z_t|x_t)}{p(m_i|x_t)} \quad (3.32)$$

$$p(m_i|z_{1:t}, x_{1:t}) = \frac{p(m_i|z_t, x_t)p(z_t|x_t)p(m_i|z_{1:t-1}, x_{1:t-1})}{p(m_i|x_t)p(z_t|z_{1:t-1}, x_{1:t})} \quad (3.33)$$

A few terms in this equation are difficult to estimate, such as $p(z_t|x_t)$, which estimates the probability of a sensor given a position with no mapping information, and $p(m_i|x_t)$, which estimates the probability of a cell's occupancy given a position without sensor data. The second term can be simplified by assuming the map is independent of a robot's position in the absence of sensor data, $p(m_i|x_t) = p(m_i)$, which represents the prior probability that any cell in a map is occupied. This prior would be high in a thickly crowded environment and low in an area with large open spaces.

As the occupancy grid map represents binary data, i.e. each cell is either occupied or free, the Bayesian filter for the opposite event is valid, and thus the ratio of the two probabilities can be expressed and simplified as:

$$p(\neg m_i|z_{1:t}, x_{1:t}) = \frac{p(\neg m_i|z_t, x_t)p(z_t|x_t)p(\neg m_i|z_{1:t-1}, x_{1:t-1})}{p(\neg m_i)p(z_t|z_{1:t-1}, x_{1:t})} \quad (3.34)$$

$$\frac{p(m_i|z_{1:t}, x_{1:t})}{p(\neg m_i|z_{1:t}, x_{1:t})} = \frac{\frac{p(m_i|z_t, x_t)p(z_t|x_t)p(m_i|z_{1:t-1}, x_{1:t-1})}{p(m_i)p(z_t|z_{1:t-1}, x_{1:t})}}{\frac{p(\neg m_i|z_t, x_t)p(z_t|x_t)p(\neg m_i|z_{1:t-1}, x_{1:t-1})}{p(\neg m_i)p(z_t|z_{1:t-1}, x_{1:t})}} \quad (3.35)$$

$$= \frac{p(m_i|z_t, x_t)p(m_i|z_{1:t-1}, x_{1:t-1})p(\neg m_i)}{p(\neg m_i|z_t, x_t)p(\neg m_i|z_{1:t-1}, x_{1:t-1})p(m_i)} \quad (3.36)$$

The occupancy assumption simplifies $p(\neg m_i) = 1 - p(m_i)$ as

$$\frac{p(m_i|z_{1:t}, x_{1:t})}{1 - p(m_i|z_{1:t}, x_{1:t})} = \frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)} \frac{p(m_i|z_{1:t-1}, x_{1:t-1})}{1 - p(m_i|z_{1:t-1}, x_{1:t-1})} \frac{(1 - p(m_i))}{p(m_i)} \quad (3.37)$$

From Equation 3.37 it is evident that the system consists of three parts. The $p(m_i|z_t, x_t)$ term describes the influence of the current sensor observation z_t , the $p(m_i|z_{1:t-1}, x_{1:t-1})$ term describes the influence of the estimate from the previous state of the cell, and the $p(m_i)$ term describes the influence of the a priori information of the map, independent of sensor measurements.

Equation 3.37 currently describes the odds ratio $Odds(x) = \frac{p(x)}{1 - p(x)}$ of the probability. Using simple algebra the probability $p(x)$ can be recovered.

$$Odds(x) = \frac{p(x)}{1 - p(x)} \quad (3.38)$$

$$p(x) = Odds(x) - Odds(x)p(x) \quad (3.39)$$

$$p(x)(1 + Odds(x)) = Odds(x) \quad (3.40)$$

$$p(x) = \frac{Odds(x)}{1 + Odds(x)} = \frac{1}{1 + \frac{1}{Odds(x)}} \quad (3.41)$$

which finally leads to the equation for $p(m_i|z_{1:t}, x_{1:t})$,

$$p(x) = [1 + Odds(x)]^{-1} \quad (3.42)$$

$$p(m_i|z_{1:t}, x_{1:t}) = \left[1 + \frac{1 - p(m_i|z_t, x_t)}{p(m_i|z_t, x_t)} \frac{1 - p(m_i|z_{1:t-1})}{p(m_i|z_{1:t-1}, x_{1:t-1})} \frac{p(m_i)}{(1 - p(m_i))} \right]^{-1} \quad (3.43)$$

Log-Odds Notation

However, applying the latter equation to every cell in a grid map can quickly become computationally expensive, and given that the mapping algorithm has to coexist with other algorithms on a resource constrained microcontroller, it is imperative to optimize the algorithm even further.

First, let us define the log-odds of the probability of x and its inverse as

$$l(x) = \log \frac{p(x)}{1 - p(x)} \quad p(x) = 1 - \frac{1}{1 + e^{l(x)}} \quad (3.44)$$

representing the map in log-odds space reduces the mapping equation to

$$l(m_i|z_{1:t}, x_{1:t}) = l(m_i|z_t, x_t) + l(m_i|z_{1:t-1}, x_{1:t-1}) - l(m_i) \quad (3.45)$$

$$l_{t,i} = h(m_i, x_t, z_t) + l_{t-1,i} - l_0 \quad (3.46)$$

where $h(x)$ is the measurement model that relates a new measurement to a log-odds probability. Equation 3.46 clearly shows the recursive nature of the algorithm, as each update step just involves an addition of the new sensor data probability and a subtraction of the prior probability. Updating a log-odds map is fast, efficient and even allows parallelism, as none of the map cells are dependent on one another.

Measurement model

The map algorithm only updates the cells that are currently in the field of view of the sonar sensors, which emits a sonar signal in the form characterized in Figure 3.

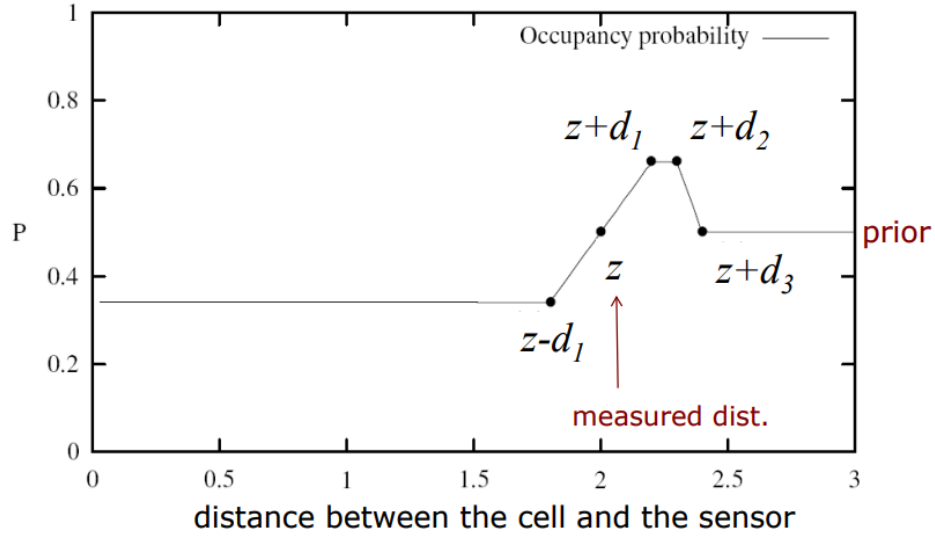


Figure 4.
Distance dependent probability update. Courtesy Burgard [15].

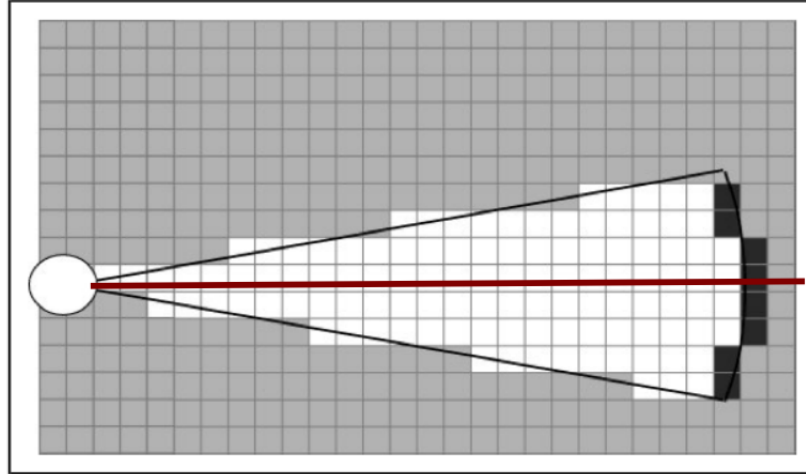


Figure 3.
Map grid cells in field of view of sonar sensor. Courtesy Burgard [15].

A few properties of the sensor is apparant from Figure 3. Firstly, the sonar does not emit a directed beam in a specific direction, but has a deviation angle θ from which an emitted beam can be reflected back. Secondly, the sensor only detects the nearest object in its field of view (FOV), regardless of the view's width. This can be shown by the arc of dark cells at the end of the beam, any of which can be the location of the real object. To deal with uncertainty in the measurement, a cell's modified probability depends on the cell's relative distance to the perceived obstacle and its deviation from the center of the sensor's orientation.

Figure 4 describes the occupancy confidence of cells in the FOV, based on on where they are on the sensor distance line. The sensor confidence can be expressed in terms of constants

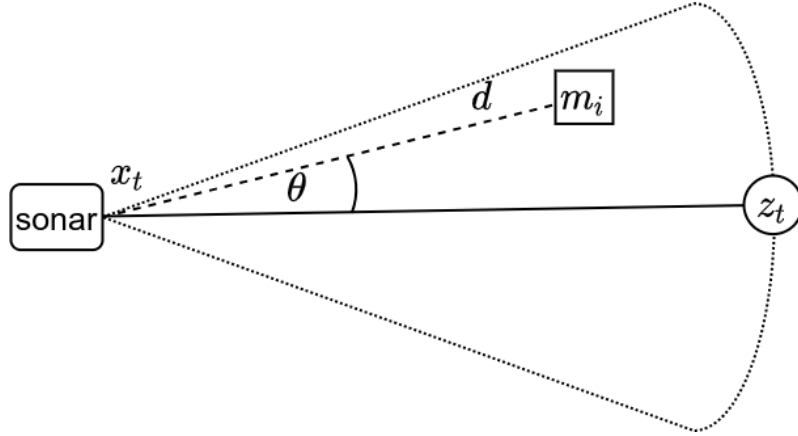


Figure 5.
Angle dependent probability update. Courtesy Burgard [15].

d_1, d_2, d_3 , which estimate the sensor noise and general thickness of obstacles. Cells that lay between x_t and $z_t - d_1$ are perceived to be unoccupied, and will have a low probability, cells between $z_t - d_1$ and $z_t + d_1$ gradually increase in occupancy confidence as the measured distance approaches. Cells between $z_t + d_1$ and $z_t + d_2$ are regarded as obstructed, even with noise in the sensor it is reasonable to assume an obstruction d_1 further from the reading. Finally cells between $z_t + d_2$ and $z_t + d_3$ technically behind the sensed obstacle and thus no information about their occupancy is available. The same is true for cells beyond $z + d_3$. This can be evaluated in a piecewise function as

$$d = \sqrt{(m_{i,x} - x_{t,x})^2 + (m_{i,y} - x_{t,y})^2} \quad (3.47)$$

$$p(m_i) = \begin{cases} p_{LOW} & d \leq z_t + d_1 \\ p_{LOW} + \frac{p_{HIGH} - p_{LOW}}{2d_1}(d - z_t - d_1) & z_t - d_1 < d \leq z_t + d_1 \\ p_{HIGH} & z_t + d_1 < d \leq z_t + d_2 \\ p_{HIGH} - \frac{p_{HIGH} - p_0}{d_3 - d_2}(d - z_t + d_3) & z_t + d_2 < d \leq z_t + d_3 \\ p_0 & d > z_t + d_3 \end{cases} \quad (3.48)$$

Figure 5 shows a cell that is deviated from the center of the sonar's FOV by θ . Assuming that sensor readings at a deviated angle will be less accurate and more prone to errors, it is wise to multiply the proposed confidence with a Gaussian distributed weight around the deviation θ . The probability density function $f(x)$ of a normally distributed set equates to

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (3.49)$$

with standard deviation σ and mean $\mu = 0$, which can be used to calculate the Gaussian weight s_θ as

$$s_\theta = \begin{cases} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\theta^2}{2\sigma^2}} & \theta \leq 0 \\ \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(1-\theta)^2}{2\sigma^2}} & \theta > 0 \end{cases} \quad (3.50)$$

Finally the measurement model can be expressed as

$$h(m_i, x_t, z_t) = \log \frac{s_\theta p(m_i)}{1 - s_\theta p(m_i)} \quad (3.51)$$

Modern approaches tend to integrate the localization and mapping problems in one of several SLAM algorithms. This approach was not pursued intentionally as those algorithms require a lot of computational power and quite accurate distance readings. The proposed system has neither, working with a low-end microcontroller and low quality sonar sensors, but rather maps the world with known poses, thus it assumes localization is fairly accurate. The focus now shifts from a highly complex simultaneous optimization problem to ensuring accurate positioning, which is done through the incorporation of three different sensors in a sophisticated UKF for precise state estimation.

3.2.3 Control

The passive aspects of the system have been dealt with, and the robot can effectively estimate its position and perceive the environment, but so far it is unable to plan paths nor act on those plans. In this section we assume a path has already been determined, and investigate the methods the robot employs to reach this goal accurately and timely.

The simplest control model aims to reach a desired state x' from its current state x by sending a set of controls u to its actuators, producing state y . This model does not incorporate any perceptual information about the environment and is an open loop controller. This model is vastly insufficient for any system operating in the real world, as it does not account for noise, faulty actuators or dynamic unexpected events. A better model also incorporates sensor data to update the estimate of its state in what is known as a closed loop feedback controller.

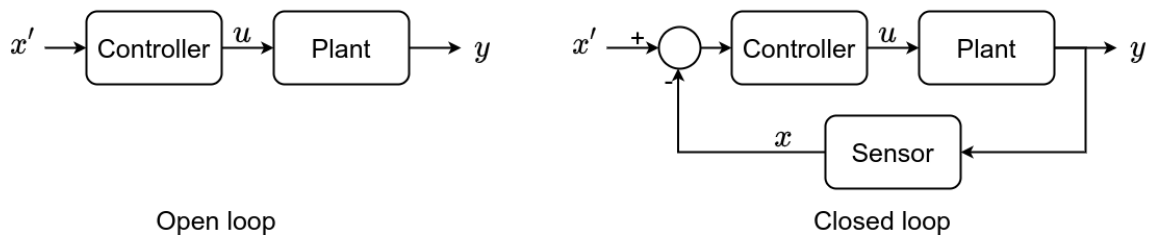


Figure 6.
Open loop vs. closed loop controller

A closed loop controller continually uses sensor data to update the error, e , between the desired and current states. The data flow model for a closed loop controller is given in Figure 7, showing the interaction between the desired state x'_t , the current state x_t , the measured state z_t and the input controls u_t . Process noise ϵ_t and measurement noise δ_t also affect the system. The controller's objective is to relate the error between the desired and current state to the required input controls that will eventually result in the robot reaching the goal.

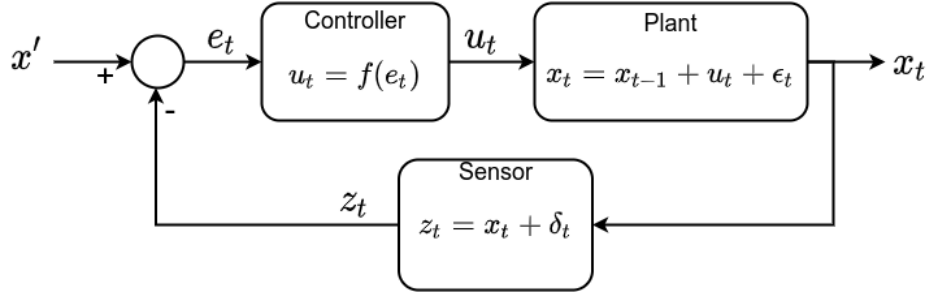


Figure 7.
Closed loop controller data flow model

PID controller theory

Let us first examine a generic feedback controller with the above-mentioned parameters. The simplest controller would set the input proportional to the error,

$$u_t = K_p(x' - x_t) \quad (3.52)$$

$$= K_p e_t, K_p < 1 \quad (3.53)$$

where K_p is the proportionality constant. Limiting K_p to be less than one ensures that the system produces an input smaller than its perceived error, leading to a stable system response. On the other hand, if $K_p > 1$ the system would produce input larger than the error, which can lead to positive feedback and instability.

The controller in Equation 3.53 works theoretically, but three physical phenomenon can degrade the quality of the system response. Firstly, the system is susceptible to noise, both in the actuators and in the sensors, which can lead to oscillations around the desired goal state, second a delay between the system's response and its perception of the response can lead to overcompensation for an action that has already been adjusted for, but due to inertia or sensor characteristics the effect is not seen yet. Thirdly, in the case where the system controls for the rate of change of a particular variable, such as the velocity of the robot's position, just reaching the desired position is not sufficient, as the system's velocity will cause it to overshoot. Rather the system should act in a manner to compensate for both its state and the rate of change of the state, to reach the goal state and remain there.

Noise and delay can be reduced by setting a lower value of K_p , but it can lead to a slower convergence rate. This is effectively a trade-off between fast convergence and low steady-state error, depending on the design requirements. In the latter case the system can be modelled

more accurately according to Equation 3.54, accounting for both its state and its rate of change or derivative. This will ensure that the input is not only dependent on the magnitude of the error, but also the rate at which we are reducing the error, to ensure low oscillatory behaviour in the vicinity of the goal. As a bonus, the introduction of a derivative term facilitates the use of a larger stable proportional constant, leading to faster convergence.

$$x_t = x_{t-1} + \dot{x}\Delta t \quad (3.54)$$

$$u_t = K_p(x' - x_t) + K_d(\dot{x}' - \dot{x}_t) \quad (3.55)$$

$$u_t = K_p e_t + K_d \frac{de_t}{dt} \quad (3.56)$$

A final modification can be made to the controller by considering the problem of steady state errors. These are small errors in the state when in the vicinity of the goal that accumulate over time to take the system off course. This is, for example, applicable if the desired state is a certain velocity to reach a faraway target, but due to a small deviation the target is not reached. Steady state errors are usually small enough that the proportional controller do not correct for them, or at least cannot do so effectively, as it is designed to deal with large inaccuracies. The solution is to correct for the integral of the error over time as part of the controller.

$$u_t = K_p(x' - x_t) + K_d(\dot{x}' - \dot{x}_t) + K_i \int_0^t (x' - x_t) dt \quad (3.57)$$

$$u_t = K_p e_t + K_d \frac{de_t}{dt} + K_i \int_t e_t dt \quad (3.58)$$

The introduction of the integral term does however put the system in danger of the error build up, also known as the wind-up effect. If the robot does not act in the way it ought to, according to its model, which can happen if it is for example stuck in a hole. The integral error will continue to increase dramatically and completely disrupt the system once the robot comes out of the hole. The solution is to only integrate the error over fixed periods of time instead of over all possible time, which will at least contain the integral error to an upper manageable limit.

Motor control

With the theoretical foundation laid, we can proceed to the mathematical models of the motor and position controllers. The motor controller restricts its operation to the performance and expectation of the motors, and aims to reduce the error between the real and desired translational and rotational velocities (v, ω). This controller not only assumes that the desired position is known, but also that the position controller, described below, established the required velocities (v, ω) to reach the goal. The motivation for the separation of the two systems is that it produces a two modular, independent software components that can function semi-independent of one another, which eases development, debugging and maintenance. For the motor controller this allows a simplistic and focused model, and for the position controller this abstracts away the physical implementation of the velocities, once again simplifying and focusing its model.

The motor controller has state $\mathbf{x} = \{v, \omega\}$, where v describes the forward velocity and ω the angular velocity of the robot. The desired state is of the same form $\mathbf{x}' = \{v', \omega'\}$. The input controls are the duty cycles that drive the PWM of the left and right motors, $\mathbf{u} = \{u_l, u_r\}$, and must have a value between -1 and 1. The sensors are the light-based odometers attached to the shafts of each wheel, which record the amount of holes n in a wheel encoder that passes by in a specific time frame Δt . This is easily converted angular velocity of the wheel, ω_i , and forward velocity, v_i , as

$$v_i = r\omega_i = \frac{2\pi rn}{N\Delta t} \quad (3.59)$$

$$(3.60)$$

with N the total amount of holes in the encoder and r the radius of the wheels. The error forward and angular velocity is calculated as $\mathbf{e} = \{e_v, e_\omega\}$, using the chassis length l . The motor controller implements a simple proportional controller in as Equation 3.62.

$$e_v = v' - \frac{v_r + v_l}{2} \quad e_\omega = \omega' - \frac{v_r - v_l}{l} \quad (3.61)$$

$$\begin{bmatrix} u_r \\ u_l \end{bmatrix} = K_p \begin{bmatrix} 1 & K_\omega \\ 1 & -K_\omega \end{bmatrix} \begin{bmatrix} e_v \\ e_\omega \end{bmatrix} \quad (3.62)$$

In essence the controller uses the translational velocity to set a velocity bar for both wheels, which is offset one way or another depending on the direction and magnitude of the desired angular velocity.

Position control

The position controller aims to move the robot from its current position to its desired position by dynamically updating the required translational and angular velocities needed to get there. The controller has state $\mathbf{x} = \{x, y, \theta\}$, describing the current position and orientation of the robot in 2D state space, as well as desired state $\mathbf{x}' = \{x', y'\}$ as the goal and $\mathbf{u} = \{v, \omega\}$ as the input controls to the motor controller.

The first step is to obtain the distance and angular offset error between the current and desired state, which effectively converts the error to polar form.

$$e_t = f(\mathbf{x}' - \mathbf{x}) \quad (3.63)$$

$$\begin{bmatrix} e_r \\ e_\theta \end{bmatrix} = \begin{bmatrix} \sqrt{(x' - x)^2 + (y' - y)^2} \\ \text{atan2}(y' - y, x' - x) - \theta \end{bmatrix} \quad (3.64)$$

the C function $\text{atan2}()$ is used as it respect the sign of the angle. Using the previous error states $e_{1:t-1}$, the integral and derivative terms can be calculated as

$$\int_0^t e_t dt = \sum_{i=0}^t e_i \Delta t = \sum_{i=0}^{t-1} e_i \Delta t + e_t \Delta t \quad (3.65)$$

$$\frac{de_t}{dt} = \frac{e_t - e_{t-1}}{\Delta t} \quad (3.66)$$

where the $\sum_{i=0}^{t-1} e_i \Delta t$ is the sum of all previous errors up to e_{t-1} and is updated recursively. Finally the PID-controller can be implemented as

$$u_t = K_p e_t + K_i \int_0^t e_t dt + K_d \frac{de_t}{dt} \quad (3.67)$$

$$\begin{bmatrix} u_v \\ u_\omega \end{bmatrix} = \begin{bmatrix} K_{p,r} \\ K_{p,\theta} \end{bmatrix} \begin{bmatrix} e_{t,r} \\ e_{t,\theta} \end{bmatrix} + \begin{bmatrix} K_{i,r} \\ K_{i,\theta} \end{bmatrix} \begin{bmatrix} \int_0^t e_{t,r} dt \\ \int_0^t e_{t,\theta} dt \end{bmatrix} + \begin{bmatrix} K_{d,r} \\ K_{d,\theta} \end{bmatrix} \begin{bmatrix} \dot{e}_{t,r} \\ \dot{e}_{t,\theta} \end{bmatrix} \quad (3.68)$$

There exists a inverse relationship between required velocity and estimated distance, thus to make sure velocity decreases with approaching speed, as well as to confine the suitable velocities to the hardware constraints of the system, the input controls are propagated through the following equation before being used in the motor control.

$$u'_{t,v} = \frac{K_c u_{t,v}}{K_a + u_{t,v}} \quad (3.69)$$

where K_c represents the hardware defined upper velocity limit, and K_a is a modifier describing how the velocity should taper off as the robot approaches the goal position.

3.2.4 Path planning

Configuration space

Abstracting another layer we enter the domain of path planning. The goal of this subsystem is to generate and maintain a viable route to a exploration goal which is aimed at maximum information and discussed in section 3.2.5. For now, we can assume the goal position has been set, but is potential on the other side of obstructed space.

The motor and position controllers operate in state space, in other words both controllers gather and manipulate states that corresponds to the internal Cartesian world map of the system. This is of course convenient when working with Euclidean distances and relative angles as is abundant in those subsystems. However, abstracting another layer to the point where mapping information becomes relevant, a new operating space is necessary to keep the problem small and tractable. The occupancy grid map used by the mapping subsystem gives a hint of what is necessary for planning, but at least two problems arise from simply using the occupancy map in path planning algorithms, including that the physical properties of the robot, such as its width and length are not taken into account.

To see why this is a potential problem, consider the robot planning a path to the goal in Figure 8. In frame 1 a state space search algorithm is used to obtain a path from the robot's position

A to the goal X. The occupancy grid map creates grid cells at a smaller resolution than the size of the robot, as this allows for a more refined model of the environment. However, if a search algorithm were to use the occupancy of the map to determine if the a route is feasible, a collision is possible, as seen in frame 2 with the robot at position B, when the the space between obstacles is larger than a cell's width, but smaller than the robot's width.

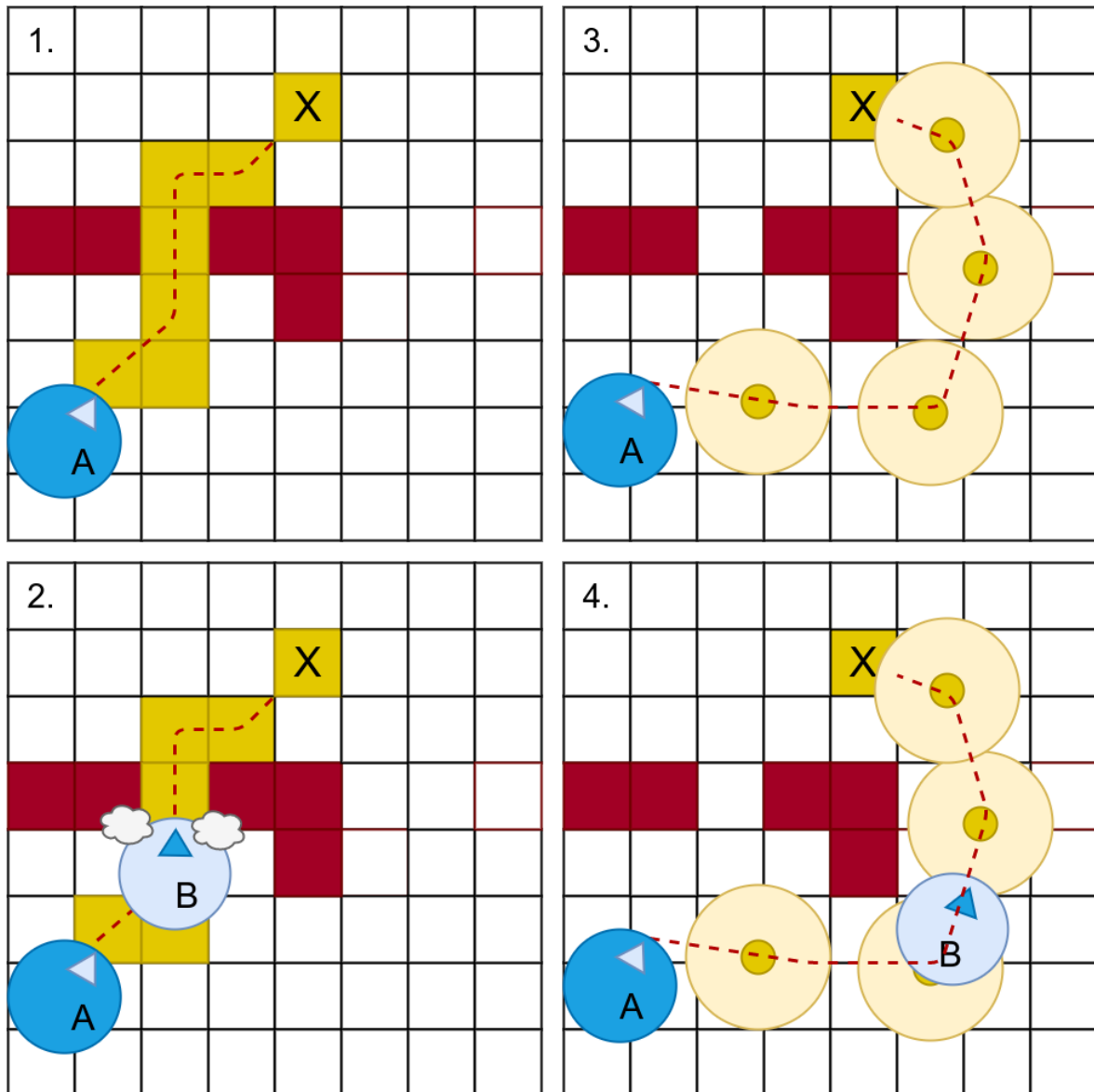


Figure 8.

A possible problem of using the occupancy grid map, evident in the collision in frame 2

In contrast, a typical configuration space representation of the environment will not confine itself to grid cells, nor search in it, but rather create nodes which are further apart from the cells and evaluate a larger area around the node's position for obstacles. The greater distance between nodes reduces redundancy, as the path planner does not want to consider positions too tightly confined to the resolution of the map, but rather at a distance more suitable to the physical

characteristics of the robot, which in this case is approximately the length of the robot. Frame 3 and 4 shows the planned route obtained by searching in configuration space, which contains fewer nodes and steers clear from obstacles.

Collision detection

The path planner deals with collision detection, both in the present and in the future. Even though the system would ideally never collide with obstacles, there is always a possibility that due to the nature of the environment or some other stochastic process, the system fails to steer clear of an obstacle and collides. It is therefore wise to at least design a backup strategy so that the system can recover from collisions gracefully. The sonar sensors are unable to sense objects directly in front of it, and thus would be of little help to detect a possible collision. Instead, the planner evaluates the expected and actual state of a system over time to detect abnormalities that could signal a collision.

Intuitively it makes sense that a collided obstacle will not be able to move in its desired direction, therefore it will be unable to move forward or unable to turn towards its goal. This can easily be evaluated by checking if there is a noticeable gap between the expected velocities $u = \{u_v, u_\omega\}$ of the motor controller, and the measured wheel angular rate $v = \{v_r, v_l\}$ from the odometers, as well as the angular rate g_ω from the gyroscope. Algorithmically this can be described as

```

if ( $u_v > R_v$  and ( $v_r + v_l = 0$ ) or ( $|u_\omega| > R_\omega$  and  $|g_\omega| < R_\omega/2$ ) then
    collisionCount  $\leftarrow$  collisionCount + 1
else
    collisionCount  $\leftarrow$  0
end if
if collisionCount =  $R_{cc}$  then
    collision occurred
    collisionCount  $\leftarrow$  0
end if

```

With velocity and angular rate thresholds, R_v and R_ω . By testing the expected and measurement data above and below a threshold, the collision detector can be configured to behave as desired. There is a danger of preemptively deeming a system stuck. If the system receives, for example, a new motor controller command from rest, and then immediately calls the collision detection function, it will perceive the system as stuck, as the system has a desired goal but is not moving. To overcome this problem, the system is not assumed to have collided initially, but only after K_{cc} sequential iterations of the detector the system is still unable to move.

The abovementioned approach allows the system to recover from collisions. However, the scenario should ideally be avoided altogether by carefully tracking the movement of the robot through the environment, making predictions about its future trajectory and avoiding paths and positions that are likely to result in collisions. The trajectory of the robot is predicted in the same way as the process model of the localization filter, using the motion model in Equations 3.15 - 3.19.

The model operates at timespan Δt , which can vary according to the system requirements for the model. As $\Delta t \rightarrow 0$, the model becomes more accurate, but also requires more iterations for a fixed timespan t . The model estimates future states accurately enough for the purposes

of collision detection if iterated at 20 Hz, the frequency of the system, for no more than one second into the future. Collision with obstacles in the grid cells at and around the estimated location is checked at each timestep until a collision is detected or the maximum iterations are reached collision-free.

A* search

Planning a path from the current state to a goal state requires a method of navigating the state space (or in this case the configuration space) to reach the goal effectively and quickly. In Artificial Intelligence (AI) there exists several search techniques that can broadly be grouped into two categories: informed and uninformed search algorithms.

Uninformed search techniques do not consider any information about the domain other than the problem definition. These methods blindly search the state space hoping to reach the goal. Some of the most prominent algorithms in this category are breadth-first search, depth-first search and uniform cost search. In the uniform cost search, each node not only knows its position in the state space, but also the cost to get there, which, by always expanding the node with the lowest accumulated cost, ensures an optimal solution. Nonetheless, these algorithms are potentially very inefficient and slow, wasting time and resources on paths diverging from the goal more often than not. A better solution is needed for rapid convergence of the problem.

The solution is to implement an informed search technique, in other words an algorithm that not only keeps track of its cost, but also defines a metric of its proximity to the goal state, known as a heuristic. This requires a level of understanding about the state space and the goal. In the case of two dimensional path planning, the distance between the goal and current state will generally form part of the heuristic function, as this is relevant data, measuring the value of the cell in a manner coherent with the representation of the search space. An important requirement for optimality in informed search is that the heuristic function should never overestimate remaining cost to the goal, but rather underestimate or estimate exactly. The closer the heuristic is to the true proximity to the goal, the more effective and useful it is.

A* search, originally developed for robotic path planning applications in the 1960s, is an informed search strategy that is both optimal and complete, i.e. it will provide a solution if one exists and it will be the best possible solution given the cost and heuristic functions. In two dimensional geometric path planning, the heuristic function $h(n)$ is the Euclidean distance between the current node n and the goal node n' , whereas the cost function $g(n)$ is the accumulated cost of getting to the current node from the start node.

$$g(n) = g(n-1) + c(n) \quad (3.70)$$

$$f(n) = g(n) + h(n) \quad (3.71)$$

with $g(n-1)$ the parent node's cost, $c(n)$ the cost to get from the parent to the current node and $f(n)$ the function of the node, which gives an indication of its utility in finding the node. The A* algorithm uses two queues, an open queue O and closed queue C to keep track of visited and potential nodes. The algorithm starts by adding the current position of the robot to O . It then enters a loop which will continue to evaluate potential nodes in O until the goal node is found or until all possible nodes have been searched with no success and O is empty.

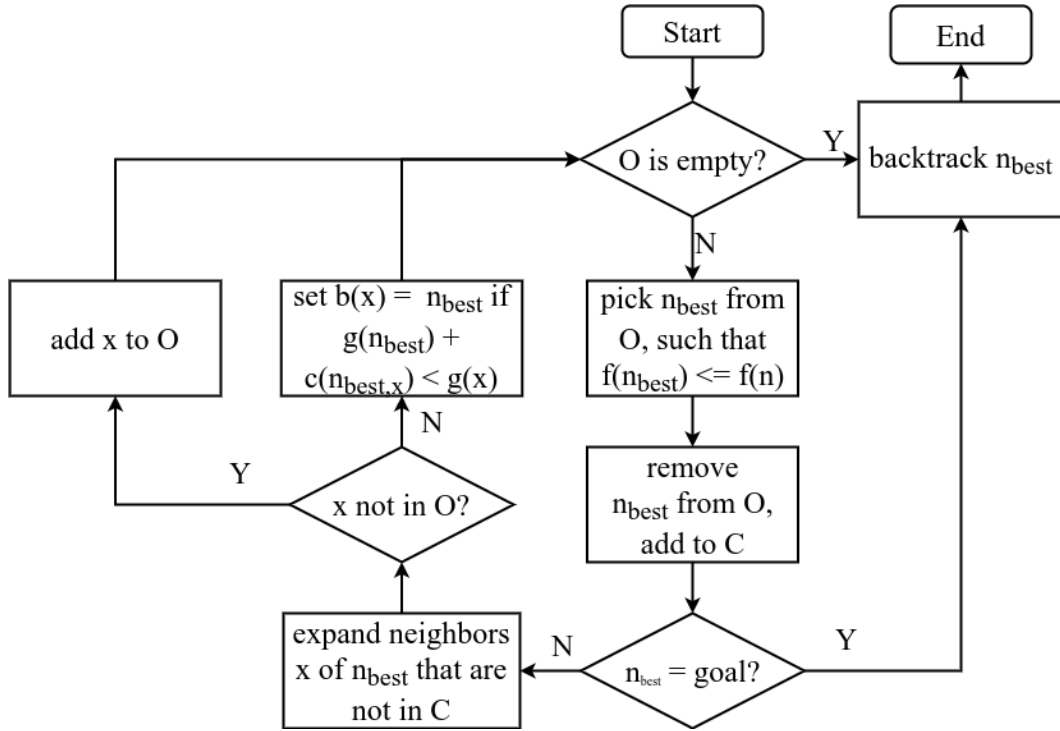


Figure 9.
A* flow diagram. Courtesy Burgard [15].

The most promising node n_{best} in O is always evaluated, indicated by the lowest utility $f(n_{best} \leq f(n), n \in O$. This node is removed from O and placed in C . If it is closest to the goal node, indicated by having a Euclidean distance to the goal smaller than a desired threshold, the search stops and backtracking starts. If not, each neighbors x of n_{best} is expanded if it is not already in C . x is add to O if not already there, otherwise it is checked if it is possible to reduce the cost to x through n_{best} ,

if $g(n_{best}) + c(n_{best}, x) \leq g(x)$ **then**
 $g(x) \leftarrow g(n_{best}) + c(n_{best}, x)$
 $b(x) \leftarrow n_{best}$
end if

and setting the new cost $g(x)$ and new parent node $b(x)$ if that is the case. In essence, the closed queue C stores nodes into those already visited, with fixed cost and heuristics functions. These are fixed due to the admissibility of the algorithm, which states that the node with the lowest combined heuristic and cost function will be expanded next and subsequently added to C . It is therefore impossible to discover a path to an already visited node that will result in a lower combined cost, as by definition this path would have been explored earlier.

The open queue O , on the other hand, stores nodes seen on the periphery but not visited yet, thus their cost can change if a new node with a lower combined cost can become its parent. The situation is depicted in Figure 10. Node 2 is expanded first due to its greater utility, $f(2) < f(3)$. Node 4 is identified as a possible step towards the goal via node 2 with a cost of $g(4) = 6, f(4) = 6 + 7 = 13$. In the next iteration, node 3 is expanded with the lowest $f(x)$ and

a route to node 4 is discovered with a lower utility function $f'(4) < f(4)$, and thus node 3 is selected as node 4's new parent.

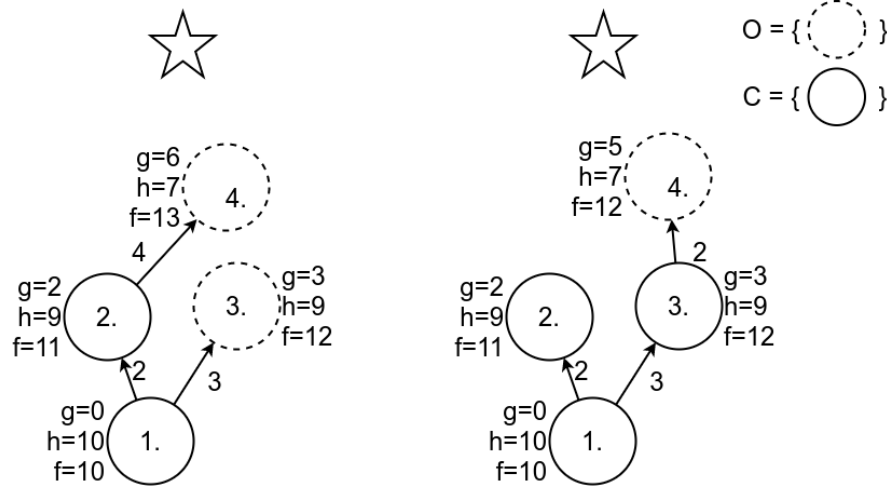


Figure 10.

Example of a node choosing a new parent to minimize its utility function

After the goal node has been reached, the algorithm backtracks through successive parents to create the optimal path from the current position to the goal. During path planning in real two dimensional space, it is important to determine a next node distance that is suitable to the operation of the robot and the nature of the environment. Configuring the nodes to be too close to one another will lead to a large search space and unacceptably long search times for a goal not even too far away. On the other hand, if the nodes are placed too far apart, there is a chance that the straight line path between two "collision-free" nodes contains an obstacle. Thus, the system chooses nodes at approximately the length of the robot, and checks for obstacles around a node in an area large enough to overlap with neighbouring nodes.

3.2.5 Exploration

The final module required for everything to work together is the exploration subsystem. This module is the highest level of abstraction in the navigation component, and is responsible for directing the robot towards areas with high possible information gain. There are many interpretations of information gain and what it means to explore new areas, and thus many solution to this problem have been proposed. This project's implementation is mainly based on the work by Yamauchi [9].

The approach is to focus on identifying a group of edges between known and unknown areas, also known as a frontier, and navigating to it. The intuition lies in assuming that the robot can learn the most about the environment in areas it has not been to. Given that the ultimate goal of the robotic vacuum cleaner is to sweep the whole confined area, this form of navigation will eventually explore all possible parts of the environment, cleaning in the process.

This method is often contrasted with a random approach, such as SRT, which aims to randomly

navigate to new areas of information gain. While this approach has seen many utility in recent years, a design decision was taken not to use SRT, because characteristics of the robot do not complement the requirements of SRT, which could lead to low performance of the subsystem.

The reason is that SRT is inherently localized in its search. The algorithm chooses a point within the safe space created by the last sensor reading which optimizes information gain, and reverts back to a previous position if no new areas are discovered. This approach works best if the system can rely on a large field of view of accurate sensor data. Unfortunately, this is not the case in this project, as the robot merely has three low-cost sonar sensors in a 110° angle FOV. Experimental results show that more often than not, the reliance on short term localized navigation in SRT caused the robot to collide with obstacles. A system using high quality range scanners, SRT might perform good enough to implement as the exploration subsystem.

Edge detection

In probability space, an unexplored cell at index (i, j) of an occupancy grid map will have a value $p(m_{i,j}) \approx 0.5$, indicating that the occupancy of the cell is completely unknown. This cell will also be an edge if one of its neighbouring cells has been explored and found to be open, $p(m_{i,j+1}) \approx 0$. Thus a filter can be constructed that weights the probability that a cell in an occupancy map is an edge. Let us start by highlighting uncertain probabilities, and suppressing certainty at both extremes, using the descending absolute value function centered around 0.5:

$$p(u_{i,j}) = 1 - 2|p(m_{i,j}) - 0.5| \quad (3.72)$$

where $p(u_{i,j})$ represents the probability that the map m at position (i, j) is uncertain. The second term in the equation will be negligible around $p(m_{i,j}) = 0.5$, causing the uncertainty probability to be close to 1. Yet as the value tends to 0 or 1, it will decrease the value of $p(u_{i,j})$. The second part of edge identification involves looking for an unoccupied neighbouring cell. This is important as an uncertain cell in the middle of a large uncertain part of the map is not an edge, thus it is unknown if the position can be reached safely. The same is true for uncertain cells with only unknown or obstructed neighbouring cells, which could be indicative of a wall or obstacle and also not reachable. An open cell can be highlighted by taking the complementary of the minimum probability of all neighbours, expressed as

$$p(-m_{i+k,j+l}) = 1 - p(m_{i+k,j+l}) \text{ for } k, l \in [-1, 1] \quad (3.73)$$

$$p(-n_{i,j}) = \operatorname{argmax}_{k,l \in [-1,1]} (1 - p(m_{i+k,j+l})) \quad (3.74)$$

$$= (1 - \operatorname{argmin}_{k,l \in [-1,1]} p(m_{i+k,j+l})) \quad (3.75)$$

with $p(-n_{i,j})$ the probability that a neighbour is free. Equations 3.72 and 3.75 can be combined to obtain the probability for an edge $p(e_{i,j})$ at (i, j) .

$$p(e_{i,j}) = p(u_{i,j})p(\neg n_{i,j}) \quad (3.76)$$

$$p(e_{i,j}) = (1 - 2|p(m_{i,j}) - 0.5|)(1 - \underset{k,l \in [-1,1]}{\operatorname{argmin}} p(m_{i+k,j+l})) \quad (3.77)$$

However, the occupancy grid map used in this system does not store the occupancy as a probability, but as a log-odds of the probability. The reasoning for this is described in section 3.2.2, but it comes down to computational efficiency. It is possible to directly convert this equation to log-odds form, but log-odds is efficient for a reason, so it might be beneficial to rather recalculate the edges using log-odds space intuition, using the same properties of edges discussed above.

The nature of log-odds representation over the probability spectrum can be seen in Figure 11. Whereas in probability space uncertainty has a value of 0.5, in log-odds space it has a value of 0, with negative values depicting unoccupied and positive values depicting occupied.

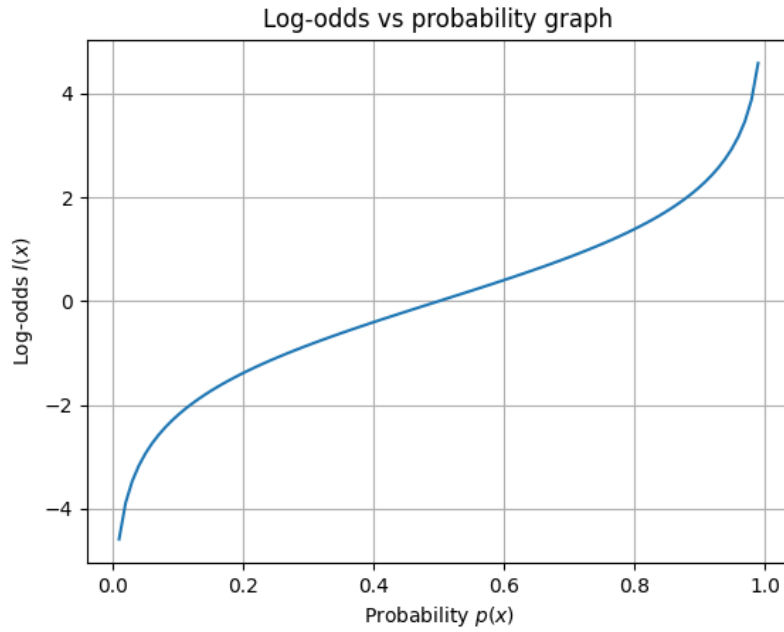
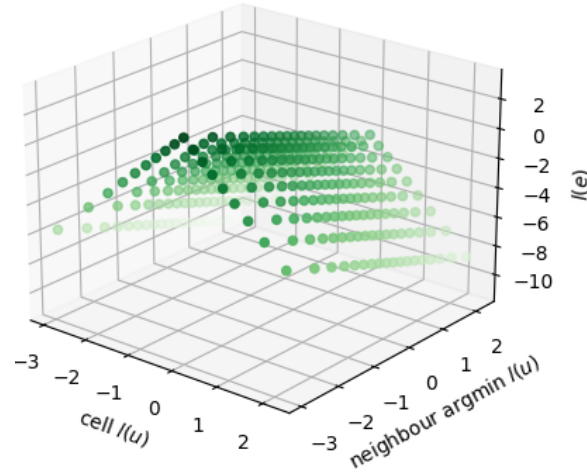


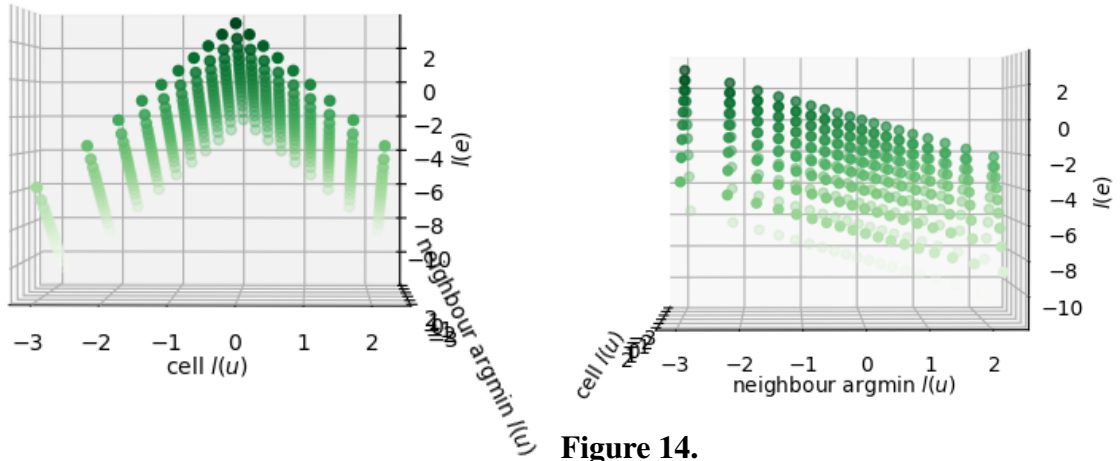
Figure 11.
Log-odds vs probability

By defining a very low and log-odds $l_{LOW} = l(0.05)$, edge detection can be implemented in log-odds space by multiplying the absolute value of the odds at (i, j) with l_{LOW} , and subtracting the minimum odds of all the neighbours, as

$$l(e_{i,j}) = l_{LOW}|l(m_{i,j})| - \underset{k,l \in [-1,1]}{\operatorname{argmin}} l(m_{i+k,j+l}) \quad (3.78)$$

**Figure 12.**

3D scatter plot showing the effect of various cell and neighbour values on the odds of cell being an edge.

**Figure 13.**

Same plot as Fig 12, showing effect of cell uncertainty.

Figure 14.

Same plot as Fig 12, showing effect of neighbour unoccupancy.

Figures 12 to 14 show the same 3D plot combining the possible values of the cell $m_{i,j}$ on the x -axis and the minimum neighbour on the y -axis to obtain an edge probability $l(e_{i,j})$ at the cell from three perspectives. Figure 13 clearly shows how large uncertainty in cell $m_{i,j}$ increases the edge probability, while Figure 13 shows how decreasing values of the minimum neighbour, in other words, an increasing certainty that the neighbour with the lowest occupancy probability is unoccupied, also increases the edge probability. Whereas in probability space the values were combined in multiplication, in log-odds space they are combined with addition, according to the log law $\log ab = \log a + \log b$.

Frontier detection

As the robot seeks to maximise information gain in exploration, detecting edges by themselves are not enough, but should rather be used to detect groups of edges near one another. This is done by contrasting and blurring the map created in edge detection. The map is blurred to remove outlier edges, i.e. edges that are not geometrically close to other edges. By simply iterating over the map and replacing each cell with the sum of it and its surrounding cells in log-odds space, the new value in the cell will be smoothed out. If very few edges are in the vicinity of the cell, the large negative weight of non-edge cells will negate the effect of the edges. On the other hand, if there are multiple edges in close proximity, the probability of a frontier occurrence will increase.

Contrasting is necessary as frontiers will typically occur in straight lines, yet the blurring mechanism blurs the field in two dimensions. Thus even a solid single cell width frontier spanning several cells in length might not prevail against large negative weights in its vicinity. In addition, once the edges have been detected, there really is no use for storing the degree to which a cell is not an edge, because blurring is a summing mechanism an edge probability of $p(e_{i,j}) = -0.3$ and $p(e_{i,j}) = -4$ will affect the result very differently, even though the cells should in theory represent the same thing. Thus, the contrasting filter watersheds the edge map as

$$p(e_{i,j}) = \begin{cases} p_{HIGH} & p(e_{i,j}) \geq p_{REF} \\ p_{LOW} & p(e_{i,j}) < p_{REF} \end{cases} \quad (3.79)$$

where the constants p_{HIGH} , p_{LOW} and p_{REF} are user configurable, but generally $p_{HIGH} > p_{LOW}$, $|p_{HIGH}| \gg |p_{LOW}|$ to bias the contrast toward edges. From simulation results, frontier exploration works best if $p_{REF} \approx 2$. After contrasting and blurring the map, a frontier position is chosen if one exists. If there are multiple frontiers, the first exploration point found can be chosen, or selected at random from the available points.

3.3 Hardware Design & Implementation

3.3.1 Sensors

The autonomous vacuum cleaner has to interact with the environment in an intelligent manner, having knowledge of the landscape as well as its own observed actions. Thus, choosing the sensor components that will give real-time measurement input to the system is an integral first step in the journey towards robotic perception. The required input data aims to update the robot's perception of reality primarily in two ways: input that tells the robot about the environment, and input that observes the robot's actions.

The sensor input that describes the environment has to quantify the landscape by measuring the distance to obstacles in the horizontal plane close to the floor. Ideally the robot should be able to get a sense of the position of a measured obstacle relative to itself, thus

both the distance to and the approximate direction of the obstacle is required. The potential sensor candidates include Lidar sensors, cameras, ultrasonic sensors and infrared scanners.

After thorough research and requirements evaluation it became clear that HC-SR04 ultrasonic sensors are the most suitable candidate for the system. An ultrasonic sensor is a distance sensing device that contains a transmitter and a receiver used to emit and receive an ultrasonic pulse respectively. Transmission of a 40kHz signal pattern starts while an echo pin is simultaneously pulled high until the transmitted pulse is reflected back into the receiver module. Measuring the length of the travelling pulse, while assuming that the speed of sound is known and approximately constant, the distance to the nearest obstacle in the direction of the sensor can easily be determined. The sensor does however have a 30° total angular range or field of view within which the closest object will be detected. This leads to uncertainty in the exact location of the obstacle, as any position in a arc around the sensor's view could produce a reading at the specific distance. Despite this challenge, which will be addressed later on, the ultrasonic sensor is still the best sensor for the job due to the little post-processing required to interpret its measurement data, its large measurement range (2cm - 4m), its ability to operate in low lighting conditions and its relatively low power consumption, which is important for a system that runs on batteries.

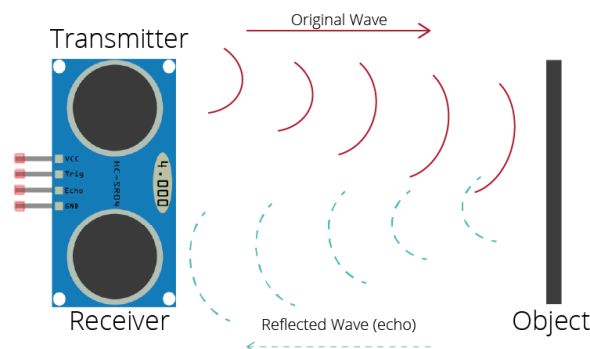


Figure 15.
HC-SR04 ultrasonic wave operation

The system implements three HC-SR04 sensors in a semi-circle around the front of the robot, placed at such an angle that the three sensors collectively scan a 120° field of view directly ahead of the robot, with each sensor placed offset by 40° relative to one another. The 10° blind spot produced by this design is small enough that it is highly unlikely that an obstruction can hide in there, without showing up in measurement readings on either side of the gap, especially across different samples taken at different locations.

A glaring limitation prohibits the use of higher resolution sensor components, such as cameras or lidar, which is the amount of processing power and more importantly, memory, required to effectively analyse incoming data. Computer vision algorithms are bulky and expensive, requiring tens of thousands of matrix operations and several levels of filtering, processing and AI-based sorting to work. These approaches are unfeasible for the simple, low-cost

solution that this project aims to implement. Memory is a particularly constrained resource in an embedded system, with every single byte of stored space important. An array of photos in different processing stages taking up a few megabytes of storages is simply not an option.

The second set of sensors address the potential non-determinism in the robot's expected actions. The robot navigates the world by setting the duty cycle of the motors turning the wheels. This action is quite simple, but prone to process noise. For example, if both motors receive the same PWM duty cycle, the expected outcome is forward propagation in a straight line, but due to differences in friction, magnetic field strength and physical construction, one of the motors inevitable turns a little bit faster than the other one, which slowly takes the robot off course until it finds itself in a completely different location and orientation than its internal map of the world expects it to be.

The solution to actuator uncertainty involves an observation feedback loop coupled with a PI-controller, which will be explored in another section. The observation feedback loop observes properties of the robot that directly or indirectly provides information about the state of the device. These measurements are fused with the robots own estimation of its state to produce an updated state. Five sensor components are chosen for this task, a gyroscope, accelerometer and magnetometer combined in a 9-DOF IMU, as well as two odometers placed on the left and right wheels of the robot. Together these sensors can measure angular rate, velocity, acceleration and magnetic induction in three dimensions, potentially providing more than enough information to correct the robot's estimate of its state.

As the robot only operates on a 2D plane, assuming that the robot will never fall of a cliff (not that accurate positioning will be of much use in that case in anyway), we can neglect measurement data in certain dimensions, such as the roll and pitch of the gyroscope and the gravitational tug in the vertical direction from the accelerometer. This simplifies the sensor fusion algorithms as we focus on the sensor data that have a real impact on the positioning of the robot.

In summary, the gyroscope, accelerometer and magnetometer of the MPU9250 IMU sensor works in conjunction with two light emitting odometers attached to the wheels to accurately estimate the position of the robot in its internal map of the world. Unfortunately the accelerometer has to be placed on an exactly level plane to prohibit gravity from affecting the readings on the horizontal plane. The gyroscope is also infamous for large amounts of bias and bias instability, leading to wildly inaccurate readings. To overcome both of these problems, the robot stays in an initialization state for three seconds upon startup, during which at least 30 IMU sensor measurements are taken. As the robot is assumed to be stationary, any deviation from the expected values can be interpreted as bias. The average of the measured bias is subsequently subtracted from every measurement taken during navigation. This relaxes the requirement for a flat accelerometer and accurate gyroscope, and experimental results prove that this simple bias removal technique vastly improves the accuracy of the measurement model state estimation.

3.3.2 Veroboard

The veroboard design initially started out as a desperate attempt to build a working prototype after several iterations of PCB boards failed to work. As veroboards are a lot easier to change, redo and improve, a design choice was taken to stick to veroboard prototyping. The veroboard is home to a small ecosystem of components, each playing their part in making the system work as a whole.

The most important component on the board is the PIC32MX270F256-B microcontroller from Microchip, boasting a 40 MHz clock, 256kB of flash memory and 64kB of SRAM, as well as the MIPS32 RISC architecture. The chip has 28 pins, of which 17 can be used as general purpose input/output (GPIO) pins. With 14 peripheral devices, some requiring more than one pin, it seemed impossible to reconcile the large peripheral need with the low pin count. Fortunately, through a mixture of iteration, optimization and careful design, creative methods were developed to squeeze as much functionality out of the pins as possible.

For example, originally the three ultrasonic distance sensors each had to be triggered by a separate trigger pin to start a distance reading, while the measurement occurred on another separate echo pin. Thus 6 out of 28 pins were wasted on one aspect of the peripheral components. Then it was discovered that the same echo pin can be reused if the correct assortment of diodes and resistors are used to pull the line low between readings and to eliminate power loss on the shared bus. So now we are down to three trigger pins and one echo pin. Triggering all three pins at once will result in potential crosstalk between the sensors and should be avoided. Finally it was found that triggering a pin and listening to an echo has the same characteristic pulse on a line, thus if the echo pin of sensor 1 is connected to the trigger pin of sensor 2, a completed measurement on sensor 1 will automatically start transmission on sensor 2. This was the final breakthrough that allowed the system to dedicate only 2 pins to all three ultrasonic sensors, while simplifying the code necessary to operate these components.

Another example is the sharing of the I²C bus. The IMU, magnetometer and OLED display screen communicate via I²C. Thus by connecting the three components to the same two wires can be used for SDA and SCL, using I²C addressing to announce the recipient of a message. Lastly, further optimization took place by only dedicating one pin, TX to the UART, as it is assumed that the robot will not receive any data but will only be used to display information.

3.3.3 Robot body

3.3.4 Power management

Part of the design requirements of the robot describes the need to autonomously manage the robot's power consumption, whereby it returns back to the docking station (or starting location) once it detects low battery capacity. The robot can detect remaining capacity by measuring the voltage over the batteries. A single 18650 battery is fully charged at 4.2V, nominal at 3.7V and flat at 3.2V. The system has three 18650 batteries in series, thus it will be fully charged at 12.6V and dead at 9.6V. We would ideally want to halt operation and return back to the charger if the battery falls below 25%, i.e. measured voltage is $< 10.2V$. Of course, measuring voltages around 12V will completely fry a 3.3V chip, whereas simply lowering the voltage in a voltage divider will cause the applicable voltage range to be very small (2.5V - 3.3V).

Instead, the nature of electricity flow in the robot is used. The L298N H-bridge draws power from the batteries at their current varying voltage, but always outputs 5V to the veroboard circuit. By taking the scaled voltage difference between battery voltage and the 5V from the H-bridge in a differential amplifier, the complete range of the battery can be estimated more accurately.

3.3.5 Vacuum pump

3.4 Software Design & Implementation

3.4.1 Data structures

Bot struct

Submaps

Node queues

3.4.2 Program flow

3.4.3 Peripherals

3.5 Simulations

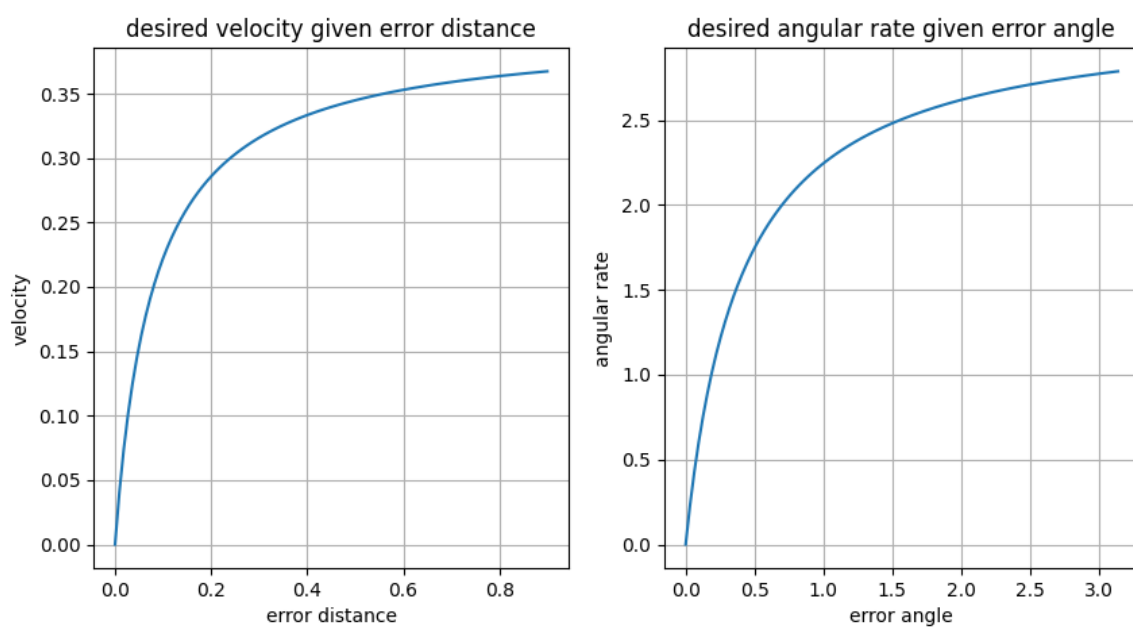


Figure 16.
Desired forward and angular velocity at distance to error

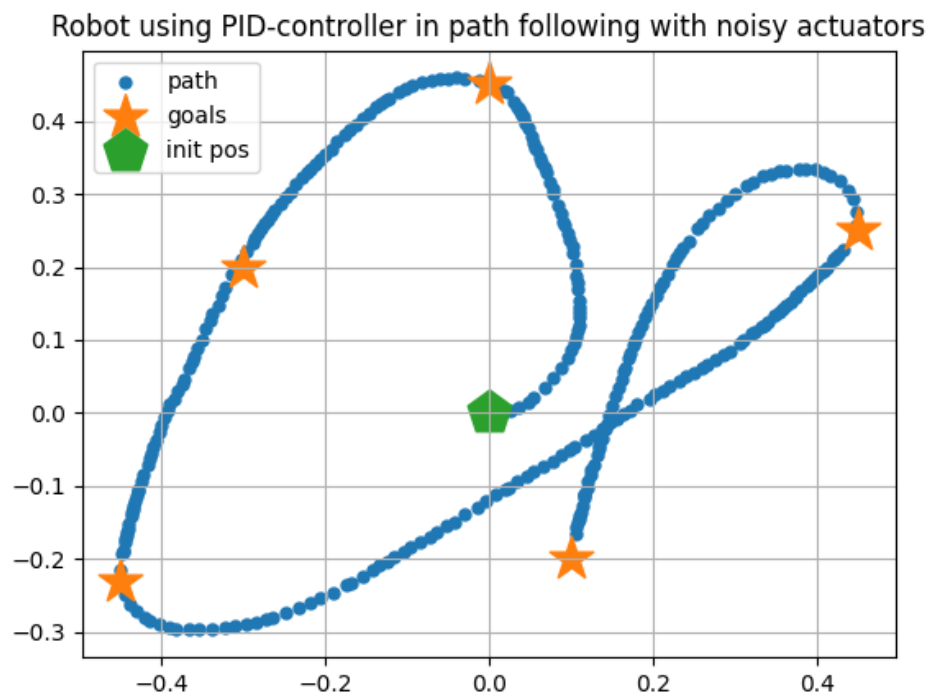


Figure 17.
PID trajectory navigation

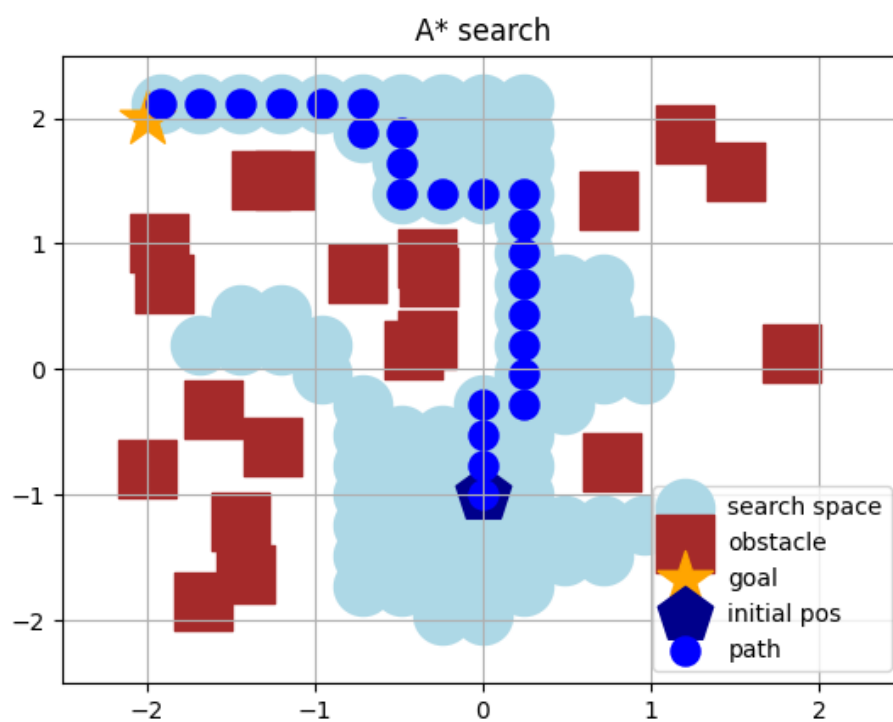


Figure 18.
A* search algorithm simulation, navigating obstacles

3.6 Visual and statistical analysis

4. Results

4.1 Summary of results achieved

Intended outcome	Actual outcome	Location in report
Core mission requirements and specifications		
The robot should cover at least 95% of the mapped floor space while cleaning.		
The robot should map specified area smaller than $20m^2$ with a $10cm^2$ tolerance in two dimensions.		
The robot should not collide with static objects, stopping at least 1cm before the object and going around it.		
The robot should be able to clean three different levels of dirty, which is measured in the amount of rice grains spread over $10cm^2$. The robot should clean at least 90% of the rice without wasting time (continue cleaning when it is already clean).		
The robot should be aware of its power levels and return to the station when the battery falls below 20%. It should dock itself to the charge port before battery falls below 5%.		
After mapping the room the robot should estimate the cleaning time accurately to within 5 minutes, taking in consideration the size of the room, dirt level at samples and obstacles present. It should also display what it is busy doing (cleaning, mapping, returning to dock) visibly for the user.		

The robot should stop at least 1cm before stairs or similar altitude drops, during a cleaning operation it should not exit the room via doorway.		
Field condition requirements and specifications		
The robot should operate indoors on a relatively flat surface.		
The robot should be able to operate in rooms with static obstacles.		

4.2 Qualification tests

...

5. Discussion

5.1 Interpretation of results

5.2 Critical evaluation of the design

5.3 Design ergonomics

5.4 Health, safety and environmental impact

5.5 Social and legal impact of the design

6. Conclusion

6.1 Summary of the work completed

6.2 Summary of the observations and findings

6.3 Contribution

6.4 Future work

7. References

- [1] Z. Yiping, G. Jian, Z. Ruilei, and C. Qingwei, "A srt-based path planning algorithm in unknown complex environment," in *The 26th Chinese Control and Decision Conference (2014 CCDC)*, 2014, pp. 3857–3862.
- [2] W. H. C. Wickramaarachchi, M. A. P. Chamikara, and R. A. C. H. Ratnayake, "Towards implementing efficient autonomous vacuum cleaning systems," in *2017 IEEE International Conference on Industrial and Information Systems (ICIIS)*, 2017, pp. 1–6.
- [3] K. M. Hasan, Abdullah-Al-Nahid, and K. J. Reza, "Path planning algorithm development for autonomous vacuum cleaner robots," in *2014 International Conference on Informatics, Electronics Vision (ICIEV)*, 2014, pp. 1–6.
- [4] K. Sugiura and M. H, "Particle filter-based vs. graph-based: Slam acceleration on low-end fpgas," *arXiv*, 2021.
- [5] Y.-M. Han, J.-B. Jeong, and J.-H. Kim, "Quadtree based path planning for unmanned ground vehicle in unknown environments," in *2012 12th International Conference on Control, Automation and Systems*, 2012, pp. 992–997.
- [6] D. Chen, J. Weng, F. Huang, J. Zhou, Y. Mao, and X. Liu, "Heuristic monte carlo algorithm for unmanned ground vehicles realtime localization and mapping," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 10 642–10 655, 2020.
- [7] C. Ju, Q. Luo, and X. Yan, "Path planning using an improved a-star algorithm," in *2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan)*, 2020, pp. 23–26.
- [8] L. Chen, Y. Shan, W. Tian, B. Li, and D. Cao, "A fast and efficient double-tree rrt-like sampling-based planner applying on mobile robotic systems," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 6, pp. 2568–2578, 2018.
- [9] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, 1997, pp. 146–151.
- [10] G. Oriolo, M. Vendittelli, L. Freda, and G. Troso, "The srt method: randomized strategies for exploration," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 5, 2004, pp. 4688–4694 Vol.5.
- [11] S.-H. Chan, P.-T. Wu, and L.-C. Fu, "Robust 2d indoor localization through laser slam and visual slam fusion," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2018, pp. 1263–1268.
- [12] D. Caruso, J. Engel, and D. Cremers, "Large-scale direct slam for omnidirectional cameras," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 141–148.

- [13] A. Harindranath and M. Arora, “Mems imu sensor orientation algorithms-comparison in a simulation environment,” in *2018 International Conference on Networking, Embedded and Wireless Systems (ICNEWS)*, 2018, pp. 1–6.
- [14] R. Milijaš, J. Oršulić, and S. Bogdan, “When measurements fail: using an interactive slam solution to fight bad odometry,” in *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2020, pp. 1–6.
- [15] W. Burgard, *Introduction to Mobile Robotics*. University of Freiburg, 2012.

Part 4. Appendix: technical documentation

HARDWARE part of the project

Record 1. System block diagram

Record 2. Systems level description of the design

Record 3. Complete circuit diagrams and description

Record 4. Hardware acceptance test procedure

Record 5. User guide

SOFTWARE part of the project

Record 6. Software process flow diagrams

Record 7. Explanation of software modules

Record 8. Complete source code

Complete code has been submitted separately on the AMS.

Record 9. Software acceptance test procedure

Record 10. Software user guide

EXPERIMENTAL DATA

Record 11. Experimental data