Design and Implementation of Unicycle AGVs for Cooperative Control

Roy A. J. Berkeveld
roy@berkeveld.net

August 30, 2016

Master Thesis in partial fullfillment of the degree of
Master of Science in Computer Science and Engineering
at the Eindhoven University of Technology

Supervised by:

dr. Kevin Buchin
Eindhoven University of Technology

dr. Dragan Kostić, MSc
SEGULA Technologies

dr.ir. Johan Smeets
SEGULA Technologies

Technische Universiteit Eindhoven University of Technology

SEGULA TECHNOLOGIES

Abstract

Autonomous mobile robot research is a broad field and new technologies are continuously being developed. Practical evaluation of novel algorithms and techniques tends to be expensive and involved due to the hardware, so research frequently resorts to simulation. This work presents the design of a low-cost robotics platform for the purpose of evaluating cooperative motion control algorithms. Much focus is put on sensor performance to reduce localization error, as this is a leading metric for algorithmic performance. This is aided by specific optimizations that take advantage of full-system integration, which is not possible for partial designs. A prototype of the hardware is built, and a partial software implementation addresses many of the integration problems. The platform uses modern technology such as Ultra-wideband radio (UWB), Lidar and computer vision, at a unit cost of around €500. Environmental awareness is shared among collaborating robots, by data structures that are well-suited for distributed updates. One use case is collaborative Coverage Path Planning under the constraints of position error. For this purpose, optimized strategies for trajectory tracking, online navigation, contour-based coverage path planning and formation control are proposed. And finally, the same software framework is also usable for simulations, and therefore suits many types of comparative analysis.

Dedicated to 刘彩霞*, for her endless support.*

Available from:

`https://roy.sh/+bd2016`

This is the 2nd edition, digital version.
The paperback version is published with ISBN 978-90-825928-0-1.

Contents

Chapter 1

Introduction

Autonomous guided vehicles (AGVs) have seen recent popularity. This work describes a low-cost robotics platform designed for high positioning accuracy that enables practical evaluation of algorithms for coordinating multiple robots. Imagine the following scene: Several low-profile two-wheeled robots (so-called unicycle mobile robots) move around an uncertain area with the common goal of covering this area. They opportunistically follow perfectly aligned formations and seem to be aware of their environment. They work efficiently as they know exactly which parts have already been covered, also by other robots. Attempts to interfere with this system seem futile as the robots reorganize to avoid colliding and substitute for one another.

By now it helps to have a concrete application in mind. Lawnmowing robots have a certain range in which they mow all the grass perfectly. Covering the area once is sufficient, but the whole field has to be covered. A typical approach is for one machine to mow in lanes, turning at the edges of the field. A human operator can easily detect the difference between freshly mowed grass and high grass, but let's just say the robot is simple and does not have such advanced vision. A robot would then be dependent on its positioning accuracy to reduce the amount of overlap between lanes. If positioning accuracy gets too low, it would certainly lose its bearing and skip an area. Furthermore, stochastically it cannot be completely certain that indeed it did succeed in mowing all the grass.

A straightforward solution is to use a larger robot that can mow a wider lane. This results in the procedure being more tolerant of positioning inaccuracy as the overlapping region is a lesser proportion of the lane. But big machines are expensive and will not be able to reach all small areas anymore. And even though it would complete its job fast, after that the machine will be idling as it will run out of grass to mow. An automated system does not benefit of rest the same way a human operator would, so small and slower, and therefore cheaper, is likely better.

One solution that is a commercial success are lawnmowing robots that move about randomly, such as the Robomow [57]. Probabilistically this will eventually lead to the whole field getting covered. But the time to completion explodes in regions that are separated only by small passageways. And in general they waste quite a bit of energy wandering about aimlessly. Some people argue that the random directions in which the lawn is mowed is in fact better for the grass. Others prefer the aesthethic of having lanes.

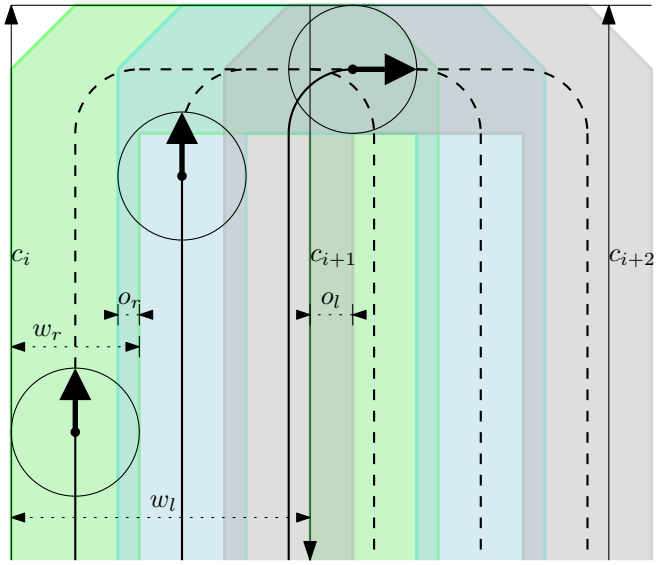

Figure 1.1: Multiple agents covering a region in formation and then reorganizing for the next lane, with minimized overlap and effectively wider lanes than without formation. The details are explained in Section 3.3.5.

So here comes the envisioned solution: small and cheap robots that function autonomously and do not waste time and energy. Arguments have been given for the need for superior environmental awareness and positioning accuracy. However there is one more trick. Even several smaller robots can be significantly cheaper than one big one. And relative positioning is very accurate, so overlap can be reduced when moving together. So when several small robots (agents) form groups, platooning the grass in a formation, a similar range to a big lawnmowing robot at lower cost is achieved. And the formation is optional. It is always possible to break up the formation and task the agents individually when beneficial. For instance when navigating to the next lane as in Figure 3.3.5. Their independence brings more flexibility than a single large robot could provide.

A similar case can be made for the vacuum cleaning of rooms or the clearing of landmines [2]. In the former, significantly more obstacles restrict the freedom of movement. In the latter, even though a lack of complete coverage is potentially lethal, current solutions are not even designed to guarantee coverage. So there are plenty of potential applications that benefit from improved behavior.

The goal of this thesis is to make a step towards achieving this functionality on real hardware. Recently, the abundance of cheap computational units and price reductions for advanced sensors are enabling this type of smart and small robotics. But low-cost fully integrated solutions are hard to find, because the problem involves a surprising number of interesting challenges:

Sensor-wise, there exists no single good sensor that gives acceptable absolute position and orientation for autonomous robotic control. Cues from the environment contribute to our understanding of the world, but none are perfect. With absolute positioning such as GPS, 3 m outdoor accuracy is considered good. Indoor positioning is limited to computationally hungry artificial and still immature vision algorithms. Ranging sensors such as ultrasonic and infrared

suffer from a lot of noise and correlating their output to a world model is difficult. Inertial sensors offer very high measurement rates suitable for robotic control, but they suffer from significant drift as the measurement error adds up. The choice of sensors plays an important role in the performance of a robot, especially related to the computational requirement of processing all the signals. A very challenging problem is the integration of all sensors into one single frame of reference. Such *sensor fusion* requires careful design and a considerable amount of tuning.

Cooperation in distributed systems depends on efficient and robust communication. Exchanging information is possible as long as all robots share a common world reference. The choice of this world must support and optimize the goal that robots intend to achieve. But a high frequency of data and unreliable transmissions can easily saturate the communication channel. And then there is a maximum latency before the added value of environmental information becomes useless. The intricate balance between these, and the process of dealing with conflicting data is key to robust performance.

Algorithmically, a lot of investigation is done in navigation and ways to cover an area with robots. However these have been investigated under mostly ideal circumstances. The hard problems of position uncertainty, collision avoidance, multiple agents, decision convergence and unreliable communication are treated at best individually. The real challenge is to come up with a solution that works under all these circumstances. Of course some assumptions will have to be made to keep our world simple, avoiding the truly unsolved and tricky areas.

All these challenges need solutions when developing low-cost robots for the purpose of area coverage. The next section lists the global requirements of such a system.

1.1 Requirements

When the system is observed as a whole, optimization goes toward performance and cost. Important performance factors are area coverage, completion time, and energy efficiency. Coverage can be expressed as a lower-bounded percentage, but this would complicate things as no area is ever covered with complete confidence. So position accuracy is preferred over the coverage percentage as it is situation-independent and can be measured by a single error bound.

All three metrics are strongly dependent on position accuracy because of the nature of the planning algorithms. Energy efficiency and completion time are of course closely related. But these are not always in agreement with each other.

In terms of cost, the project budget is quite limited and a key requirement is the ability to perform experimental validation with a functional robot. The specific application is not important for our goal, as long as indoor operation can be demonstrated. Though it would be nice if the activity is a practical application for demonstration purposes.

An early requirement always has been to build several robots to evaluate cooperative control. Even though the project budget does not facilitate building more than one, the design will continue to assume that several are available for the theory of cooperation. Hence, the title of this thesis. The limited time being available for this project is motivation to reduce the

required implementation effort. Preferably by purchasing the combination of parts that solves the most time-consuming problems while still achieving novel performance.

This platform will also serve to evaluate several technologies that only recently have become affordable. With this set of requirements there is still a rather large freedom in choice. More detailed requirements will appear as the problem is decomposed and the available technologies are evaluated.

1.2 Research goal and objectives

The main goal is to realize a low-cost robotics platform for the purpose of evaluating cooperative motion control algorithms, specifically area coverage. Several subproblems can be identified within this rather general engineering problem. The solution for each of these fulfills a research goal.

1. Hardware design of the robot platform.
   An evaluation is made for which hardware offers the best trade-off between cost, environmental awareness, robustness, latency and computational overhead. This leads to a selection of hardware for a single robot with total cost of approximately €500.

2. Controller design for sensor fusion and trajectory tracking.
   The low-level controller takes all sensor values and does actuation of the platform. By fusing sensor data and careful tuning, localization accuracy better than any individual sensor can be achieved.

3. Scheduling and design for system performance.
   Many aspects govern the design of a high-performant system with the restrictions imposed by the choice of hardware. Designed is the precise configuration that optimizes latency, data bandwidth and decision capabilities of the robot.

4. Cooperative data structures and analysis of communication.
   Sharing of knowledge about the environment can help the robots make better decisions and increase performance. The essential information to enable cooperation has to be exchanged at certain data rates and detail. For a given number of robots at a typical network, investigated is which type of data is beneficial.

5. Localization in the common world frame.
   Absolute positioning in the world requires evaluating time-consuming algorithms that are suboptimal for real-time control. Performance characteristics of ultra-wideband radio (UWB) are evaluated, which is a technology that has only recently become affordable. It is investigated whether intermediate results of the trilateration calculation can improve localization accuracy and latency.

6. Behavioral algorithms for navigation and area coverage.
   Once a robot achieves environmental awareness, a high-level abstraction can simplify algorithmic interaction. Such an abstraction is proposed in combination with a novel approach for the real-world variant of the Coverage Path Planning problem. Characteristics of the cooperative and thus distributed version of this algorithm are also investigated.

1.3 Outline of the thesis

This thesis is about the realization of a complete platform. Its structure follows the typical phases of development, which are reflected in the following chapters:

Chapter 2: Preliminaries and related work

Background relevant for the research can be found here. The many problems that are addressed are identified and relevant existing solutions are presented and evaluated for this application. In addition, a categorization of available sensor techniques is presented. This overview can serve as a parts catalogue for the development of other low-cost robotic platforms.

Chapter 3: Design

The first steps of the design are a subset of sensors, and the selection of associated hardware, to satisfy the requirements from Section 1.1. Design of the controller is an essential consequence of this selection and will be outlined here. Next, the aspect of cooperation is considered. The necessary shared knowledge and data structures are described in a preliminary software architecture. And finally, a design of the behavioral algorithms is proposed. These include navigation and coverage path planning.

Chapter 4: Implementation

This chapter contains the details and decisions that are not revealed until an attempt at realization is made. All the steps necessary to actually make the system operational are here. A significant part of this is the reasoning behind choices in the implemented code. Of particular mention are the scheduling and performance aspects, which could not be decided before the interfaces are realized. Several components have been shown to require workarounds to get good performance. A remarkable one for this is the system used for absolute localization in the world, before stable operation is achieved.

Chapter 5: Conclusions

Finally, all prior findings are presented in a concise format. This work closes by suggesting potential avenues of future work.

Chapter 2

Preliminaries and related work

A mobile robot contains a feedback system acting on the robot's state of motion in a stationary inertial frame of reference. Unicycle mobile robots are underactuated in the sense that they cannot move sideways, they can only move forward and rotate. To reduce model complexity, it is assumed all movement is at constant height and rotation occurs only around the orthogonal axis. The pose of a robot consists of a singular orientation θ and a position (x, y) relative to a fixed reference $\mathcal{O}$. The pose $p = (x, y, \theta)$ is a point in the three-dimensional configuration space $C = \mathbb{R}^2 \times [0..2\pi)$, with orientation modulo 2π. C is subdivided into free space C_f and obstacle region C_o s.t. $C_f \cup C_o = C$ and $C_f \cap C_o = \emptyset$. Obstacles in the space are essentially padded in C_o with the distance r_θ from the far point on the opposite side of the robot's body to its center of rotation (in the direction $\pi + \theta$), to stop the robot from colliding with the object.

Note that unicycle mobile robots in general are limited to movement along their orientation θ and cannot always move freely about the configuration space. That is, they do not satisfy the property of controllability due to a nonholonomic constraint. As a solution it is assumed that all projections of C_f on x, y are identical, as is the case with a circular robot of radius r with central pivot point when all $r_\theta = r$. This essentially reduces motion in C_f to two dimensions, as turning in place would have no effect on obstacles. By extension, any continuous path in C_f becomes controllable.

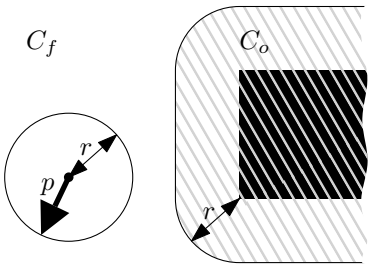

Figure 2.1: The simplified configuration space C with illustrated pose p near an obstacle.

Motion is defined as a path through C_f. If there is a continuous path $f : [a, b] \to C_f$, where $a < b$, $f(a) = p_a$ and $f(b) = p_b$, then the robot can physically reach p_b from p_a. Motion planning is the creation of these paths to achieve certain goals, and will be treated in Section 2.2. Among such goals is navigation and obstacle avoidance, exploration, following formations, and area coverage. One role of motion planning is to ensure the path remains in C_f, even when it changes due to dynamic obstacles. Unlike paths, trajectories incorporate the robot's dynamics by mapping poses to time. This allows interaction with dynamic obstacles and can ensure that the robot stays within its physical capabilities such as maximum velocity. Then trajectory $t : [a, b] \to \mathbb{R}$ is continuous and monotonically increasing, where $t(i)$ is the time at which the robot reaches the pose $f(i)$ for $a \leqslant i \leqslant b$.

Localization provides an estimate for the robot's state of motion. One such estimate is p itself, which contains position and orientation. The controller performance is directly affected by the accuracy of its state, so typically the first and second derivatives are also estimated, which are velocity $\dot{p} = (\dot{x}, \dot{y}, \dot{\theta})$ and acceleration $\ddot{p} = (\ddot{x}, \ddot{y}, \ddot{\theta})$. Especially the position relative to $\mathcal{O}$ is difficult to determine accurately. The background and several techniques for performing localization are described in Section 2.1. The detailed configuration of C_f can be deduced by relating various environmental measurements to p by exploration and mapping. This technique is widely known as Simultaneous Localization and Mapping (SLAM) [5]. As knowledge of C_f is necessary for successful path planning, this will be treated in Section 2.1.4, and later Section 3.2.2.1.

In research of motion planning and high-level robot behavior, an accurate state of motion is often assumed. This is especially the case with coverage path planning, where typical applications such as machine milling [4] do not have a feedback system. The reverse is true for research on localization [64], where objects are commonly assumed to be static, especially for the duration of a measurement. To simultaneously compensate for delay and dynamics, it is necessary to know both the true measurement latency and the state of motion of the system. Systems that incorporate both have the potential to perform better than any system that attempts to solve localization and motion planning independent of one another. Section 2.2 will go into detail on various aspects and methods of motion planning.

Other aspects that significantly increase the difficulty of the problem are multi-agent systems and inter-robot communication. In essence, communicating accurate state among robots is desirable as better environmental awareness permits more informed decisions. However this communication adds additional latency and the rate of measurements becomes stochastically uncertain. This work will also address the difficulty of designing such a distributed system of measurements in Section 4.3.1.

Several similar integration projects [52] have preceded this attempt, with different subsets of technology. In Section 2.3 a summary of their work and the relation to this thesis is presented.

2.1 Localization

Localization is the means of estimating the robot's own pose p, at least in part. p consists of the position and orientation of a system, usually of itself relative to $\mathcal{O}$. Methods to perform localization strongly depend on the available sensor types. The characteristics of these systems classify them in three categories:

1. Dead reckoning: systems that do integration over time of purely local sensors.
   Such measurements can happen at relatively high rates, but suffer from drift.

2. Unique landmarks: direct measurement of references such as visual codes and beacons.
   Such systems are most certain of the results, capable of giving error estimates. However they depend on the presence of calibrated and costly specialized infrastructure.

3. Recognition: using maps of environment measurements for matching with heuristics.
   These can give superior results under the right conditions, but as they use heuristics

they are less reliable. With these methods false positives and confidence are typically hard to determine.

A good solution seems to involve the best of several of these methods. So each category will get proper attention in the next sections. In addition, the sensors used for recognition can typically also be used to detect obstacles and guide motion planning, and is treated in Section 2.1.4. The details behind combining these different sensors are presented in Section 2.1.5.

2.1.1 Dead reckoning

Also known as ded (or deduced) reckoning, dead reckoning integrates relative measurements over time to advance a prior known p. Note that the sensor measurements are all relative to a prior state, and this past state may be undefined. Measured are typically relative distance, angular velocity and acceleration, by means of wheel encoders, gyroscopes, and accelerometers, respectively. Such measurements can be made frequently and with high accuracy, as making them depends only on the measurement system itself.

Accuracy increases as these updates are made more often, up to the limits of numerical stability. Sensors are not perfect and their error consists of two parts, noise and bias. Noise is inherent to all measurements and can be reduced by adding a low-pass filter and, up to a point, by increasing the sample rate. Drift occurs when the sensors are biased, when they have a non-zero mean. The incremental error caused by bias makes any system that depends solely on dead reckoning diverge in state.

The computation of change in position from a relative movement depends on the heading angle. Orientation drift is the most significant contribution to position error. Small errors in θ quickly make x and y diverge over time, known as Abbe error [75]. As the error in the angle keeps growing, future position updates become less accurate. This makes a dead reckoning system unreliable for long-term absolute positioning, such as with mapping and pathfinding.

The accelerometer also measures the component of gravity, which is an absolute orientation reference and always points down. Unfortunately this direction is orthogonal to the plane of navigation so it can not be used to determine the heading angle. In fact, with slight misalignment the gravity component will leak into the x and y components in the form of bias. Accelerometer bias is a serious problem [24] and compensation is necessary.

Using any of the sensors individually has obvious drawbacks. Especially with second-order derivatives like accelerometers as it is not possible to determine when the system is stationary. The trick is combining them in a smart way to cancel out the problems each experiences. They are summarized in Table 2.1. A wheel encoder offers a lower-order measurement of the pose than an accelerometer. Encoders provide a zero reference and solve at least the velocity drift problem. An accelerometer can compensate for the slipping of the wheels by detecting true relativistic motion. The zero reference can sometimes be determined solely from an accelerometer by a lack of vibration that is typical when the vehicle is experiencing motion. The methods by which these sensors can be combined are discussed in Section 2.1.5.

Position encoders are realized with various techniques [51]. For wheeled robots rotation of the wheels causes a translation in position, so rotary encoders on the driving motors capture this translation. Versions for rotary encoders exist as absolute- and incremental encoders. For

Sensor	unit	sample rate	Major error
Wheel encoder	m	events	slipping
Accelerometer	m/s^2	1 kHz	integration
Gyroscope	rad/s	1 kHz	integration

Table 2.1: Overview of sensors for use with Dead Reckoning

wheeled robots the absolute orientation of individual wheels is unnecessary, all that matters is incremental distance. Quadrature encoding ensures that a difference between clockwise and counterclockwise rotation is measured. For these reasons, wheel rotation is commonly measured with quadrature incremental rotary encoders. The implementation varies as this is usually integrated with the driving motors, which have very different specifications and design depending on the application. All encoder output signals result in pulses that are captured by a discrete counter on a microcontroller. Each pulse indicates a fixed step size change in angular rotation of the wheel, and thus distance, known as encoder resolution. The time difference between two pulses of the same direction determine the average wheel velocity.

From an implementation perspective, accelerometers and gyroscopes are rather similar devices. Modern affordable versions of these sensors use micro-electromechanical systems technology (MEMS), and are sold as complete integrated circuits with a digital interface. Often, three one-dimensional sensors of both types are placed in orthogonal orientation to capture full 6-degree inertial movement. An overview of the commercial MEMS sensor market is given in [7]. This incidentally allows readily available silicon to perform the signal processing that would usually be handled by a separate microcontroller. Such signal processing tends to include sampling at configured rates, analog-to-digital conversion within configured range, digital output buffering and even filtering. All this is intended to offload a central processor from the requirement to do real-time processing itself. A disadvantage is that most of these features increase latency. The choice and configuration of the sensors is essential for performance. Two such highly configurable sensors are the InvenSense MPU6050 and the Bosch BNO055. Each includes both a 3-axis accelerometer and 3-axis gyroscope, and has an I2C interface. These receive specific mention as they also incorporate advanced filtering (discussed in Section 2.1.5), and they have been used in related work before [6, 37].

2.1.2 Unique landmarks

Sensing with landmarks has been widely studied [26, 18]. However when landmarks look alike, there is the possibility of ambiguity. Landmarks that can be identified uniquely do not suffer from ambiguity, which can be done through identification or some recognized coding. These type of landmarks are usually artificial.

In terms of natural landmarks, one can think of the Earth's magnetic poles. As the north- and south-pole are unique, it is possible to determine absolute orientation on the sphere. A 3-axis magnetometer can be used to extract the Earth magnetic field component, which gives an absolute orientation. But because of environmental interference filtering is necessary, which reduces the measurement rate and increases latency.

Well-studied are visual codes [63]. These are implemented with camera systems and depend

on line-of-sight and good environmental conditions. Accuracy depends on the resolution of the vision system and how well it is calibrated. It is common for the robots themselves to carry such a code, and an external calibrated environmental vision system communicates positions to the robots. Image processing is still considered expensive in terms of computational requirements, and having a shared static calibrated system simplifies this. The alternative is where each robot carries its own camera and image processing systems to perform pose estimation, where relative position can be determined by analyzing the perspective (size and orientation) which codes are seen under. When pose estimation is applied to a landmark of known position, the robot's own pose can be determined. Each camera still needs calibration, but modern marker systems such as Aruco [22] facilitate such a setup. All errors in this system can be quantified and given a worst-case bound, such that when a pose estimate succeeds the maximal error is known with high certainty.

Another type of unique landmarks are devices that act as radio beacons [26], such as Wi-Fi access points or global navigation satellite systems (GNSS) such as GPS. Digital protocols typically embed unique identifiers, and wireless communication penetrates objects to some degree. Depending on the physics of the system, several techniques can be used to determine location information from these signals.

- **RSS: Relative Signal Strength** – Signal strength decreases over distance per the inverse square law, which can be used to determine relative position. However signal strength is heavily susceptible to environmental interference and line-of-sight. They also suffer from multipath interference due to reflections, which disturbs the direct signal as it is arriving.
- **AoA: Angle of Arrival** – Using several directional antennae creates a measured difference in signal strength from which the angle of the signal can be derived. This is a directional technique and can be combined with techniques for distance measurement.
- **ToA: Time of Arrival** – When transmissions include a highly accurate timestamp, the differences between transmission and arrival time can be determined. Transmissions occur at the speed of light at a rate of $3.3 \times 10^{-9}\,\mathrm{s\,m^{-1}}$, requiring very accurate clocks and synchronization to tell the difference between signals. This is the basis for GNSS technology such as GPS. It is essential that clocks between devices are synchronized with a base station.
- **TDoA: Time Difference of Arrival** – Whereas ToA uses the time difference between transmitter and receiver directly, TDoA uses the difference between arrival times of pairs of transmissions from several fixed transmitters. The problem of positioning changes from finding intersections of circles to intersections of hyperbolas.

Recent work [52, 61] has been with using radio-frequency identification (RFID) tags for indoor triangulation using RSS and AoA. The accuracy of such a system depends on various properties such as tag density, environmental interference, and antenna directionality. Practical setups exhibit high degrees of noise, which limits positioning in typical indoor applications to several meters, similar to that of other RSS-based techniques such as those using Wi-Fi. Only with optimized setups under good circumstances, accuracies of down to 0.1 m have been seen [52].

Techniques that use time differences for radio signals (ToA and TDoA) require precise timing due to the speed of transmission. This involves a design with a specific focus on localization, such as is the case with ultra-wideband radio (UWB) or GNSS such as GPS. Accuracy in direct line-of-sight scenarios is potentially as high as the time resolution of a receiving clock. GNSS satellites are far away and their signal-to-noise ratio is very low, so receivers have very sensitive antennae. The consequence of which is that with any obstruction receivers may lose the signal, which makes them unsuitable for indoor use. Another problem is signal reflection, which creates multiple paths to the receiver, similar to an echo. Such multipath interference can overlap with the original signal, or the original signal may be weaker than the indirect signal. For time difference techniques, interference is a significant cause of measurement error, and can be approximated by a normal distribution due to the entropy in radio signals.

For GPS satellites, the typical outdoor resolution is 3-10 m. Galileo is a European GNSS that supports up to 0.01 m accuracy for paying commercial users, and 1 m accuracy for public use. By using techniques such as real-time kinematic GPS (RTK-GPS), GPS performance can be improved to 0.1 m which is sufficient for the proposed application. With RTK-GPS, a carefully tuned stationary basestation performs extensive signal correlation and corrects for atmospheric interference. If at any point in time the GPS signal is interrupted, the RTK-GPS module loses its correlation coefficients and they will have to be recomputed, which takes several minutes. Solutions for RTK-GPS are often used for land surveying equipment and are expensive. Recently, affordable RTK basestations such as the $235 Reach RTK by Emlid[1], the $500 Swiftnav Piksi[2], and the $50 Navspark NS-HP[3] have become available. Such corrections are communicated wirelessly over protocols such as RTCM-3 to nearby receivers that support them. But any GNSS solution is not suitable for indoor use due to their low signal strength, and the corrections from RTK-GPS become invalid in such a situation. This technology however would be ideal for outdoor applications such as lawnmowing and demining.

Ultra-wideband radio (UWB) uses TDoA where anchors that are fixed to the inside of a building replace the function of satellites of GNSS systems. Their advantage is the use of one wide channel for signal transmission. Obstacles reflect only narrow frequency bands, and by using a wider band, multipath interference is reduced. Accuracies of 0.1 m can be achieved, but this depends on the specifics of the implementation. The high proximity of the anchors to the receiver gives stronger signals, which enables bidirectional communication necessary for clock synchronization.

To summarize, interesting systems for the proposed high-accuracy indoor application are UWB radio and computer vision with pose estimation of unique codes. GNSS solutions like RTK-GPS are very promising in general but are not suited to this demo application due to their outdoor line-of-sight requirement. Each has the potential for the necessary high accuracy and is likely to stay relevant in the future. And their properties are sufficiently non-overlapping such that each contributing uniquely toward environmental awareness.

[1]Reach RTK by Emlid, `https://emlid.com/shop/reach-rtk-kit/`
[2]Piksi by Swiftnav, `https://www.swiftnav.com/piksi.html`
[3]NS-HP by Navspark, `http://navspark.mybigcommerce.com/ns-hp-rtk-capable-gps-gnss-receiver/`

2.1.2.1 Ultra-wideband radio

UWB radio can be used for accurate and reliable measurements of distance. The technology consists of a transceiver with a highly accurate clock that achieves 10^{-11} s temporal resolution of received messages. At the speed of light, this translates to a theoretical distance resolution of 3 mm. Error sources include clock jitter, clock desynchronization, calibration errors, direct interference and multipath interference, and results in the reduced accuracies that are advertised. Of these, clock jitter and direct interference result in noise with a Gaussian distribution. The other three result in an environment-depending bias, but can be compensated for with proper techniques. Upper bounds for clock desynchronization can be determined and calibration errors can be minimized by verification. UWB radio can use low-frequency carrier signals, which are better suited to travelling through obstacles. By using a wide frequency band, the direct signal tends to be the strongest, which reduces the effects of multipath interference significantly. The infrastructure consists of several active beacons, called anchors, that are fixed in the environment at known positions. The anchors surround an area within which the position of UWB tags is to be determined. Both anchors and tags are typically active transceivers, capable of both sending and receiving. For performing accurate localization, all anchors need to have their clocks synchronized to within picoseconds of each other. Clock drift caused by temperature oscilations requires timely synchronization, and interference make it difficult to communicate reliably.

Once range measurements can be reliably made, multilateration of tags becomes possible. A real-time location system (RTLS) is built on top of this ranging capability to provide such localization. Usually this involves known locations of the anchors. RTLS may be centralized, in that one central system governs when measurements of individual tags occur and collects their locations by requesting a response. In such a system, the tags themselves may not be aware of their own location as the multilateration computation is done elsewhere. Decentralized RTLS also exist, where each tag performs its own multilateration on observed or requested range measurements. A decentralized system may see the anchors as available infrastructure and use this opportunistically, and does not require any hardware beyond the anchors alone. In this case, the tag needs to know the network configuration. And clock synchronization between anchors needs to be taken care of externally. Also whereas in a centralized system, the central controller can easily avoid congestion by following a perfect schedule, in a decentralized RTLS this may not be possible. Packets may be sent at conflicting times and congestion may occur, making decentralized localization in a cooperative system potentially problematic. Evidently, centralized RTLS systems are more popular for industrial use due to their reliability and predictability.

In [73] an overview is given for several methods, and specific technologies implementing UWB radio. There are several manufacturers of UWB radio equipment. But not many offer an open ecosystem suitable for innovative solutions. The following companies offer UWB solutions:

- **Decawave** has designed an integrated UWB transceiver IC and sells this technology to UWB integrators. It is capable of 0.1 m accuracy of localization with a line-of-sight range of 290 m. Indoors, accuracy is reduced to 0.3 m and the typical range is reduced to 25 m. The DWM1000[4] sold for $25 is a complete module containing the DW1000 IC and antenna, which is well-suited for integration in embedded platforms. This component is

[4]Decawave DWM1000 UWB module, `http://www.decawave.com/products/dwm1000-module`

the necessary ingredient for realizing anchors, tags, and command interfaces. Decawave itself does not sell solutions for RTLS, but leaves this up to third party implementors.

- **Nanotron** produces an 80MHz RTLS localization system[5] with 1 m outdoor resolution, reduced to 3 m resolution indoors. The range can be tuned to various needs by signal amplifiers, and the low frequency is good at penetrating objects. This is a centralized system and tags are unable to deduce their own location independently. An academic test kit is available for €3848.
- **Time Domain** produces a 2.2GHz RTLS solution[6] with 352 m range and 0.02 m resolution outdoors. Resolution is reduced to 0.1 m indoors, and tags can independently perform localization in a decentralized fashion. Development kits costs $10000.
- **Ubisense** has a complete RTLS solution[7] at 6 GHz to 8 GHz frequencies with up to 30 Hz update rates and 0.15 m outdoor resolution. It uses both AoA and TDoA techniques to have more information for performing localization. The high update rate is realized by internal filtering and does not reflect the number of TDoA measurements. An academic research kit costs $12500.
- **Zebra** offers the Dart UWB RTLS system[8] at 6.25 GHz to 6.75 GHz frequencies with an outdoor range of 200 m and 0.3 m precision. Their demo kits sell for $12000.

The Time Domain system meets nearly all criteria for performing distributed localization, except the cost is prohibitive. The Nanotron system has such poor performance that integrating it with the real measurements would be prohibitive. The Ubisense and Zebra systems are more expensive and have worse performance than the Time Domain system. Most companies intend to sell the technology as complete solutions and typically require additional license cost for running RTLS software. They do not seem to provide the raw range measurements which will be useful for integration with our sensors, but rather provide derived measurements that are already extensively filtered. So only Decawave offers non-restrictive modules with favorable characteristics. Decawave has produced the DW1000 integrated circuit and the DWM1000 module containing it. This module is available by itself, but the whole RTLS infrastructure would have to be built on top of it, which is quite involved. Several solutions exist that already take care of this using the DW1000 IC:

- **Ciholas DWUSB**[9] is a centralized system for performing RTLS at 20 Hz rates and with 50 m line-of-sight range. An additional restriction is that the master must be able to see every anchor to be able to successfully perform clock synchronization. The included code is free of license cost, but is quite limited and similar to an academic demo, not a full product. A minimum of six anchors is recommended, and in addition one tag and master node for the central server are needed. At a cost of $179 each, a system without the required server costs $1432.

[5] Nanotron RTLS, `http://nanotron.com/EN/PR_find.php`
[6] Nanotron Rangenet, `http://www.timedomain.com/rangenet/`
[7] Ubisense Dimension4, `http://ubisense.net/en/products/Dimension4`
[8] Zebra Dart UWB,
`https://www.zebra.com/us/en/solutions/location-solutions/enabling-technologies/dart-uwb.html`
[9] Ciholas DWUSB, `https://products.ciholas.com/products/dwusb`

- **Decawave's TREK1000**[10] is a evaluation kit for the DWM1000 modules from Decawave. It is far from an RTLS solution in that clock synchronization is unsolved and they are only useful for ranging between two modules. The TREK1000 which can only perform a 2-module ranging demo costs $924.
- **OpenRTLS**[11] has many products derived from the DW1000 IC. Anchors are relatively expensive at €350, while tags are cheap at €43. The RTLS service needs licensing, and a starterset software development kit is available at €9000. For a custom implementation of localization algorithms a tag development license has to be purchased at €3000. Solutions of 0.1 m accuracy are advertised.
- **Pozyx**[12] offers a decentralized system for localization, where tags are in control and can organize localization independently. Tags and anchors can be configured for either function and perform the necessary clock synchronization at the time of localization. Tags have an additional IMU to determine orientation. Anchors have to be powered externally but will communicate over UWB radio frequencies. Tags and anchors cost €135 each and come without further license cost with an open source platform. The developer kit that can perform localization costs €1000. Tags are interfaced as Arduino shields and have a well-documented API and open-source library, which should facilitate development.
- **Sewio RTLS**[13] is a simplistic RTLS solution without automatic anchor configuration and a focus on the end-user. Anchors have to be connected through a wired network. Their RTLS Server product uses centralized localization of which not much detail is available. A virtual machine needs to run on a server and licenses are required, which puts the system out of the scope for an embedded application. The RTLS TDoA kit costs €4250 and includes licenses, but excludes the wired infrastructure cost.

Line-of-sight range of the Ciholas DWUSB platform is so low that any wall will likely cause clock synchronization to fail completely. The Trek1000 from Decawave lacks too many features to be suitable for an implementation of RTLS to succeed in time. Licensing costs of OpenRTLS are prohibitive while their hardware is quite attractive. The architecture of Sewio RTLS server signals toward high installation costs and is not suitable for embedded use in this application. In contrast, the decentralized nature of Pozyx seems like a good fit for this application, and several problems such as clock synchronization have already been solved. In conclusion, only Pozyx offers an affordable solution for UWB that is well-suited to the application.

2.1.3 Recognition based on the environment

The third category of sensors concerns any type of external environmental detection that does not uniquely identify landmarks. These sensors typically build up partial knowledge of the local environment. If the external situation is carefully designed it is possible to use these sensors to determine the system state. But it is likely not to be the case due to obstructions and general uncertainty as to what the measured values truly represent.

[10] Decawave TREK1000 evaluation kit, `http://www.decawave.com/products/trek1000`
[11] OpenRTLS real time location systems, `https://openrtls.com/page/rtls`
[12] Pozyx centimeter positioning, `https://www.pozyx.io/`
[13] Sewio RTLS-TDoA, `http://www.sewio.net/rtls-tdoa/`

- **Camera systems** – The information embedded in camera images is very rich and is researched extensively in the field of computer vision. Relevant computer vision problems for our application include 3D object segmentation, recognition and tracking [18]. Most of these have high computational demands and are unsuitable for embedded platforms. But with certain artificial techniques or accelerated processing using specialized hardware, some methods may be feasible. One such example is pose estimation of Aruco markers [22], which has a robust and efficient approach to detecting unique 2D codes.
- **Contact sensing** – When the robot touches something nearby, a switch is physically triggered. This is suitable in the sense of an emergency stop, when all other sensing methods fail. Some robots like the iRobot Roomba[14] vacuum cleaning robot use this as their primary navigation aid.
- **Infrared ranging** – Reflecting infrared light off of an object and then detecting the angle of reflection on a linear CCD gives distance. But in general this technique suffers from interference due to reflections and different material properties. So it is not very reliable.
- **Sonar** uses ranging with sound by producing ultrasonic pulses and detecting returned echo. These sound waves travel at $330\,\mathrm{m\,s^{-1}}$. The time between sending a pulse and sensing its multiple echos determines object distances and due to the slow propagation speeds is easily implemented. But reflection of sound signals on objects is very common and interference is a significant problem. So the microphone is typically directional to minimize interference from nearby surfaces. Even then, distance measurements are unreliable.
- **Lidar** is short for Light Detection and Ranging. It uses ranging with infrared lasers by sending modulated pulses. The principle is to catch the reflection of a sent pulse and measure the time in between. This requires highly accurate timing to get good resolution, nevertheless sub-millimeter resolutions are typical. Use of signature correlation reduces interference by high-intensity light sources and other Lidar units. The technology is very robust and has only recently become affordable [34].
- **Radar** uses directed electromagnetic pulses in the same way Sonar and Lidar do. The advantage of EM pulses are that they can travel through some objects and reflect better off of electrically conductive surfaces. The range of Radar is typically much higher than Sonar or Lidar. The wavelength determines how big the object to be detected is, typically radar is suitable for large objects on long distances.

Of these sensing methods, contact sensing is very certain, but has poor range which makes it best suitable for minimizing the effect of collisions. Computer vision with cameras has great potential but also has heavy computational requirements, which is unlikely to be available in embedded platforms. Infrared and sonar are cheap, but suffer from poor reliability and are best avoided unless significant time is spent on filtering the results. Lidar and radar appear to be very robust technologies, but radar is commercially not available. Lidar is increasingly applied in the automotive industry, which is a clear signal that the technology is mature enough for adoption. Lidar has seen recent fully-integrated developments [34] that make it affordable for low cost robotics applications.

[14] iRobot Roomba, `https://www.irobot.com/roomba`

2.1.3.1 Lidar: light detection and ranging

Lidar is already a very robust technology for obstacle sensing in 3D. Its maturity and suitability for autonomous guided vehicles is obvious as the automotive industry is beginning to adopt the technology for self-driving cars. There are several ways in which Lidar can be implemented, with different resulting characteristics. This variation can be seen by its use from height mapping by satellite systems, to 3D object scanning by surface reconstruction of point clouds.

First a breakdown of how Lidar is performed. Robust Lidar methods transmit a modulated beam of light with a laser, of which the reflection is received by a photodetector nanoseconds later. The signal that this photodetector receives contains a pattern that corresponds to the modulation of the laser that was transmitted. This method is called coherent detection and is in contrast with incoherent detection, which simply detects the amplitude of a pulse of energy. Coherent detection increases robustness to interference and allows for continuous transmission. The modulation pattern at the time of reception can be mapped back to the time of transmission, which results in a very accurate time difference and a resulting distance measurement. Use of unique or long patterns for pattern correlation also prevents interference that may originate from other Lidar units that are scanning in the same area. This results in a single distance measurement.

In order to get more than a point measurement, several different methods of scanning exist.

- **No scanning** – The result is a simple distance point measurement from the spot size on the object that is being pointed at.
- **Linear mirror scanning** – Scanning takes place with a stationary laser and receiver, but an actuated mirror turns this into a range.
- **Linear rotational scanning** – Similar to mirror scanning, but the whole unit is rotating. This gives full 360 degrees of vision with typically thousands of samples per second. Data and power are transmitted through a slip ring, which could be mechanically less reliable.
- **Phased-array scanning** – A high-speed solid-state method of scanning that results in a high-density 2D field of points. This typically results in millions of samples per second under a fixed field-of-view, but the scanning pattern may be arbitrary. These sensors use comparable technology to MEMS sensors, for which it has resulted in both a strong increase in capability and decrease in cost. But these have not yet appeared on the market.

For this application it is preferable to have full rotational vision, as this realizes full awareness in all directions of motion. It also has a minimal quantity of data, which helps with real-time processing on embedded platforms. The sensors that use phased-array scanning have only limited field-of-view, but they also measure at angles out of the plane. Phased-array scanning therefore has the capacity to detect low obstacles much better, but they also require significantly more processing. Note that even with phased-array scanning, the data above the height of the robot is irrelevant for obstacle detection and can be discarded.

As there is a strong dependence on Lidar, one assumption is that there are not prohibitively many reflective surfaces in the environment, such as glass or metal. Strong reflections from

such objects can corrupt the pattern and possibly cause the correlation to fail. This situation can easily be guaranteed in an experimental setup, and permits low-cost technology. But improvements in the technology are expected to improve the quality of pattern recognition and increase the signal-to-noise rate.

Standalone Lidar sensors begin at $400 for the Slamtec RPLIDAR[15]. Well-known are Hokuyo scanning rangefinders[16], which start at $1100. And the truly modern solutions such as the Velodyne Lidar[17] used on Google self-driving cars can cost as much as $70000. But in this case, mass production has done something wonderful and decreased production price of such a radial lidar to $30 [34]. A Lidar unit with similar specifications as the Slamtec RPLIDAR is now available in commercial vacuum cleaning robots such as the Neato Robotics Neato XV[18] series, and the whole robot can cost less (at $300) than even an individual Lidar unit does separately.

2.1.4 Simultaneous localization and mapping

SLAM has the potential to completely solve the localization problem from a single environmental sensor alone. For this reason it has been given considerable attention in research, both regarding visual sensors [18] as well as directly using landmarks [5]. However, such techniques involve processing a lot of data with computationally intensive algorithms, which are not well-suited for real-time evaluation in embedded systems. Recently exceptions have appeared, such as Power-SLAM [55] which has linear complexity in the number of landmarks and gives a continuously improving estimate, making it suitable for real-time performance. But this specific implementation requires uniquely identified landmarks and uses an extended Kalman filter for the implementation, which makes it not robust against position errors in single landmarks as are frequent with Lidar-based measurements. Extraction of sets of landmarks is possible, especially in indoor situations with straight walls that lend themselves to corner detection [71]. But sets of landmarks must be consistently identified which is difficult in obstacle-rich environments with occlusion. In addition, SLAM cannot directly apply to large open areas such as the lawnmowing scenario from the introduction, as no landmarks are available.

For these reasons, this thesis will not focus on such an implementation as more suitable methods are available. Note that SLAM is possible on this platform if a suitable algorithm is found as the sensor data is simply there. And indeed if the system is equipped with Lidar, SLAM has the potential to reduce position error to the order of millimeters.

2.1.5 Sensor fusion

There are many reasons why several different measurements should be combined to give an improved result. Dead reckoning as described in Section 2.1.1 is not only the result of odometry, but includes inertial sensors such as accelerometers and gyroscopes to accommodate

[15]Slamtec RPLIDAR, `http://slamtec.com/en/Lidar/A1`
[16]Hokuyo scanning range finder, `http://www.hokuyo-aut.jp/02sensor/index.html#scanner`
[17]Velodyne LiDAR Products, `http://www.velodynelidar.com/products.html`
[18]Neato XV robot vacuum by Neato Robotics, `https://www.neatorobotics.com/robot-vacuum/xv/`

for drift. And gyroscope drift can be compensated for with the magnetometer's slow absolute measurement to get a fast updating absolute orientation. The high-frequency low-noise *relative* accelerometer can be integrated to get distance traveled, which when combined with low-frequency high-noise *absolute* position measurements such as UWB or GPS results in high-frequency low-noise absolute position measurements. A system that accomplishes this is said to perform sensor fusion.

First, a method to combine these measurements into one statistical representation is needed, so variables that cover the full system state are selected such as the physical properties of the system. But the mathematical relations between these measurements are quite varied. So second, the equations that compute this multivariate state must be represented in another system, in which the contributions of individual sensors translate to the multiple states.

Estimating state under uncertain measurements is commonly solved by recursive Bayesian estimation, of which the Kalman filters and particle filters are well-known. Kalman filters are optimal estimators for linear systems that exhibit Gaussian measurement noise. For nonlinear systems, the extended Kalman filter performs linearization to obtain a state estimate [43]. A particle filter works with Monte Carlo techniques and can approximate nonlinear systems better than the extended Kalman filter [68]. However such Monte Carlo algorithms are not deterministic in computing time and are a poor fit for embedded real-time systems.

A classical Kalman filter makes certain assumptions.

1. All measurements have a zero-mean Gaussian error distribution with known parameters.
2. All measurements are performed at the same constant rate.
3. All measurements are taken instantaneously, with zero latency.

For such a filter, if the standard deviation has non-zero mean, stability of the state is severely at risk. In practice, no sensor has non-zero mean, so to guarantee stability all lower-order states must be directly estimated. A non-zero mean can be partially overcome by using a high-pass filter. If the true measurement error is larger than expected, the estimate will consistently overreact and become unstable. And if it is smaller, the system will not take optimal advantage of the accuracy and discard valuable data.

The filter will re-evaluate all measurements at every timestep, so measurements that do not experience the same constant rate will be repeated. A solution for (2) is a multi-rate Kalman filter, which dynamically sets the measurement errors to infinity during timesteps where they are not new. The Kalman gain is to be kept constant and one consequence is that the highest observed frequency must be a common multiple of all sensor rates. If not, the measurements will experience varying significance causing the system to become unpredictable.

The result of violations in property (3) is that the estimated value suffers additional measurement error quantified by the duration of the latency. For instance, if the robot has angular velocity of $\pi\,\mathrm{rad\,s^{-1}}$, then a 0.1 s delay of the orientation would incur an additional position error of 5 % over the traveled distance. This is a cause of significant drift, and must be minimized. Practical improvements in the implementation can help significantly, but have a limit. So a technique called delay compensation is discussed in [74], which essentially processes a delayed value as if it had occured at the time when it was new.

2.1.6 Review of localization

Many techniques and sensors have been observed, which will be summarized by their suitability to this application. The focus is on techniques that do not overwhelm the processor with data, yet provide the system with high quality measurements.

Of such techniques, a UWB system for localization conveys the best summary; a direct estimate for absolute position. At minimal processing time and with other prior considerations the Pozyx system seems ideal, but requires relatively costly infrastructure.

Odometry such as wheel encoders will likely come for free, and they give a low-order estimate for true measured relative distance. Furthermore, MEMS inertial sensors such as accelerometers and gyroscopes are essential for dead reckoning due to their robustness as they directly observe the unfalsifiable physics of the system. And a magnetometer is essential to stabilize orientation drift.

Lidar gives robust distance measurements in many forms, but all we are really interested in is obstacles. And for that, a radial scanning Lidar for efficient obstacle mapping seems best. Many versions of online SLAM exist that use radial Lidar data, but the localization algorithms may be too involved for low-cost computers so this is left to UWB.

Computer vision techniques that analyze the image of a camera may be too much for an embedded platform. But there are obvious advantages to performing simple pose estimation of unique landmarks, especially at the relatively low cost compared to UWB positioning.

And finally sensor fusion has the powerful capability to produce a unified state from all these sensors. Kalman filters are widely studied for sensor fusion, and their suitability for this application has been shown before. A multi-rate Kalman filter is suitable in particular as it can deal properly with various different sampling rates of the sensors.

2.2 Motion planning

Motion planning is a broad field in robotics and is discussed globally in much literature [39, 40, 67, 13]. The configuration space C_f particular for unicycle mobile robots has already been defined in the start of this chapter. There are many ways to generate paths in C_f and perhaps more importantly, how to follow them. All are part of motion planning. Navigation is a basic requirement for a robot, but it is not straightforward in difficult scenes, so specific strategies are needed. Collision avoidance is a more involved topic that can have consequences on different level of the system. And finally there is a whole class of algorithms dedicated to solving certain goals, of which this work considers coverage path planning. The next sections will introduce each.

Often, for the purpose of motion planning, localization is deemed ideal. This is not true for most practical implementations, and also not for the proposed platform. There are two positioning domains that matter for these systems. One is the mathematical and ideal absolute reference domain that all participants know about but cannot measure directly. Another is the observed domain under all uncertainties. Translating an update from the observed domain to the absolute domain is not straightforward as mapping parameters are always uncertain.

There is always an unknown positioning and orientation mismatch. Because of the orientation mismatch, distances that are observed further away from the robot generally have higher position error. In cooperative systems, this may be less true as position information that is local to one robot can be shared among others.

2.2.1 Trajectory tracking

For a robotics system it is important to be able to match a planned path to the actual motion of the robot. Physical limitations of the robot naturally restrict the planned path, as it does not make sense to control the robot past what it is capable of. Recall from the introduction of Chapter 2 that any path $f : [a, b] \to C_f$ is controllable in the simplified configuration space. The same is not true for arbitrary trajectory $t : [a, b] \to \mathbb{R}$, as any change in the tangent of f requires a corresponding change in orientation of the robot. As t maps the path to time, it is also restricted by the maximum $\dot{\theta}$ of the robot. Furthermore, instant change in velocities is not realistic, so in general $\ddot{p}$ is limited as well.

In literature there is a clear focus on generating geometrically smooth paths [54, 16] that solve this problem directly. Including some specific to mobile robot motion planning [69, 31]. Some [50] achieve piecewise composition under desirable properties such as G1 and G2 continuity, which describes the geometric derivatives of such paths. There are even methods to fit parametric curves to arbitrary paths [3]. But simple may be better. Robots that exhibit velocity control of the wheels get setpoints that result in circle arcs. Furthermore, any system that quantizes the curve introduces a piecewise path which has discontinuities.

On the other hand, ideally a path planner would be oblivious to the peculiarities of the robot and wants to suggest at most a sequence of setpoints. From such a planned path it may be possible to generate a feasible trajectory that under the dynamics of the robot can indeed be followed. And there are some well-defined systems from control theory for realizing this with multiple robots [60].

The major problem is with localization uncertainty. A planned path takes place in the absolute domain, while trajectories to execute are taken in the observed domain. The observed domain can suddenly jump based on corrected position measurements, meaning our controlled position is erratic. And a new trajectory will be recomputed around a continuously teleporting path, which makes it impossible to follow a predetermined trajectory. Control algorithms will have to be tuned to adjust for this and not overcorrect, or movement will be unstable.

From such an observation it becomes clear that accommodating the path to the physical capabilities of the robot is a futile effort. This justifies the oblivious strategy to path planning, where no system dynamics are considered at all. The primary goal then will not be to follow the path exactly, but to follow it as well as is possible under the circumstances at some set velocity. But even this set velocity may need deviation if maintaining velocity makes the system deviate too far from the path.

Another technique of implementing motion planning is by means of potential fields [15, 23]. Each pose in a potential field has a potential, resulting in a gradient towards a local minimum. Potential fields are commonly used for mobile robot navigation tasks, as they are well-suited for obstacle avoidance. But many limitations exist [35], with a major limitation that navigation

tasks in complex environments inevitably suffer from local minima. And if obstacle avoidance is used, passage between closely spaced objects may be enough of a repellent to completely exclude that path.

Despite their limitations for general motion planning, potential fields find use for trajectories under uncertainty. A potential field depends only on the path and environment in the absolute domain, so it remains static for a given planned path. Whenever the position estimate jumps, a new setpoint can be determined directly from the already defined potential field. Thus the whole vicinity around the path is well-determined by one system, which makes potential fields efficient for this application. Furthermore, the dynamics of the robot can be incorporated within the potential field, which can be defined for any pose $p = (x, y, \theta)$. Only care has to be taken that the defined potential field has a consistent gradient toward the end of the path and does not suffer from local minima on the way. And the field can even be tuned to the specific goal to be completed, such as when performing coverage path planning to prefer overlapping over leaving gaps.

2.2.2 Formation control

Significant research has been done on platooning [8]. This involves the coordinated movement of robots under certain strategies. Accommodating a formation during movement of several robots may help with collision avoidance which is otherwise likely due to the chaos of it, as seen in [14]. Or in the case of this application, coverage path planning as explained in the introduction is aided by such an apparent formation.

Some methods to realize this are:

1. **Behavior-based** formation control coordinates actions to have robots perform the exact same operations simultaneously. A potential field method corrects for detected variations.
2. **Leader-follower** approaches appoint one independent leader and has every other robot follow one other robot with a certain offset. This may lead to ripples in robotic response due to sensory delay, where the deceleration of one robot causes a larger deceleration in the robot following it.
3. **Virtual structure** methods treat the group as a whole as one system that is controlled. The individual participants from that group then adjust to always match the local virtual structure, even if its global behavior deviates.

Both the leader-follower approach [47] and the virtual structure approach [59] have seen use with unicycle mobile robots.

For the purpose of coverage path planning, a formation does not necessarily need to be rigid, as long as inter-robot distance does not become too great. In the presence of obstacles it would be useful if the robots continue to observe each other. Furthermore, each robot is interested in improving the percentage covered, meaning the overlap with other robots must be quite rigid.

The behavior-based approach and virtual structure approach both value the formation above local goals, so they are less suited for this application. Using the leader-follower mechanism the only control concern is to get the overlap with the past actions of the leader as exact as

possible. This may even ignore the distance to the leader to a degree, as long as it can be accurately observed for the subsequent coverage path planning. So a leader-follower method is likely suitable for the purpose of formation control relevant for this application.

2.2.3 Collision avoidance

Collision objects consist of static ones (walls and fixed objects) and dynamic ones (people, other robots). A collision is a state of uncontrolled travel into any such object. Uncontrolled in this case means the laws of inertia will transfer energy in unpredictable ways and the robot will not move according to its model. For dead reckoning style navigation, the chaos of such a collision event usually results in a major error. Therefore, collisions are best avoided.

Several methods to perform collision avoidance exist. With a deformable virtual zone (DVZ) [38], a safety region is defined around the robot that grows as velocity increases. Obstacles that enter this region can trigger a response, such as adjusted movement away from the obstacle or an appropriate decrease in velocity. With the dynamic window approach (DWA) [46], a forward simulation of various velocities of the robot is performed. Any paths that result in a collision are forbidden, by which the method completely excludes any chance of collision. Receding horizon control is a method for path planning with intrinsic collision avoidance that works especially well for high-velocity systems [62]. Collision avoidance can also be directly incorporated in the control algorithms, as in [36]. Potential field methods are especially suited for natural collision avoidance, and are well-studied for it [15]. And methods exist that are specifically tuned to coverage path planning [30].

One problem with DWA occurs when it is used to guide path planning. By choosing the best of remaining valid solutions, the goal may be inaccessible. For instance if the robot arrives next to a narrow alley at high velocity and wants to enter it, DWA prevents it from making a wide turn and crashing into the walls. But instead of preemptively slowing down or approaching the alley head-on, it will now miss the turn entirely. This is in contrast to the DVZ, which may not always restrict the robot sufficiently to actually prevent a collision. It just permits safer behavior in the proximity of obstacles, and the actual response to obstacles in the zone must be tuned appropriately.

Furthermore, a robot has no model of the dynamic behavior of obstacles that are moving. With DWA such obstacles are not considered until the point where they block the current path. So the only hope is that deceleration capacity is sufficient. Whereas DVZ uses heuristics for objects within this zone and may become increasingly careful as obstacles begin to change.

In addition, other methods for collision avoidance may be employed. Coverage path planning algorithms are already keeping track of the environment with a map, which includes obstacles. From such a map, collision-free paths can be selected easily. Then the problem with this is again position uncertainty, where the robot is closer in reality to an obstacle than expected. Both DVZ and DWA seem well-suited to a system that is equipped with rotational range detectors such as Lidar. Such a range detector should be the primary measurement for collision avoidance as it is independent of the effect of jumps in position that occur for map-based methods. If rotational Lidar is combined with a DVZ, the system can become quite robust and safe. Note that any method for collision avoidance can be defeated by sufficiently high-velocity obstacles.

2.2.4 Navigation

Navigation of robots through a space is an essential ingredient for motion planning. In the absence of obstacles, a straight line can be drawn from start to destination. But in an indoor situation, obstacles are plentiful and the environment has similarities with a maze, where dead ends should be avoided. So commonly, navigation implies pathfinding toward some goal, which includes a natural form of obstacle avoidance.

Many methods exist to perform navigation. Potential field methods such as [21] can be tuned specifically for the nonholonomics of unicycle mobile robots. But as was discussed in Section 2.2.1, a planned path can be oblivious to the dynamics of the robot. Methods that use cellular decompositions of the space build graph summaries that show reachability between regions. These include the incrementally constructed trapezoidal decomposition, and the Boustrophedon decomposition [1] which is sees popularity among coverage path planning. These have a close relation to generalized Voronoi diagrams, which can also be directly constructed from Lidar range measurements [45]. A strong point is that graph search algorithms on such summaries can efficiently satisfy completeness of navigation. Completeness says that if a path exists it will be found. And for practicality, if it does not exist further exploration may be needed. But the difficulty is that such a graph has to be constructed dynamically in an online manner, as the environment changes. Even more problems are involved when one considers distributed updates to such a datastructure.

Graph search algorithms can also operate directly on quantized maps of the space [13]. The standard A*-algorithm is popular for this purpose, but incurs significantly more operations than summary-based methods. Especially when the distance between source and destination is large. In practice, maps are incomplete and may be outdated. Due to dynamic obstacles however, all attempts at navigation can potentially fail and a solution to reach the destination may not exist. Exploration and online replanning then become significant. And as the robot is moving at some velocity, the time available for replanning is limited, making continuous re-evaluation of the A* algorithm a significant problem. So-called anytime algorithms have become popular for robot navigation as they still result in potentially suboptimal solutions when time runs out. Informed incremental versions of these algorithms reuse findings from prior runs, as with D*-Lite [33]. Furthermore, dynamic versions of these can deal with changes in the environment, such as with AD* [42]. These offer a complete solution well-suited for real-time incremental navigation with mobile robots.

For coverage path planning however a cellular decomposition is typically computed anyway. So it makes sense to combine both: a summary graph based search for the fast guarantee of completeness of navigation for long-range goals, with a method that is suitable for local dynamic online cell-based navigation that can take short routes around obstacles.

2.2.5 Area coverage

This problem is known as coverage path planning and has been researched in many different ways [19].

Frequently a subdivision of the scene is made into regions, as part of a divide-and-conquer approach. Such a coverage method is used by [10], which performs the cellular decomposition

of the scene by Boustrophedon decomposition. A robust online variant of this is described in [29]. And multi-robot variants are also common [58]. Such cellular decompositions can then be combined into regions for lane-based infills, as is done for agricultural robots in [11].

With coverage control, both coverage path planning and trajectory control are unified under a set of control laws. Such a set of control laws can even realize complete coverage without any determined plan other than the state of coverage [30].

Energy-efficiency of these paths can also be a direct concern [17]. However minimizing goal completion time and thus path length is strongly correlated with minimization of energy. Other research has been focussing on minimizing the number of turns [4], as turns are expensive in terms of energy and time. When performing a turn the system has lower-than-optimal velocity and the robot has to convert its forward momentum.

Other techniques also exist. Such as the linked-spiral method of [9]. The use of neural networks in [72] and [44]. Or the spanning tree method described in [25], of which [76] describes a multi-robot approach.

For multi-robot approaches several strategies can be used. One of which is to distribute the robots among the scene which results in an almost linear speedup in the number of robots [28]. Another is to perform formation control [58], where the goal to stay in formation with respect to side-distance is more important than the goal to follow the trajectory. The latter avoids unnecessary gaps within lanes and gives wide coverage while reducing overlap, again with linear speedup in the number of robots, but with less total path length.

Most multi-robot approaches assume some sort of reliable communication between the robots. Unreliable communications has been investigated among coverage control [53]. And some techniques do not necessarily need communication as they can determine relative distance directly from sensors, such as with leader-follower based formation control.

Another difficulty are methods that work under the constraints of position error. A solution ideally consists of multiple robots in formations with minimal overlap. But because of the uncertainty, many techniques that plan paths in advance will result in gaps. Methods to correct for overlap such as potential field based formation control of Section 2.2.2 inevitably deviate from a globally preplanned path. So an incremental online coverage path planning algorithm is desired.

Such real-time replanning of coverage path planning has also gained recent attention in [20]. This technique is for a single robot and uses lanes. But when multiple robots turn at the end of a lane, the robots will be forced to leave gaps as they reorganize, which may not end up being covered. So a spiral algorithm seems like it is better suited. This allows all robots in the formation to perform an incremental online wall-following algorithm continuously while preserving relative overlap with the leader, where the wall is a virtual border between the already covered and remaining open regions. Such an approach is unlikely to leave any gaps and seems to optimally minimize overlap in large open regions.

2.2.6 Review of motion planning

Even though motion planning is a complex field with much innovation in terms of algorithms, only few solutions take into account all practical aspects experienced by the robot. And solutions to various problems have to be combined to develop a suitable strategy.

Trajectory tracking with a local potential field allows arbitrary path following, where solutions for the nonholonomic constraints are provided by the potential field alone. This method is particularly suited to planning under position error. A global path planner can then be oblivious to these constraints and provide any setpoints as it desires. Map-based navigation has been considered from two perspectives, locally the AD* algorithm is used on a grid-representation, and for distant navigation a cellular decomposition of the space makes the navigation method both complete and fast. To prevent collisions under position error, the deformable virtual zone (DVZ) technique is suitable with heuristics based on a risk assessment of the environment. The contents of this zone is directly derived from Lidar distance measurements. This incidentally enables the leader-follower style formation control that can benefit spiral-based coverage path planning. And further tuning of the potential field for trajectory tracking allows for optimizations to steering for coverage and overlap regions.

Such an approach offers complete motion planning with several optimizations that suit performance characteristics of embedded systems and unicycle mobile robots under position error. It also leaves room to further refine heuristics that can improve final performance.

2.3 Systems integration

Several prior attempts have been made that integrate much of what is needed for a full system. But the solution presented here is relatively unique. This section will show the final ingredients necessary for integration.

2.3.1 Radio positioning

Positioning robots based on radio signals has received considerable attention. In [48], a multi-rate Kalman filter sensor fusion implementation is realized with several inertial sensors and GPS. The paper describes its implementation in detail and evaluates the setup on a manually driven car. In rare cases integration similar in size to the scope of this thesis is attempted. One such attempt from [52] aims to integrate both localization and motion planning on the basis of an RFID infrastructure. The focus of that work is heavily related to control theory and to a lesser degree generic software techniques. But the hardware radio design is accomplished from the ground up, resulting in autonomous control of real robots.

2.3.2 Mechanical platforms

Several affordable robotics platforms exist from multiple manufacturers. Considering their cost and the benefits that come with well-tested platforms, it is quite tempting to choose an

off-the-shelf solution. The listed platforms are mature and have known interfaces for control, which reduce development time.

- **i-Cart mini** is a very minimalistic but perfect example of a unicycle mobile robot.
- **Adept MobileRobots Pioneer** is a very varied platform worth exploring, but also expensive.
- **PMB-2** can carry quite a payload and is exactly what an industrial evaluation of a robotics platform would look like. It is however not low-cost.
- **Kobuki** is meant purely as a robotics research platform without direct purpose. This is designed for robotics and control, so it should be a great fit for many research applications.
- **iRobot Create 2** is a very cheap robotics platform without vacuum function. It is not available in Europe.
- **iRobot Roomba** is a reliable vaccuum cleaning robot that comes in many editions. It uses the probabilistic method of space coverage with direct-contact sensors so it is rather poorly equipped in terms of sensors.
- **Neato XV Signature** is a vacuum cleaning robot that has Lidar built-in and performs SLAM for whole-room coverage already. It is not intended for robotics control, but a debug port on the unit enables such behavior nevertheless.

As the choices are limited and features are varied, this makes for some easy decisions. The Kobuki platform seems perfect but if a separate Lidar is included, the price is twice as much as its competition. The iRobot Create is by far the cheapest platform at $200 without Lidar, and supports programmatic control, but its lack of availability excludes it. Since the features that the mature Neato XV Signature platform offers cannot be matched by any other robot, it is the obvious remaining choice. Some work may be required to get it to behave, but prior efforts instill confidence.

2.3.3 Software platforms

Computing platforms are becoming increasingly more low-cost as IC technology matures. Five categories of platforms are identified which classify most software platforms currently on the market.

1. **Microcontroller (MCU) platforms** contain low-power and low-capability ICs which are not capable of running algorithms that require more than several kilobytes of data memory. They commonly follow the Harvard architecture, where instruction memory is separate from data memory, and are useful because of their simplicity. Their behavior is predictable, which helps to ensure real-time performance. Such platforms typically carry the lowest cost, as they are simple and there is much competition. The Arduino platform[19] has seen much popularity for its entry-level IDE that can use many different types of electronic components.

[19] Arduino Foundation, `http://www.arduino.org/`

2. **System-on-Chip (SoC) platforms** contain significant amounts of memory and processing power in an extremely integrated package. They are much like common computers in that they follow the von Neumann architecture, where both instructions and data are stored in the same memory. Their widespread application in smartphones has lowered cost to the point that they are increasingly adopted as a central component for many embedded platforms. This leads to an abundance of computing power for embedded platforms, which are commonly referred to as single-board computers (SBCs). However these are typically less flexible than personal computers as every component is fixed and less busses are exposed for expansion. One such example is the The Raspberry Pi platform[20], which has seen recent popularity due to its low cost and usability.

3. **Signal processing systems** typically find use in highly optimized workloads. These often contain special ICs such as a digital signal processor (DSP), complex programmable logic device (CPLD) or field-programmable gate array (FPGA). Such ICs can be configured to execute certain algorithms with great performance, but such algorithms must be specially designed for the target. In addition to this special IC, they usually also contain a microprocessor to support configuration, control and possibly some further pre- and postprocessing that does not fit the specialized IC too well. A good example of this is the Parallella platform [56], which contains a 16-SoC RISC architecture with an additional FPGA and ARM SoC.

4. **Personal computers (PC)** are the typical consumer desktop or laptop systems with separate CPU and often removable storage and volatile memory. Many standalone versions for embedded use exist, also referred to as SBCs. They are usually equipped with the prevalent components typically found in consumer computers, such as processors with the x86 instruction set architecture. What still sets them apart from the SoC market is that their architecture is designed for expansion, as evident by the many busses and sockets. For instance the peripheral component interface (PCI and PCI Express) is used by many third-party products for graphics, storage, networking and others.

5. **High-performance computers** share much similarity to PCs, except they contain hardware that is specialized at performing certain tasks. Frequently seen in servers, it can consist of a motherboard that supports many standard components simultaneously, such as multiple CPUs. Some configurations even increase the volatile DDR memory into the terabyte range. The real success can be seen by not just multiplying hardware, but also with custom specialized hardware. Recently, solid-state drives have been able to offer an order of magnitude higher number of IOPS over traditional hard disk drives at comparable cost. And in the recent trend of general-purpose computing on GPUs (GPGPU), grapics cards can be used for executing optimized algorithms. One example is the NVidia Tesla P100[21] which advertises performance at 21 TeraFLOPS. This makes such systems preferred for the development and implementation of algorithms that do much arithmetic, which is typical for computer vision.

The ordering of this list is deliberate. From top to bottom an increase is observed in both capability, but also power consumption and price.

Robotics research platforms usually opt for a PC, sometimes equipped with a GPU if vision

[20]Raspberry Pi Foundation, `https://www.raspberrypi.org/`
[21]NVidia Tesla P100, `http://www.nvidia.com/object/tesla-p100.html`

Product	CPU	RAM	GPIO	I2C	UART	SPI	USB	Wi-Fi	Cost
Raspberry Pi Zero	– –	512	26	2	1	1	1	–	$5
C.H.I.P.	– –	512	80	2	1	1	1	+	$9
Pine A64	+	512	46	2	3	2	2	+	$15
Raspberry Pi 3	+	1024	26	2	1	1	4	+	$35
Odroid-C2	+ +	2048	47	2	4	2	4	–	$40
BeagleBone Black	–	512	68	2	4	2	1	–	$49
Intel Edison	+ +	1024	26	1	1	1	1	+	$76

Table 2.2: Comparison of several popular System-on-Chip platforms. CPU in performance from + + to – –, RAM in MiB.

processing is done. This guarantees that they have sufficient computing power available for the algorithms that are developed. However PCs usually do not perform direct digital IO well, as this is not a common use case. Recently SoC platforms have been improving significantly to the point where they are on-par in terms of computational performance of entry level PCs, at much lower total cost and power consumption. In addition SoC are relatively close to MCU platforms when it comes to controlling electrical voltages and other digital hardware. This makes them especially suitable for interfacing the sensors described in Section 2.1.

A closer look at several SoC platforms is given in Table 2.2. All of these run some distribution of Linux and have similar support for devices and software. The table lists primary performance characteristics and connectivity options for external hardware, as well as price. The most capable platform appears to be the Odroid-C2, which happens to be the most recent. The Raspberry Pi Zero platform is attractive at its very low price point. But both the Odroid-C2 and Raspberry Pi Zero lack wireless connectivity, which when added over USB would increase cost. The C.H.I.P. platform is the best choice for a minimal and optimized system as it has both Wi-Fi and 4GB storage included, which the Raspberry Pi Zero does not have. The Raspberry Pi Zero, C.H.I.P. and BeagleBone Black have lesser computational performance, with only 1 slow core available. As this may be restrictive for development, they are avoided. The Intel Edison platform is considerably more expensive with little advantages. Both Raspberry Pi platforms have a connector for an external 5MP or 8MP camera for $25, which is a high-bandwidth device that benefits from the CSI interconnects found in modern smartphones. Using a USB camera instead gives lower quality and image resolution at higher cost. So a Raspberry Pi is preferred for many projects that incorporate computer vision. Cumulatively, the most suitable and versatile candidate is the Raspberry Pi 3 platform. If computer vision is not desired, the Odroid-C2 platform is the better choice.

Chapter 3

Design

Design of the system is decomposed in three parts. The hardware architecture governs the physical capabilities of the system including the sensors. The software architecture defines the operational framework that allow this hardware to be used. And the algorithms define the high-level behavior that actually use the system to accomplish goals.

3.1 Hardware architecture

The whole system can be broken down into several physical units. The logical parts of the system have been named in Chapter 2, as well as several candidates for their implementation. These will now be combined into one concrete and coherent system, to propose the hardware architecture.

A diverse combination of technologies is needed to optimize performance, and cost is a major restriction. Designing a whole robot from scratch is too big of a task and will likely lead to many iterations and high development cost. Economies of scale teach that low-cost solutions are found in mass-produced products, as was confirmed in Section 2.3.2. A good starting point is to choose such a commercial platform as the base of development and expand upon it. The Neato XV series of vacuum cleaning robots is without competition, as it is cheaper than even a standalone Lidar unit. Any other choice of platform would lead to a significantly more costly solution to achieve similar performance. The robot platform provides actuation and several essential sensors for free, see Section 3.1.1. Supplementary sensors are selected in Section 3.1.2. Yet the platform does not support external sensors, and software development is also not possible. As a workaround, the platform and all sensors are connected to an external computer, described in Section 3.1.3. This completes a functional system that is fully independent. But with communication and localization infrastructure it could perform much better. This infrastructure is explained in Section 3.1.4. Figure 3.1 shows the resulting functional decomposition of the hardware.

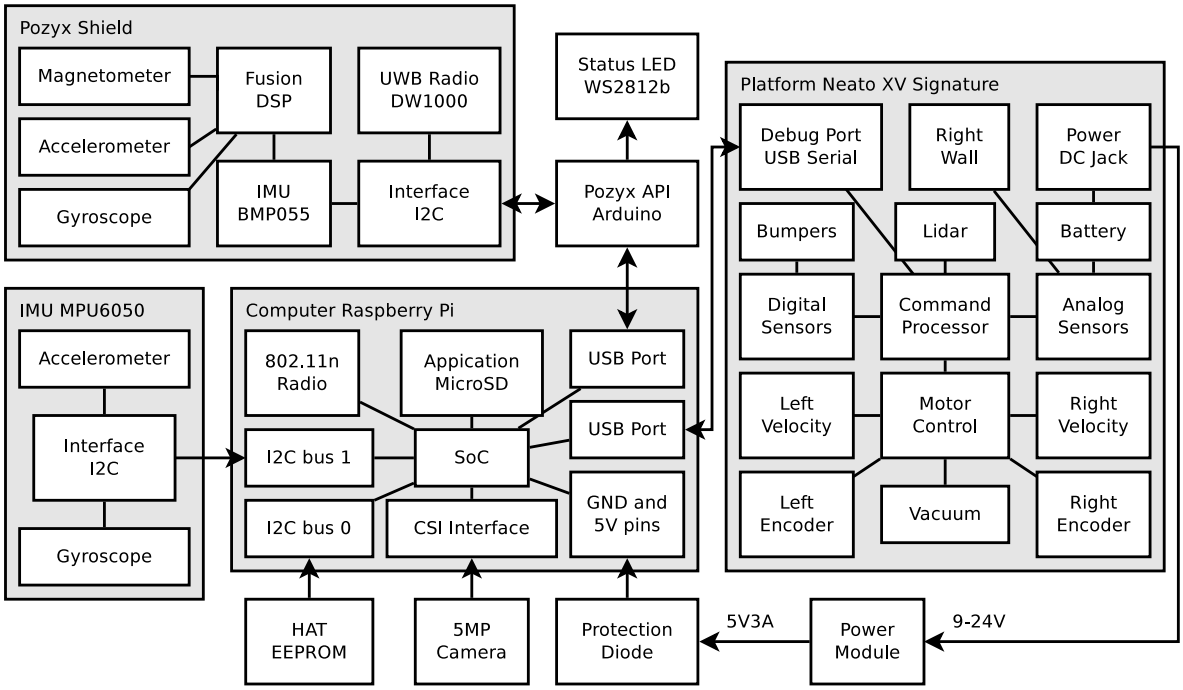

Figure 3.1: Functional block-level diagram of all related physical hardware.

3.1.1 Mechanical platform

The Neato XV series of vacuum cleaning robots stood out in Section 2.3.2 in terms of cost and environmental awareness. Despite all the advanced sensors inside, mass production makes the system affordable. For this project a Neato XV Signature was purchased of hardware revision 64, firmware version 3.4. It contains both a radial scanning Lidar at 5 Hz and an exposed USB debugging port. Figure 3.2 describes the physical layout. The unit has a 0.24 m wide brush at the front, a diameter of 0.3 m with equally wide square front, wheel centers spaced 0.24 m apart, and a maximum velocity of $0.3\,\mathrm{m\,s^{-1}}$. The bounding radius r that is used for the flattened configuration space C is $\sqrt{2 \cdot 0.15^2} \approx 0.213\,\mathrm{m}$ due to the square front. Some areas cannot be reached and will not be cleaned unless specific path optimization is done for the shape of the robot, which is omitted for this generic platform. There is an exposed external power jack for external charging, which we will repurpose to provide power out for the computation platform.

Enabling the unit's debug mode offers full control over motors, encoders and Lidar output, as well as other sensors. This debug mode does not deactivate the Neato's safety features completely, and is not meant for high-speed operation but rather controlled testing. Among the other available sensors are four front bumpers for collision detection, a wall sensor on the right of the robot, a drop sensor to detect lifting, Hall effect sensors for magnetic marking of areas, an accelerometer and status information about the battery and user interface. Commands are atomic and handled at a fixed rate of 100 Hz

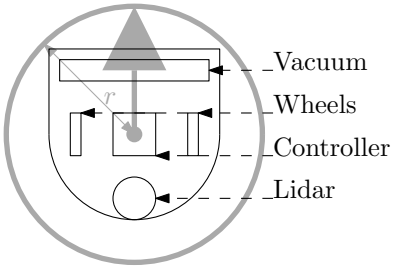

Figure 3.2: Platform layout superimposed on its representation.

Symbol	Unit	Sensor	Equation	Rate	Error
x	m	UWB	x	10 Hz	noise, interference, calibration
y	m	UWB	y	10 Hz	noise, interference, calibration
θ	rad	magnetometer	$\theta_0 - \theta_N$	100 Hz	noise, interference
$\dot{x}$	$\mathrm{m\,s^{-1}}$	encoder	(3.1)	25 Hz	discretization
$\dot{y}$	$\mathrm{m\,s^{-1}}$	encoder	(3.1)	25 Hz	discretization
$\dot{\theta}$	$\mathrm{rad\,s^{-1}}$	gyroscope	ω_z	1000 Hz	noise, bias
$\ddot{x}$	$\mathrm{m\,s^{-2}}$	accelerometer	(3.2)	1000 Hz	noise, bias
$\ddot{y}$	$\mathrm{m\,s^{-2}}$	accelerometer	(3.2)	1000 Hz	noise, bias
$\ddot{\theta}$	$\mathrm{rad\,s^{-2}}$	–	–	–	–

Table 3.1: Relation between the pose and sensors, with sample rates and error types

by this debug port, so scheduling of commands is important for final performance.

The implementation should maximize performance of this interface as much as possible, and work with the properties of the resulting system. If the debug port is found to be sufficient for control purposes, the platform's control system can stay in place and minimal modifications are necessary. All additional electronics will be powered from the platform's battery.

3.1.2 Sensors

Sensors contribute to an accurate estimate of the robot's own pose $p = (x, y, \theta)$ in the global configuration space C_f. p relates to velocity ($\dot{p}$) and acceleration ($\ddot{p}$) for which more frequent and better direct estimates are available. Table 3.1 lists the types of sensors that contribute to these estimates. It is obvious that high update rates for x and y clearly depend on integration of accelerometer data. A known problem with integration is that the constants of the lower-order estimate may be left undefined and can drift. To avoid long-term instability, a direct estimate is required for every lower-order estimator. Table 3.1 shows this to be the case, so stability of pose estimation is inherent to the design.

A secondary purpose of the sensors is to fill in the configuration space $C = C_f \bigcup C_o$, for aid in motion planning. The platform already provides a practical Lidar scanner for relative positioning and obstacle detection.

The platform's wheel encoders can be used for relative positioning and velocity measurements. The wheel encoders have relatively low resolution as pulses occur only once every 1 mm of travel. The encoder pulses are are handled real-time by the platform, which provides an interface for reading out the pulse counters. A method to convert wheel encoders into velocities is derived from Equation (11.14) of [66] as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{\cos(\theta)}{2} & \frac{\cos(\theta)}{2} \\ \frac{\sin(\theta)}{2} & \frac{\sin(\theta)}{2} \\ -\frac{1}{d} & \frac{1}{d} \end{bmatrix} \cdot \begin{bmatrix} v_l \\ v_r \end{bmatrix}, \tag{3.1}$$

where d is the distance between the two wheels. At 25 Hz readout rate, discretization noise is significant. Therefore it depends on smoothing to use this estimate of velocity, which in

turn increases latency. The current velocity setpoint is likely a better low-latency estimate, as velocity control is achieved within the platform by means of an optimized feedback loop.

The system design depends on high-speed low-latency local measurements of relative acceleration $a = (a_n, a_f, a_z)$ and angular velocity $\omega = (\omega_n, \omega_f, \omega_z)$ for integration, with their normal, forward, and z-components. The platform's communication bus only has 100 Hz command capacity which is shared among many functions, so reading the accelerometer and gyroscope inside of the platform interferes and is not practical. Therefore, inertial measurements are done with a separate module containing the MPU6050 Inertial Measurement Unit. Connecting this module directly to the computer gives high sample rates of the accelerometer and gyroscope. These inertial sensors are oriented on the robot as seen in Figure 3.3. The measurements from the accelerometer are relative to the robot's heading angle θ, and are rotated with

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = R(\theta) \begin{bmatrix} a_f \\ a_n \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} a_f \\ a_n \end{bmatrix}, \tag{3.2}$$

to get absolute acceleration. The only degree of freedom in rotation is that around the z-axis, and the robot is restricted to movement in the plane. This makes the direct equation $\dot{\theta} = \omega_z$. By design, a_z, ω_n, and ω_f are always zero, and they can be ignored or measured to indicate error. Note that a_n represents centripetal force measured during curved paths or wheel slipping, so it is still relevant to incorporate this.

A magnetometer provides a direct measurement for the absolute orientation $\theta = \theta_0 - \theta_N$. The robot measures θ_N, the angle between the forward direction and the magnetic North. θ_0 is the constant angle between magnetic North and the x-axis in the world frame.

For estimating x and y directly with absolute positioning, Pozyx UWB localization tags are used. The coordinate system of the UWB localization system is adopted directly as this is shared by all robots. In this system, all anchors u_i are at some constant height and numbered from 0 with u_0 as the origin. To uniquely determine the x, y-plane, $u_1 - u_0$ is the direction of the x-axis, and u_2 has positive y, as seen in Figure 3.3. The infrastructure is further explained in Section 3.1.4. Pozyx tags have an interface that is pin-compatible with the Arduino format, and supports 3.3 V operation. However the Pozyx API firmware runs only on the Arduino platform, so a real-time Arduino-compatible microcontroller is introduced to act as an intermediate. One advantage of this setup is that the Arduino can prepare data in advance in real-time for transmission to the computer, which reduces the latency to that of communication overhead between the computer and Arduino. The Pozyx system requires calibration of the anchors, and offers both manual static configuration and an automatic calibration method. It is important that the exact same reference is used repeatedly among all robots as inconsistencies are a potential source of error, so for experimental setups a static configuration is stored and shared. The Pozyx API provides both ranging and localization methods, and simplifies obtaining absolute x- and y-estimates to reading from a serial port on the computer.

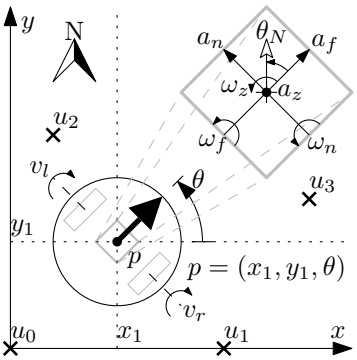

Figure 3.3: Relation between sensors in the frame of reference.

In addition, a standard Raspberry Pi 5MP high-resolution camera module is equipped for pose estimation. This serves two functions. With the camera pointing toward the front of the robot,

relative distance between robots can be accurately measured for formations. With the camera pointing towards a ceiling equipped with unique landmarks, accurate absolute pose can be determined to validate the measurements of p. Including it makes the setup more complete and versatile as a research platform. The camera has been developed by the Raspberry Pi Foundation with computer vision algorithms in mind, which can now be implemented. Finally, it could serve as an indoor alternative to the relatively expensive UWB localization system, making a potential solution extremely low-cost.

3.1.3 Computer

The Raspberry Pi 3 Model B offers a quad-core ARM SoC at 1.2 GHz with 1 GB RAM. It has an integrated 802.11 2.45 GHz Wi-Fi radio for high-bandwidth (54 Mbps) wireless communications. An interface to all attached components is provided by four USB ports and a 40-pin 3.3 V GPIO header. The platform will be connected through a USB port.

The power consumption is 700 mA at 5 V, which it gets from a 5 V 3 A buck converter between the platform's 14.4 V NiCD battery bank. The Raspberry Pi cannot shut down to the point where it is not consuming any power due to design limitations. Circuitry is added to allow a true shutdown and poweron via momentary button or IO pin. Software on the computer can then detect the platform's battery level and react accordingly.

To interface with external hardware, an adapter board is attached to the 40-pin header. This adapter follows the Raspberry Pi Foundation's HAT specification (Hardware Attached on Top) and provides the following functionality:

1. Power-down management circuitry
2. Power protection diodes
3. Buck converter from platform battery supply
4. MPU9250 IMU board conversion logic
5. An Arduino shield header configuration for the Pozyx tag, with protection resistors between IO lines.

A protoboard version of this adapter board serves to quickly validate functionality of the schematic. This custom electronics will be placed on top of the robot, using the elevated Lidar housing as a base for attachment.

3.1.4 Infrastructure

Each robot will participate in an ad-hoc mesh network over their 2.45 GHz 802.11 Wi-Fi radio for communication. Besides their integrated radios, this requires no external infrastructure, the shared configuration is purely virtual. Communication is packet-based, but due to unreliable transmissions some packets may be lost. The TCP protocol provides means to retransmit messages, which ensures reliability even under packet loss, at potential extra latency. As long as the mesh network forms a connected graph by wireless range, packets between any two robots will arrive.

For absolute localization, Pozyx anchors will be positioned around the space. Ranging between four anchors is required to make Pozyx tags on the robot perform localization. Line-of-sight between anchors and tags reduces the interference, so practically anchors are mounted high up on walls surrounding the area of navigation. For best accuracy, six such anchors will be placed within range of the 2D plane of navigation. Pozyx anchors must be located near power outlets and are not waterproof, so they require an indoor setup. These operate on UWB radio frequencies, which work relatively well through walls, as 20 m to 30 m indoor range is advertised. The exact configuration of the anchors beyond these constraints is different for every environment, so it will not be treated until the implementation in Section 4.1.5.1.

For charging, each robot will dock with a charging station. This is done by backing up against a special powered base that can be detected with platform's Lidar. The location is memorized, and the robots can detect occupancy and measure accurate relative position to dock successfully.

3.2 Software architecture

The software is designed for quick iteration and testing of features, and modularity is a core aspect of this. The interfaces will be simple and modules are open for extension. As much as possible, implementation-specific concerns are isolated to the local modules. And the architecture is intended to reflect this. Note that performance should not take priority over maintainability and best practice, unless performance is otherwise so bad that it prevents a goal.

System requirements state that this software should run on a multi-core embedded robotics platform. As it is a robotics system that processes sensor values directly, it experiences some real-time characteristics. These are explained in the Section 3.2.3 and lead a significant part of the design. As many tasks will be occuring simultaneously, single tasks must be prevented from starving other tasks. Management of execution priorities is therefore necessary. Since such multi-process systems may be hard to analyze and debug when something goes wrong, the system must allow for inspection.

The hardware that was selected runs a standard Debian Linux operating system which is very accommodating as a development platform. For the languages, Python 3 is chosen for most algorithms and modular structure as it is a readable and high-level language. Where necessary, C is used for local optimizations, but only if necessary to meet performance goals. C integrates very well with Python. The webserver for communication is also implemented in Python 3 using readily available libraries. In one instance, web-based client-side code uses JavaScript. For service management, simple Bash scripts and Systemd service definitions are written. Code is pushed to the robot through Git, which has hooks for automatically redeploying all services after commits.

3.2.1 Code organization

The codebase is subdivided into modules that abstract from the details of individual subproblems. The modules are chosen such that their dependencies are minimized and interfaces can

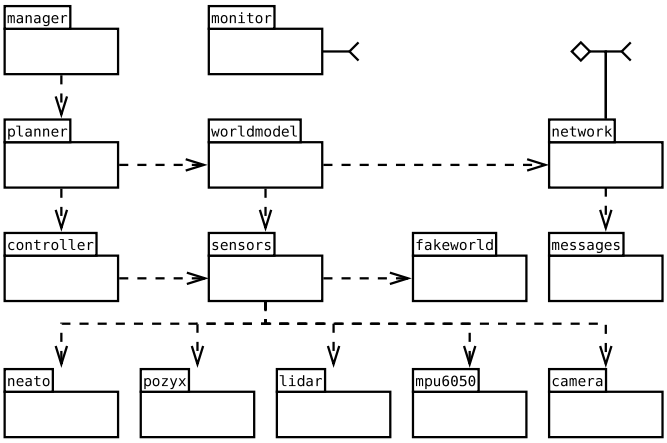

Figure 3.4: Dependency graph of system modules.

be well-defined. The following modules and their dependencies are shown in Figure 3.4.

- `manager` provides high-level orchestration of all robots by means of task allocation and delegation. The manager only manages the task queue for the planner. It performs reactive scheduling, mostly dealing with planner errors as completion cannot be guaranteed. Depending on the desired setup, managers of several robots can work cooperatively, or be a slave to one central manager that manages all robots' task queues directly.
- `planner` contains all algorithms for dynamic online generation of paths necessary to fulfill given tasks, like navigation and coverage path planning. The resulting path is offered to the controller.
- `worldmodel` holds the data structures and update methods for the derived world representation. This world model W is used for all high-level decision making. Incoming and outgoing state changes are also processed and prepared in this module. The optimal telemetry rate is determined by shaping the data volume to current network capacity.
- `controller` contains the low-latency control algorithms necessary for high accuracy and smooth path following. It contains a high-performance filter for sensor fusion to get the best instantaneous pose estimate p to directly actuate the wheels. This module has real-time characteristics and receives the sensor data through a low-latency path, ignoring W entirely.
- `sensors` provides a generic interface for sensor access and processing. This enables caching and thus access for more than single dependencies, and provides hooks to override functionality for the purpose of simulation.
- `network` provides message routing and communication channels between robots. It serves the functions of service discovery and connection management, as well as means for message transport across communication channels. A feedback system for bandwidth management can signal channel congestion.
- `messages` defines the communication language between robots as high-level objects. All communicated world updates and external commands are defined here. It provides message (de-)serialization.

- `fakeworld` provides alternative (virtual) sources of sensor data and defines physics in a virtual world for simulation purposes.
- `monitor` serves as a debugging tool, it connects to all available robots over their network interface and provides a web-based visualization of the world model from all messages that are transmitted. The design of this module is as non-invasive as it can be, as it attaches to an existing communication bus as an external system.
- `neato` is a driver and abstraction layer for all sensor events and actuator control on the Neato platform.
- `pozyx` provides an interface to the Pozyx Arduino API and algorithms for processing Pozyx UWB tag data.
- `lidar` contains algorithms for aggregating and processing Lidar data.
- `mpu6050` is a driver for the inertial measurement unit.
- `camera` contains algorithms for performing pose estimation on image data.

One architectural choice that greatly simplifies the implementation is the abstraction of sensors in one module. This enables sensor fusion and the world model to coexist with different performance requirements and without additional complexity. And it also makes for an elegant way to support simulation in this system.

Note that the `controller` module is connected to the `sensors` module while avoiding the `worldmodel` module. This part of the system has to meet real-time performance and all its dependencies are carefully considered. The details of this separation can be found in Section 3.2.3.

3.2.2 Data structures

All sensors contribute to knowledge of the environment, which is stored in the world model $W = (W_S, W_E, W_N, W_C)$. The data structures that store this information are optimized for typical queries in the system. For typical applications it is useful to store three types of data; a map of obstacles or free space W_S known as the space map, a database of named locations W_E known as the entity map, and a structure to simplify navigation W_N known as the navigation graph. For the application of coverage path planning, an additional data structure is necessary to keep track of the state of coverage, W_C, known as the coverage map.

3.2.2.1 Space map

For generic collision-free navigation in the free space C_f, the local surroundings must be known. Path-finding queries will be mostly based on proximity by recursively evaluating neighboring areas. In practice, storage precision below that of positioning noise does not matter, so a form of quantization can simplify storage. The quantized space $S = \mathbb{Z} \times \mathbb{Z}$ is then a logical 2D grid derived from the configuration space C, with $s : C \to S$ mapping position to cells. When stored as an array S gives predictable performance with constant-time arbitrary lookups. The size is not prohibitive as C_f is not sparse and practically bounded by

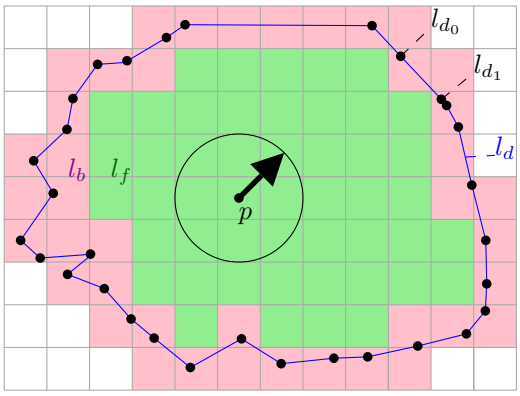

Figure 3.5: Lidar data $l = (p, l_d)$ quantized with s to an obstacle region l_b (pink) and free space l_f (green).

the range of the infrastructure. For larger areas, the indexing method can be altered to follow a space-filling curve such as the Hilbert curve [41]. Such curves have the property that the average in-memory distance between neighboring elements is small, making access more likely to fall into cached regions of a map.

The Lidar provides direct boundary points of obstacles when offset by p, but at 2000 points/s incrementally storing all points becomes prohibitive. In addition, information is lost if we store only the boundary point originating from the Lidar. The true measurement is that there is no obstacle between pose p up to the boundary point, which is done 360 times at 1° increments. With this interpretation, the Lidar does not discover obstacles, but free space. So a sensible initial state would be $C_f = \emptyset$, $C_o = C$. Lidar measurements take the form $l = (p, l_d)$ with $l \in L = C_f \times (\mathbb{R} \times \mathbb{R})^{360}$, where l_d are the distance measurements converted to absolute positions in C. As in Figure 3.5, from l we determine $l_b, l_f \in \mathbb{P}(S)$ by s, with l_b the cells on the boundary and l_f the internal free space. The combined size of l_b and l_f has a constant upper bound determined by the resolution and maximum range of the Lidar.

The measurements are not perfect and suffer from noise, and error points on the boundary will not have a well-determined position so the region becomes uncertain. The minimum grid size depends on the achieved positioning accuracy. For now the assumption is made to have a grid size of 10 cm, which works reliably with an average positioning accuracy better than 5 cm. The actual value to store inside of a grid cell must somehow include the reliability of a measurement as a metric for comparison. As the map changes over time and updates may arrive out-of-order from several sources, the age of the measurement is also relevant. The number of detected points in a grid cell is inversely proportional to the distance of the robot, so the basic formula for reliability $r \in \mathbb{R}$ will have the form $r = \frac{n}{\epsilon d}$, where n is the number of detected points, d the centerpoint distance from the lidar and the grid cell, and ϵ a nonlinear error derived from the current estimated position error that requires tuning. The space map is $W_S : S \to \mathbb{R} \times \mathbb{R}$, with $W_S(x, y) = (t, r)$ with timestamp t and reliability r. A cell is likely in C_f if it is within l_f, but cells in l_b may also belong to C_f. The merging of Figure 3.5 with the stored values of the existing map is then as such:

1. Any cell in l_f will now belong to C_f, but only if t is newer than the stored value.
2. Any cell in l_b with higher r than the stored value will replace the stored value.

3. Any cell in l_b with nonzero reliability replaces a sufficiently old stored value.

Initially all cells are in C_o with zero reliability.

This provides mechanisms for detecting both new free space (1) and new obstacles (2), while maintaining measurements of higher significance. And (3) ensures that obstacles with high reliability due to statistical variance are eventually re-evaluated and do not get stuck forever, but only when new data is available. Moreover, these updates preserve the properties of idempotence and commutativity, meaning the updates can be applied out-of-order and multiple times without different results, as long as all old updates are applied before the old-age threshold used in (3) is met. These properties are significant for systems with multiple distributed contributors to the datastructure and ensure a stable system.

3.2.2.2 Entity map

The entity map W_E can be seen as an object-position database. For various purposes it is necessary to indicate labeled objects from P in the space C with exact coordinates (x, y) or pose $p = (x, y, \theta)$. Then, $W_E : P \to C \times \mathbb{R}$ with $W_E(a) = (p, t)$ maps label a to its known pose p at time t. This is used for both communicating accurate location and tracking of objects through the scene. Such objects can be waypoints, locations of charging bases, other robots, and even dynamic obstacles. As the purpose is rather generic, all of the following operations must be possible: insert, update, delete, key-based search, neighbor search, and region queries.

Allowing all operations simultaneously in a distributed manner requires mutual exclusion, which involves proofs of consistency for such distributed systems. Instead, each object is considered to have an owner who is the only agent permitted to perform updates. During an insert, the owner field is indicated. Another agent may perform an insert on behalf of the owner, and another agent may delete the object again on behalf of the owner. Updates of objects that occur after they are deleted are ignored. Inserts are planned outside of the scope of this data structure, which can easily be guaranteed when planned centrally. When done in a distributed manner this puts the burden of proof of consistency on that implementation. Query operations depend on the state of the agents. But as all updates have timestamps, only the most recent data is used and all queries will use only recent information. Consistency in such a system is obvious as long as all operations arrive.

Labels of objects follow a structure, where for instance `relative_3_4` represents the relative pose of robot number 3 as seen by robot number 4. This structure allows selection during search, but the exact structure is not specified in this design.

In terms of data structures, a 2-dimensional KD-tree can take care of the query operations of neighbor search and region queries in $O(\log n)$ and $O(k \log n)$ respectively. The pose is stored as extra information in each cell, indexing only occurs on the x, y-values. Label-based lookups can be realized in $O(1)$ time by simultaneously maintaining a hash table where labels are the keys. The hash table contains a reference to a leaf in the KD-tree, which makes updates of non-position fields in $O(1)$ time. Inserts, deletes, and updates of position are bounded by the KD-tree complexity, where all of these operations take $O(\log n)$ time.

3.2.2.3 Navigation graph

Navigation in the space map W_S can be done directly with search algorithms. However as the map has a lot of open space, traditional grid-based search strategies are slow, especially at high resolution and with few obstacles. A good approach is the Ariadne's Clew Algorithm [49], which alternates a directional search with local exploration near detected obstacles and backtracking. But there is no guarantee that the map is static during execution of the search, so computations have to be fast as they would block updates, and copying W_S is prohibitive. So any evaluation of a significant number of grid cells can cause problems, which bounds the search region.

The navigation graph $W_N = (V, E)$ is a summary of the space C_f that allows partitioning of such search algorithms into manageable parts. W_N is the result of a successful online cellular decomposition of the space. The details of the construction of this graph are abstracted from in the description of this data structure. A further explanation to its use can be found in Section 3.3.3. Nodes from V are identified by a unique number and represent such a cell, or region, and edges from E indicate reachability. Each node is associated with a point in C in the region, which can be used for construction, and a timestamp.

Updates to this data structure must be carefully managed when multiple agents are involved. The easiest way to restrict these is to periodically recompute the navigation graph entirely, completely replacing W_N. But this involves a complete scan of W_S, which is prohibitive. If mutual exclusion can be guaranteed any agent can ensure that the invariants are preserved. Another solution exists if the properties of commutativity and idempotence can be achieved with certain updates. Such a suitable *commutative replicated data type* (CRDT) exists, as described in [65], and will be used for W_N.

A CDRT only guarantees consistency of the data structure, and not of additional invariants. In particular it does not prevent the addition of duplicate cells for the same region if a robot did not receive that cell yet. This is made less likely if such updates are derived exclusively from local updates of l_b from Figure 3.5. But a solution is still needed, which involves a method to clean up the graph. The advantage of a geometric cellular decomposition is that such duplicate vertices will have very similar geometric distance. Therefore, any vertex added by a robot will be removed by that same robot if at some point a near enough vertex is introduced that has an older timestamp. This guarantees that only the earliest node is eventually present, and that not both are removed. Such a check can be made whenever a new vertex is received from other robots.

For the data structure of W_N an adjacency list representation is used, where vertices maintain a list of their neighboring edges. The position and timestamp associated with each vertex is stored in the entity map W_E. The neighbor lists are bounded by constant size due to cells either splitting or merging, making add and delete operations $O(1)$, so those operations are bounded by W_E at $O(\log n)$ time. In addition, all edges are also stored in a hash table which allows $O(1)$ adjacency queries. Finding the closest node on this graph is an operation on W_E and takes $O(\log n)$ time.

3.2.2.4 Coverage map

Multiple robots will be adding to the completion state of the coverage algorithms. Keeping track of the exact regions that have been covered is impossible due to positioning error. Proposed is a stochastic method for tracking coverage, where for every position in C_f we have a probability of coverage. This probability is set by an estimate of the positioning error.

Following the same reasoning as for the space map W_S, the coverage map W_C has S as domain, quantized into a 2-dimensional grid as $W_S : S \to \mathbb{R}$. Coverage updates compute the maximum of the past cell in S with the probability of the update region in the center of this cell. Commutativity and idempotency of the maximum function ensures that W_C remains consistent under distributed updates.

Figure 3.6 compares perfect positioning (a) to a linearly interpolated model (b) to a normally distributed position error (c). The precise error distribution is of significance, and from (b) is obvious that a single-slope linear interpolation is an accurate fit for typical situations. Note that the linear interpolation is significantly bounded in the number of grid cells as the tail of the distributions is cut off. This ensures compact $O(1)$ update size. When position error exceeds beyond the robot radius, the maximal probability is lowered but the robot still contributes to the coverage progress with whatever is possible.

The shape of the update region is also relevant. A rectangular shape fit over the current position and heading angle as in Figure 3.7 does not account for change in orientation in the near past. The distance between the shapes is exaggerated and illustrates potential lack of overlap when turning angles are sharp. This effect is reduced by widening the tail of the update region slightly over distance, as in the trapezoidal representation from Figure 3.7. Coverage probability in the update region decreases slightly with length along the tail and significantly with deviation from the linear projected path. Even then the update region should always be longer than the traversed distance. High frequencies of such internal updates with small traversed distance shortens the interval and makes it unlikely that a grid point is included. This limits the interval to a minimum length of the maximal distance through a grid cell, at $\sqrt{2} \times 0.1$ m.

An alternative would be to use the centerline between the current pose and the previous pose for which this region was computed, instead of the current orientation. This ensures that all path segments join up together without gaps. But the current brush orientation is very certain and would not be represented by such a method, as the brush is off-center and significantly affected by orientation.

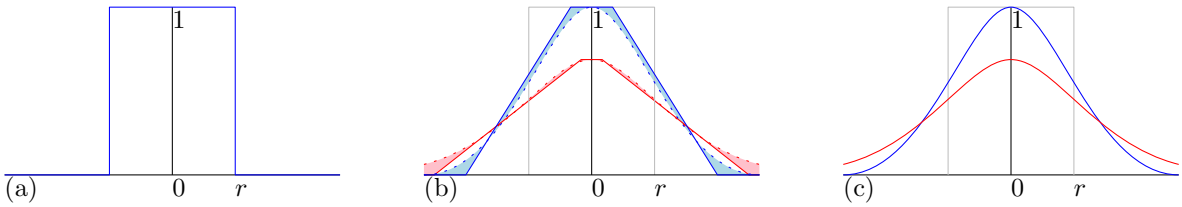

Figure 3.6: Coverage probability functions from robot with radius r and center 0, (a) with no error, (b) with linear approximation at $\epsilon < r$ (blue) and $\epsilon > r$ (red) and (c) with true distribution.

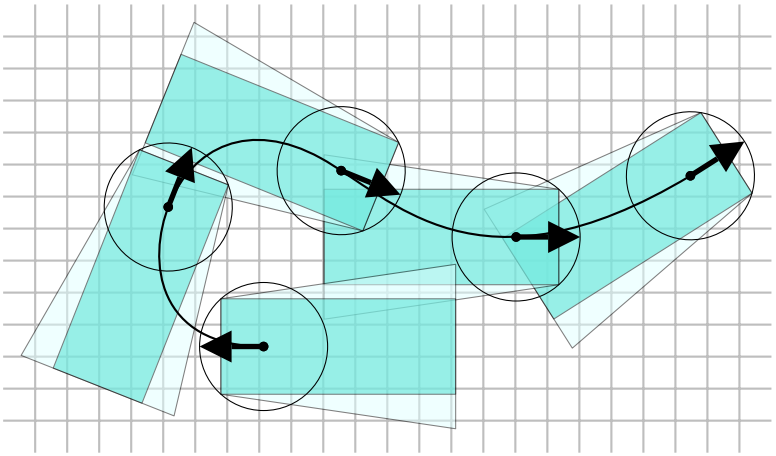

Figure 3.7: Update regions of the coverage map when navigating.

Readily covered regions that become obstructed will preserve their coverage state and be unused until this is freed up later. Newly uncovered regions will be immediately available for coverage path planning algorithms as obstacles disappear. To restart coverage path planning, W_C only needs to be reinitialized with zero probability of coverage. This could be done selectively, for instance for spot cleaning. Shortly after such a reinitialization, updates from the near past may still arrive and be processed. This is deemed acceptable due to the recency of coverage.

3.2.3 Scheduling

High-quality results can only be obtained when computations and measurements are recent and in-phase. Latency is a problem and needs to be well-understood and compensated for. Otherwise the system may overactuate and become unstable. Since performance is a leading metric, now follows a global picture of execution priority in the system.

The part that does direct control of the actuation of the robot and its pose estimate must satisfy a hard real-time requirement, or the sensor fusion algorithm breaks. This implies that the controller and all its sensor dependencies have upper bounds on execution time, and is isolated from the rest of the system by asynchronous interfaces. The rest of the system is more opportunistic, but a soft real-time deadline is still in effect as the planner period repeats every 0.1 s. The processing of network data in the controller precedes the busy period of the planner, so these must also not take significant amounts of time. New sensor data is processed with higher priority than the planner, as communicated system state may not incur too much latency. If a planner deadline is missed, the controller may actuate past the partially planned path which pauses the robot until the planner has a chance to catch up. This may result in stuttering motions and is definitely problematic for the final performance. If the soft planner deadline is missed, but not significantly, the controller will have sufficient amounts of planned path available to continue, but with a reduced feedback loop and thus lesser performance. The manager however is not at all interested in performance and observes the lowest-priority task. It may be preempted by exceptions in the planner module however.

Details of how latency is reduced depends strongly on the scheduling that is possible from different component performance. Individual schedules will be examined as part of the implementation, when the specific capabilities and system limits are discovered.

3.3 Algorithms

Several algorithmic challenges need special attention. These are described here because they have an impact on the high-level behavior of the robot and there is no obvious best method to realize them. Most challenges are with motion planning, including navigation, coverage path planning, and trajectory tracking. But localization also sees optimizations, such as Spring UWB localization. The summary analysis in Section 3.3.6 describes how they combine and contribute to the system as a whole.

3.3.1 Spring UWB ranging localization

Complete localization functions are provided by the Pozyx system, but it has two problems:

1. By the time the result of a full TDoA (time-difference of arival) localization is collected a significant amount of time has passed, resulting in significant latency.
2. By performing a full localization, the resulting error distribution loses its Gaussian characteristics, which is problematic for subsequent filtering.

So a solution is derived by performing incremental two-way ranging with individual anchors.

The heart of the algorithm is the SPRING position:

$$\textsc{Spring}(\overline{p}, \overline{u_i}, d) = \overline{u_i} + d \cdot \frac{\overline{p} - \overline{u_i}}{|\overline{p} - \overline{u_i}|},$$

for the last-known position estimate $\overline{p}$ and anchor position $\overline{u_i}$ with measured distance in the plane d. Note that the range between anchor and tag is converted to distance in the plane, as an anchor may be positioned out-of-plane. The result represents a new position adjusted in the direction of the anchor, as if the range measurement were completely accurate. The SPRING position directly uses the filter's most recent estimate $\overline{p}$ to correct, and Gaussian noise is completely preserved in the direction $\overline{p} - \overline{u_i}$. This way, the filter can extract maximum information from the measurement for estimating the true absolute pose.

In the system, the SPRING position is computed immediately for each new range measurement that arrives. But its accuracy depends on the accuracy of a prior pose, without which the system can not converge. So initially when the robot is stationary, full localization is performed instead. And only when the localization measurement becomes consistent the SPRING update can take over. The position keeps converging as long as sufficient measurements arrive.

It is useful to have subsequent range measurements orthogonal to each other. But different positions have different ideal orders, and choosing a fixed order may lead to a worst case. To avoid a potential worst-case, a random order of evaluation of the anchors is chosen. This favors most positions in the scene.

In conclusion, latency is significantly reduced by only retrieving a single range measurement, which is better capable of correcting the robot position and reducing incremental angular drift that results in Abbe error. This method preserves the Gaussian characteristic from the measurement uncertainty in one dimension, and orthogonality of measurements ensures that this gets distributed over both dimensions in sufficiently short time. The lack of aggregation and subsequent increase in noise for point results may seem a disadvantage, but the filter is especially well-equipped for this data. More frequent noisy estimates with a closer match in specified properties are a better fit for making instantaneous estimates.

3.3.2 Local potential field trajectory tracking

Because of position error there is a mismatch between the planned path and the true trajectory. Following the planned path exactly will lead to an offset due to incremental error. The position estimate will oscilate around the true position, and a method is needed to move toward the optimum. To realize this, a potential field is constructed around the planned path.

There is no need to supply a complete planned path for computing a local potential field, as the robot has a maximum velocity and only the local region is relevant. So the length of the path can be limited to a quantized constant-sized segment that is at least as long as the maximum traveled distance over a timestep including maximal position error. Each setpoint on this path contains a position and target velocity. The heading angle is derived from subsequent segments.

Such a potential field would ideally have the following properties:

1. The potential field decreases along the path by the points of closest distance to that path.
2. The potential field increases as distance from the path increases.
3. The gradient of descent corresponds with the desired velocity.
4. The angular change during descent must be realistic.

The minimum of this field obviously lies at the end of the path segment, and all points closest to the end of this segment have identical value which implies zero velocity. Such a potential field is defined on the 3-dimensional pose $p = (x, y, \theta)$, but the discontinuity in angle at the setpoints will always break property (4). This can be solved by simplifying the field to a 2-dimensional one by allowing velocity to decrease below (3) to match the planned rotation. Further discontinuity exists as some points in the field may be equidistant to several points on the path. So the field is defined as a vector field, of which the gradient will always lead further along the path. These properties do not uniquely define the field, as the proportional priorities of (1) and (2) are undefined.

To define the field, two configuration parameters are introduced. Angle a indicates the maximum angular deviation from the setpoint orientation, and r is the radius of a circle arc, which sets the desired angular velocity to perform path corrections at. In this way, deviations in heading angle within the potential field are bounded, which avoids unintended turns. Corrections close to the path will never be excessive, as the angular velocity for path corrections is honored. So far, this only defines a potential field under a straight line, as illustrated in the blue region of Figure 3.8.

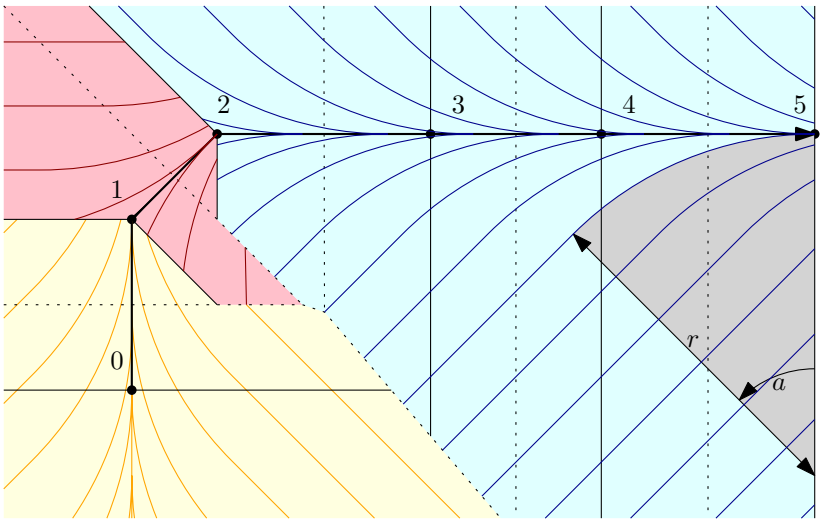

Figure 3.8: Potential field around 6 setpoints. Dotted lines indicate Voronoi boundaries. Each colored region observes a distinct potential field. The gray area illustrates the configuration parameters r and a. Colored lines show the direction of the potential field, decreasing along the path toward the right.

Segments of the path may observe a different setpoint direction, so the region must be segmented. The closest setpoint to the robot determines that near point on the path, as segmented by the dotted Voronoi boundaries of Figure 3.8. Note that the closest point on the path from the robot's position can still be in a different Voronoi cell, and this is ignored for simplicity. For some robot positions in the Voronoi cell, the closest point on the path has already passed the setpoint, for instance the pink region of cell 1 and the blue region of cell 2 in Figure 3.8. In those cases, the potential field is determined by the next setpoint instead. To ensure that the potential field brings the robot closer to the path, this boundary is determined by the direction orthogonal to the preceding segment for the exterior angle, and the direction orthogonal to the following segment for the interior angle. As this decision is taken after the closest setpoint is selected, these boundaries stop at the Voronoi boundary. Discontinuities in regions such as the right most part of cell 1 in Figure 3.8 are not a problem as the robot transitions through these while increasing distance along the path and decreasing distance from the path.

A common problem with potential field methods is that they suffer from local minima. This is usually a problem with more complex environments, where potential fields are used as a global navigation method. Since the local goal is to follow the path and the potential is guaranteed to decrease along this path, this problem is avoided. Note that the field cannot be continuous where two segments at different orientations meet. The strong directionality of the field ensures that no local minima can occur, and this is good enough. For this system, global navigation is handled on a higher level that does not suffer from local observations.

An extension can also include a repellant force for nearby obstacles, to realize collision avoidance. But as collision avoidance is already taken care of at the planning stage, embedding it in the controller is considered an extra complication. The advantage would be a potentially much closer proximity to obstacles without risk of collisions, as the controller responds more rapidly.

The size of the path segment is bounded by a constant. This ensures that determining the closest point and the configuration of the local potential field is done in $O(1)$ time. The resulting vector is a simple relation between the distance of the infinite line through the segment, and the parameters a and r, also determined in $O(1)$ time. Potential fields can be used like this because the region is free of obstacles. The shape of the potential field will give a smooth trajectory towards the planned path, with properties that deliberately bound oscillations in orientation under uncertainty. A potential field is defined for the whole surroundings, so any possible position that is determined will result in the robot following the path. The distance from this path segment should never become too great as a path is always determined around the robot's current position.

3.3.3 Incremental online navigation

Recall from Section 2.2.4 that many methods exist to perform navigation. In the absence of obstacles, a straight line can be drawn from start to destination. But in an indoor situation, obstacles are plentiful and the environment has similarities with a maze, where dead ends should be avoided. In such a situation, exploration of the environment becomes necessary, and with a grid-based space map W_S, the search graph becomes rather big. A popular algorithm for navigation in high-dimensional spaces is the Ariadne's Clew Algorithm (ACA) [49], which alternates so-called search and explore phases, during which it places landmarks that enable backtracking. A strategy better suited to 2-dimensional pathfinding is to perform a standard graph search, such as A*. But A* has no natural bound in the search range and may end up stuck in a dead area, consuming too much time in certain cases. The D* algorithm [70] is an incremental search algorithm that can quickly replan the path given a change in the environment, reusing the previous found path to speed up the search. So-called anytime algorithms begin with a suboptimal path and provide incrementally better versions as time allows. For instance with anytime dynamic A* (AD*) [42], which is also suitable for dynamically changing environments. AD* is well-suited for complex navigation and are quite efficient due to partial reuse of previous plans, but would still involve extensive evaluation of elements in W_S when distance between them is large.

In Section 3.2.2.3 the navigation graph W_N is introduced, which specifies relations between a set of nodes that indicate reachability. Each such a node has an associated position, and every position in C_f can observe at least one node without obstacles. Therefore, the act of navigation from point a to b can be decomposed into parts. If a direct route is possible between a and b, that is chosen immediately, otherwise the navigation graph is used. First, the node v_a from W_N corresponding to the current region is selected by a nearest-neighbor search in W_E. The same is done with the node v_b corresponding to the destination, which gives the start and end positions in the navigation graph. Then, a shortest-path algorithm such as Dijkstra's algorithm with weights corresponding to distances is used to find a sequence of nodes $v_0, \ldots, v_{n-1}$. And in execution, the prior AD* navigation algorithm is partitioned to navigate from a to v_1. After line-of-sight is established with v_1 it continues with the current position to v_2, iterating until finally observing v_{n-1}. Immediately after observing v_{n-1} it navigates directly to b. By skipping v_0 and not following the path entirely toward the destination, the robot will take the shortest route around obstacles in the regions that are governed by a node. And because the AD* algorithm is used, it is robust against many dynamic environmental changes, even if W_N becomes outdated.

The final ingredient is online construction of the navigation graph, which will only briefly be touched here. The navigation graph represents the generalized Voronoi diagram (GVD) as described in [45]. The paper also shows how a GVD can be constructed directly from laser rangefinder data. But this is not an online algorithm and does not deal with dynamic changes in the data. [1] uses a Boustrophedon cellular decomposition to obtain so-called vast and narrow regions. This method determines so-called split nodes along a chosen cardinal direction for decomposition. Furthermore, the connectivity graph resulting from a Boustrophedon decomposition corresponds to a GVD. This relation is interesting as the detection of a graph split in the Boustrophedon decomposition is as simple as finding an obstacle closer to the robot along the decomposition direction than an existing split node. Such a technique can be used to easily detect a change in the cellular decomposition, which subsequently translates to a local update of the GVD. The data structure of W_N described in Section 3.2.2.3 ensures several properties relevant for distributed updates of this graph.

3.3.4 Incremental contours coverage path planning

The goal for coverage path planning is to ensure that robots visit everywhere they can, without skipping any area. The W_C data structure is continuously incremented through movement in C_f, and all grid cells of S are available for storing progress. But when W_C is queried, it is masked by the coverage mask C_M, where all entries not in C_M have been covered with probability 1. C_M represents the free area in W_S, treated with the morphological operation of erosion for 1 grid cell (0.1 m). Erosion of the coverage area is necessary as the robot can not reach close to obstacles and gaps would be left, which would fail the completion criteria. The mask C_M can also be used to resrict the act of coverage path planning within a certain region.

Now follows a routine to incrementally generate paths. The input is a pose $p = (x, y, \theta)$, a coverage probability threshold, an offset distance, and a boolean to indicate clockwise or counterclockwise direction. The coverage probability threshold corresponds to a contour line in W_C, which can be found by searching the grid for two neighboring cells that have a higher and lower coverage probability than the threshold. The algorithm first searches orthogonal to the direction of movement of the pose p, to its left for a clockwise direction or its right for a counterclockwise direction. When found, a marching squares method of searching the next points of the contour from the grid cells in W_C is used, which simply finds all neighboring cells that preserve the transition. An accurate position on the contour can be determined with linear interpolation from the coverage probabilities of the cells. If no such point is found, the path generating routine errors. Instead of tracing the whole contour, it is traced back until slightly before the pose and forward only a few positions. The length is determined by the maximum velocity of the robot and the current measurement error, being long enough to last at least until the next iteration. From the resulting contour segment, the offset setpoints are computed that make up the output path, which is a fixed offset distance to the right (clockwise) or left (counterclockwise) of this contour.

In order to cover the area, the robot is first given the assignment to navigate forward until it is an offset distance away from the border of the masked W_C. After arriving it performs an angled turn to align with the edge in the desired clockwise or counterclockwise direction. Then the path generating routine is called, iteratively, which provides continuously the next setpoints for coverage path planning. Upon error of this routine, a bounded nearest-neighbor

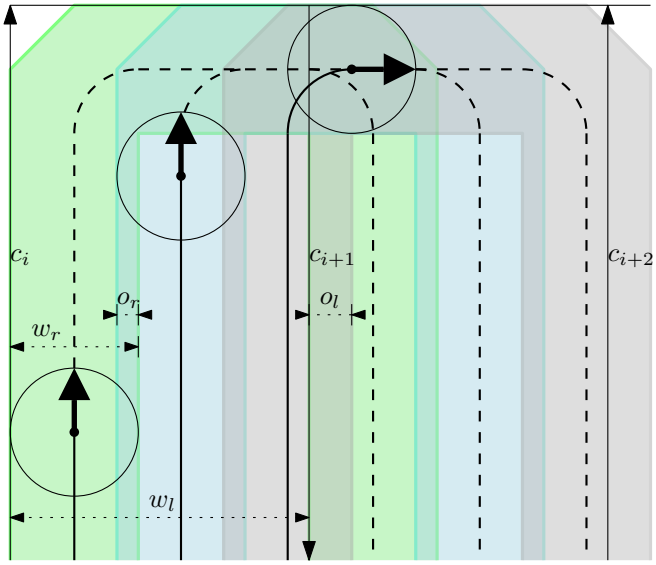

Figure 3.9: Multiple agents covering a region in formation and then reorganizing for the next lane.

search is done on W_C for areas that are still under the threshold. If the search fails or reaches its bounds, the algorithm completes. Otherwise, the robot navigates into the found area and repeats from the routine described in this paragraph.

The robot will naturally spiral around the region until it reaches the center. It ensures that the overlap is exactly as much as required depending on the statistical positioning error. If any gaps do occur, the final search will take care of those. If the region for the search is not bounded, the algorithm may spend a lot of time searching inefficiently. The algorithm can simply be rerun to cover more area later even with a lower threshold, so such a bound is sensible.

All operations are designed to take minimal time. The most heavy operation is likely the morphological operation to compute the mask of C_M, but this can be done in a lazy fashion, distributed as $O(1)$ operations over the course of the algorithm, as the robot evaluates those cells anyway. Then the search algorithm for low-probability covered cells remains. But as the search area is bounded in size, this is also prevented from becoming a performance problem in the system.

3.3.5 Offset-based formation control

Formation control has the potential to achieve a speedup in coverage time by reducing the amount of overlap during coverage path planning. This can result in situations such as the lane-based coverage demonstrated in Figure 3.9. This is possible because relative distance measurements between robots are more accurate than absolute localization in the space ($o_r < o_l$). With three robots in formation, the effective lane width $w_l = 3w_r - 2o_r - o_l$ is bigger than $3w_r - 3o_l$), when all three robots pass independently.

Any coverage algorithm supports incidental platooning by reconfiguration of the effective coverage width of the robot, and an offset to position the first robot. In an alternating

lane-based coverage algorithm, alternating lanes observe different directions. The first robot will therefore leave a gap in one direction, expecting the other robots to fill this up. But this may not happen if the position accuracy suddenly decreases, potentially leaving the gap uncovered. With platooning for lane-based coverage where direction is essentially inverted, this will always be a risk. With the contour coverage algorithm from Section 3.3.4, the front robot will always follow the outermost contour and every robot following can adjust to W_C as they pass to guarantee a true covered area.

In addition, when the area is narrowed by an obstacle, following robots may reduce their offset from their predecessor to follow in line. With lane-based coverage in the wrong direction, this would result in more repeated areas, which is inefficient. In contour-based coverage such a platoon would always collapse toward the contour, leaving one large uncovered area which is subsequently efficiently filled. Due to this changing offset and collapsing of the width, the method of movement is just as free as single-robot movement in C_f.

3.3.6 Summary analysis

Not all functions of localization and motion planning have been described by the above algorithms. Mapping is the result of continuous Lidar obstacle regions. Collision avoidance is a result of pathfinding in C_f and a deformable virtual zone from Lidar range measurements. Sensor fusion is the result of a Kalman observer on several sensors simultaneously. And significantly more extensive methods are certainly possible.

All the algorithms described in this section work together. Each algorithm exists to solve specific problems and not all simultaneously. Algorithms and data structures, as well as the sensors are more simple because dependencies abstract away from essential problems. The the combination of trajectory tracking with potential fields, contour-based coverage path planning and offset-based formation control establish behavior that should be quite efficient and robust. Trajectory tracking with potential fields and the UWB spring localization algorithm complement each other to reduce position error, where tracking limits abrupt change in heading angle while the spring algorithm corrects for true position. And the navigation algorithm helps manage complex scenes for this embedded solution.

As these algorithms execute in an unpredictable environment, their success is hard to guarantee. Las Vegas algorithms always complete, but their runtime is unpredictable. Monte Carlo algorithms complete in deterministic time, but they may not succeed. Repetition of a Monte Carlo algorithm until it succeeds turns it into a Las Vegas algorithm. However, the best behavior of a system is not always realized by blindly retrying the task that has been given. Therefore all tasks that the `planner` module executes are Monte Carlo algorithms, which the above motion planning algorithms are.

This leaves the `manager` module to implement derived Las Vegas algorithms, strategically scheduling several different Monte Carlo algorithms in repetition. Such strategies are beyond the scope of this design. If the expected completion time and success probabilities of all Monte Carlo algorithms in a Las Vegas schedule are reasonably known, then the expected run time of the derived Las Vegas algorithm can be analyzed, even under uncertainty.

Chapter 4

Implementation

The purpose of the implementation is to ensure that all components fit together well. Total system performance is optimized, and each component has its individual unique contributions. The resulting system follows the hardware and software architectures defined in Chapter 3. For software, the successful implementation of one module results in some measurable performance. This enables verification on the module-level. The interface and performance characteristics of a module guides the implementation of its depending parts. A consequence is that modules without further dependencies are first to be implemented.

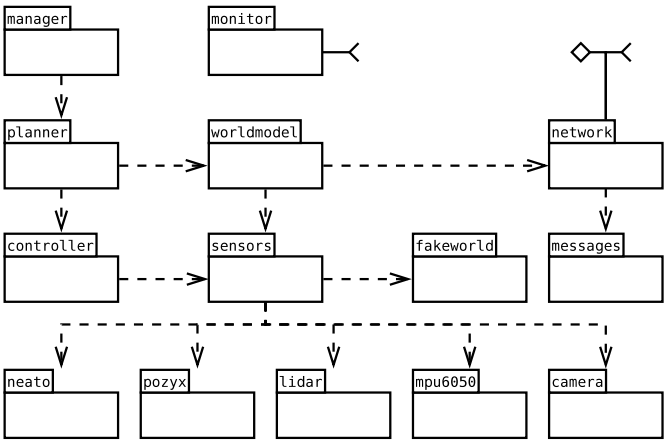

Figure 4.1: Dependency graph of system modules.

Figure 4.1 shows these dependencies between software modules. A natural order of implementation can be derived as follows:

1. Assembly of the supporting electronics
2. Low-level hardware drivers and processing (`neato`, `lidar`, `pozyx`, `mpu6050`)
3. Virtual sensors and physics for simulation (`fakeworld`)
4. Sensor abstraction (`sensors`)
5. Control algorithms (`controller`)

6. Communication (`messages`, `network`)
7. High-level abstraction of the world and its debugging system (`worldmodel`, `monitor`)
8. Algorithms to execute individual tasks (`planner`)
9. High-level orchestration (`manager`)

A good design minimizes invasive changes but cannot anticipate everything. A bottom-up approach ensures known properties at each phase of the implementation. This mitigates risk and helps optimization of the whole stack. Only after all individual modules have been implemented, whole-system verification can take place. This chapter will honor the above order and group these items into sections of successively higher-level behavior.

4.1 Hardware integration

Several issues were discovered during the integration of the hardware. Their solutions will be presented here. The functional decomposition from Figure 4.2 will lead this section. The specific realization is a one-off prototype assembled from parts, purely to reduce the project's duration. A single circuit board design that is plug-in compatible with the platform, the Raspberry Pi and Pozyx tag is certainly feasible and even desired when producing several units. However only the prototype is progressively documented here.

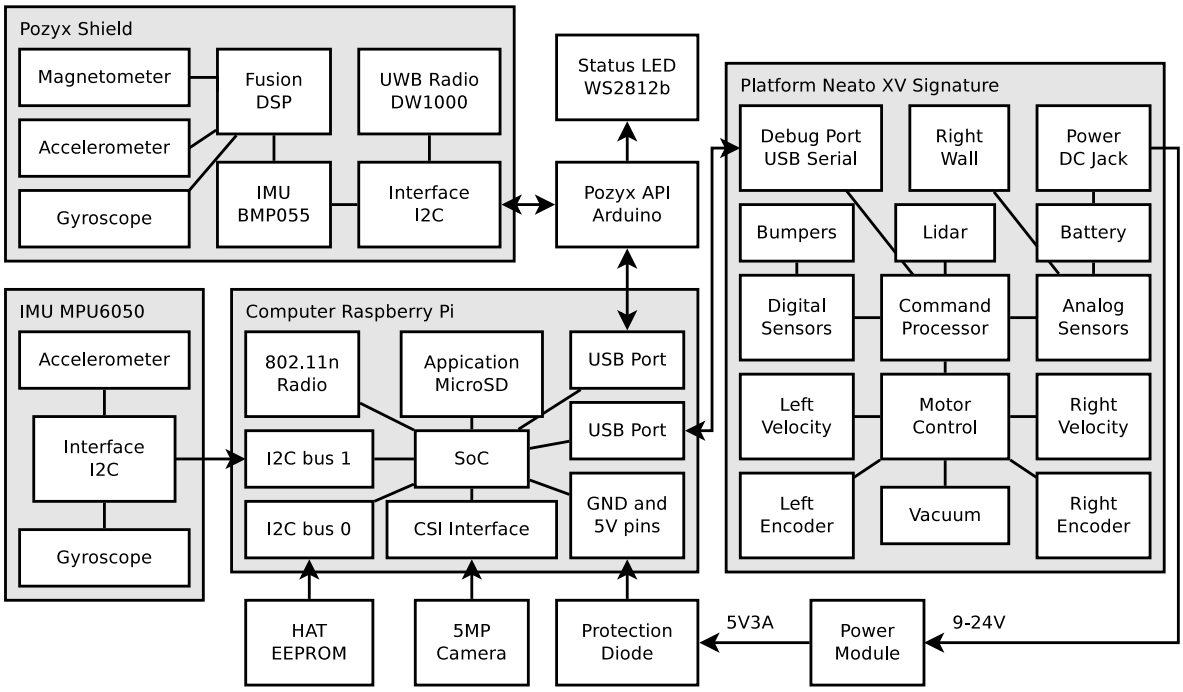

Figure 4.2: Functional block-level diagram of all related physical hardware.

4.1.1 Modifications

In some situations the hardware does not exactly suit our application and specific modifications are required. Modifications to parts from the initial state after purchase are described here. These are designed to be as unobtrusive as possible. This section does not describe the two circuit boards that connect most of the electronics together. As these are nontrivial parts, they are described in Section 4.1.2.

The power source for the whole system is the platform's (Neato XV Signature) on-board 14.4 V battery. However, the only physically exposed connections on the platform are its charge bumpers, a DC jack for power input, and a USB slave port. The platform has a 5 V internal supply, mainly to power the Lidar. But the Lidar is sensitive to voltage fluctuations and the 5 V supply cannot provide sufficient current for all electronics. Besides, the USB slave port is isolated from the internal 5 V supply of the platform. Charging can be done through the charge bumpers or the DC jack. However both contain a protection diode that prevents current from going out. To get power out of the system, one of these has to be bypassed. But the charge bumpers are easily shorted and removal of that protection diode is likely to result in a fault eventually. So the protection diode for the DC jack (D15 in Figure 4.3) is removed and bridged instead. The DC jack can still fulfill its original purpose of powering the platform for testing, but now care has to be taken with the polarity. After modification the DC jack can supply power from the battery, with other means of protection still in place. This can then be fed to an auxiliary 5 V DC buck converter to power the computer, which has power circuitry specially designed to power all other 3.3 V and 5 V electronics.

The Pozyx shield is designed to be interfaced by I2C bus from an Arduino-compatible system. Pozyx advertises an USB-port to interface with the computer, but as of this writing that function is not available. For the chosen Raspberry Pi computer no compatible driver exists, and sharing I2C bus 1 is not satisfactory due to real-time constraints. As a solution, an Arduino-compatible microcontroller board connected over USB will be between the Pozyx Shield and the computer. Specifically, we choose an Arduino Pro Mini module with an ATMega328 MCU running at 16 MHz, connected via a USB-to-UART adapter to a USB port of the Raspberry Pi. This microcontroller is then isolated at 5 V and can then run the Pozyx driver without modification. Software on the microcontroller will transform requests from the computer into commands for the Pozyx shield. Results will be reported back over the

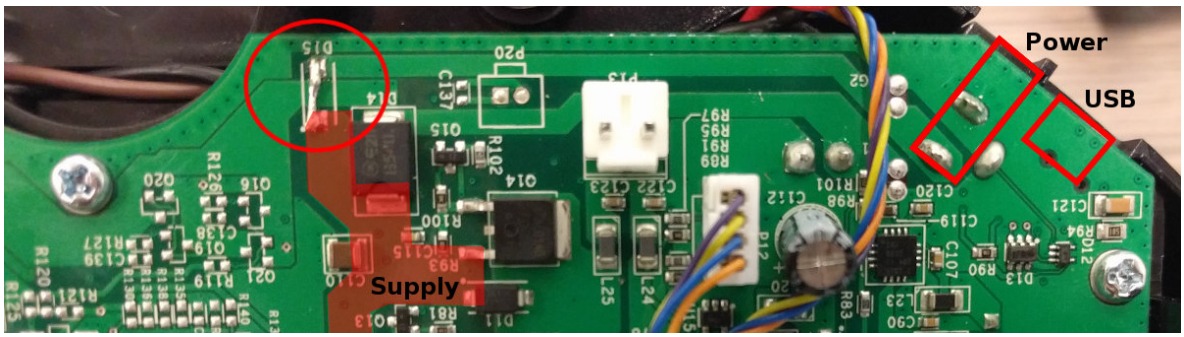

Figure 4.3: Platform circuit board, where D15 is replaced by a jumper wire. The internal power supply plane is marked red. Power and USB connectors are on the right.

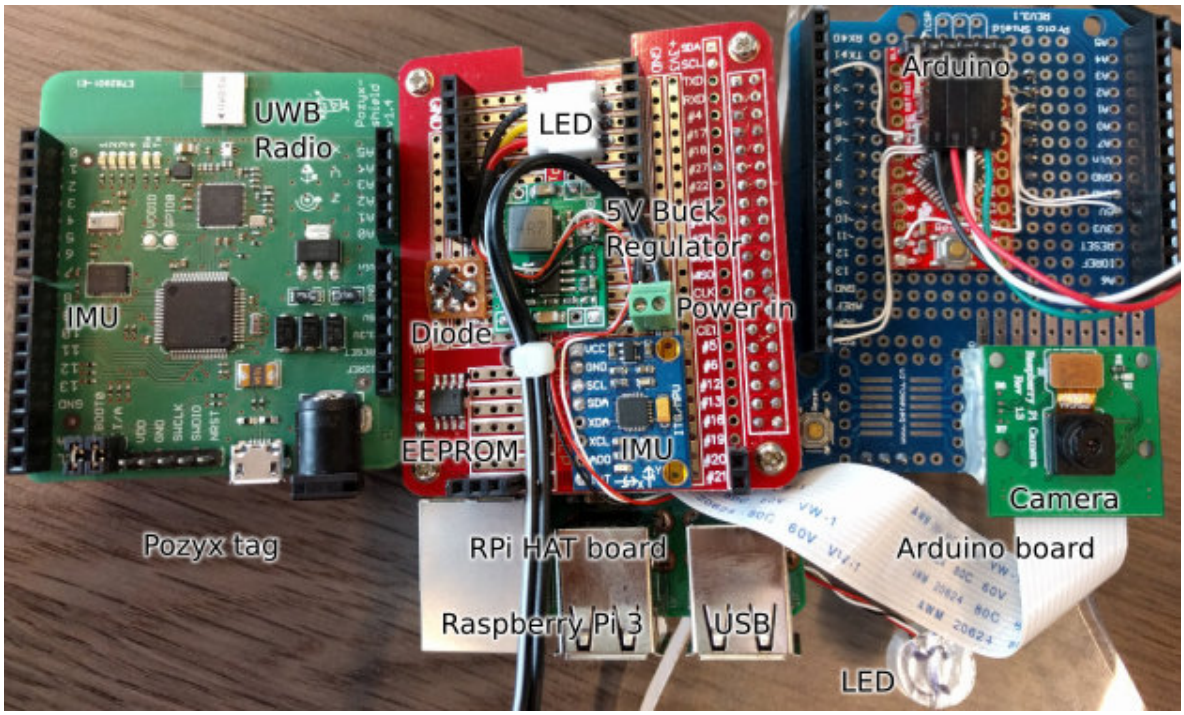

Figure 4.4: Circuit boards and components to realize the electronics interface.

virtual serial port over USB. The design of the software running on the MCU will be outlined in Section 4.1.5.

4.1.2 Adapter boards

Figure 4.4 shows the individual circuit boards with labeled components. These implement the functionality from Section 3.1.3 as is listed on page 36.

The middle red HAT board has the correct footprint to connect to the Raspberry Pi 3 computer board, to which it is currently attached. It is a generic prototyping circuit board on which the labeled electronics has been placed. The blue IMU is the MPU6050, and has been positioned in the exact rotational center of the robot. The green buck regulator contains a power management IC and the power input next to it is connected to the platform's DC jack connector. The output is fed through the brown diode board which is designed to have negligible forward voltage as per the schematic. After the diode it directly powers the computer. The LED connector goes toward the WS2812b RGB status LED that is mounted in the light guide on the bottom of the picture. The EEPROM is used by the Raspberry Pi HAT specification to identify the add-on board and has been programmed according to the specification.

The left green board is the Pozyx tag, to which no modifications have been made. It plugs straight into the top of the right blue board, which is only meant to mate to the pins of the Pozyx board for which no space was left on the red board. The black pins on the middle red board connect them mechanically. On it, the red Arduino board is mounted, and the wires of

its 5 V UART connection go to a USB serial adapter which will be plugged into the USB port of the computer.

The green camera module is mounted to the right blue board to face the ceiling. It is connected with a 14-pin flex cable directly to the Raspberry Pi's CSI connector. It can also be mounted to face the front of the robot.

4.1.3 IMU interface

Many drivers and modes are available to interface the Invensense MPU6050, and many more configuration parameters are present. One special mode of this chip is that it supports on-board sensor fusion of the accelerometer and gyroscope, which is why it was selected for this project. When configured for sensor fusion it can reduce the effect of orientation drift to a significant degree. But to enable this the digital signal processor (DSP) has to be configured, and some internal restrictions subsequently prevent the system from running at high rates. The highest-achievable sample rate with the DSP enabled is 200 Hz, but the datasheet itself notes that the 100 Hz configuration is recommended as it is significantly more stable. Most drivers that exist use the DSP to get higher accuracy, at the expense of sample rates, latency, and bandwidth. The paper of [6] uses the MPU6050's DSP configuration at 200 Hz for their comparison. But this configuration suffers from significant latency due to the signal processing involved. There are essentially two mutually exclusive directions for optimization. If low-latency is desired, then no processing may take place. If high integration accuracy is required, as is necessary for robust orientation, then the many layers of processing inevitably give a delayed value. It is tempting to choose only one alternative, but there is a second sensor available. The Pozyx tag has a Bosch BNO055 robust orientation sensor on board. Using that for orientation as opposed to the MPU6050's gyroscope, it is possible to get the best of both worlds. So the decision is made to not read any orientation from the MPU6050 at all.

The `mpu6050` module implements the driver for the Invensense MPU6050 IMU as follows. For the MPU6050, the highest accelerometer bandwidth (260 Hz per the datasheet [32]) is only achieved with most forms of processing disabled. Full representation of the bandwidth (Nyquist rate) requires a minimum sample rate of $2 * 260 = 520$ Hz, of which the next-lowest configurable rate is 1 kHz. 1 kHz is convenient as many other system rates are a constant multiple of this. It is very important for the filter that this rate is constant and the time interval between readings remains constant as well. The data rate for communication with the chip over the I2C bus can be selected. At the default 100 kHz this is barely enough to read one register at 1 kHz. At 400 kHz this results in lower latency since bits arrive faster, which is preferred. Keeping the data short also helps, so omitting the 2 bytes of the z-accelerometer should reduce latency by 25%, which is significant. This is fine as the z-accelerometer will be zero unless some error condition occurs, which is also be detected with other sensors. Knowing that processing increases latency, configuration registers for all processing filters are left at their default (off). To make sure all samples that are produced are captured, the computer is set up to react to interrupts from the MPU6050. By pulsing the voltage on the interrupt pin, the MPU6050 indicates that the next value has just become available. It will do this at a constant 1 kHz rate, continuously after it has been configured. The computer has been anticipating this event, so without any delay it requests to read 4 bytes of the 6-byte register

for the accelerometer. Accelerometer latency was measured to be less than 0.3 ms up to the point where it is incorporated in the integrating filter. In addition, some power savings are realized by disabling the z-accelerometer and x,y-gyroscopes. No processing of the inertial measurement unit occurs in the `mpu6050` driver module, raw values are only scaled to SI units.

The Bosch BNO055 IMU is isolated behind the Pozyx tag processor, but can be read out through API calls. It can unfortunately not be configured to have interrupt-driven samples like the MPU6050, but luckily the default configuration is very sensible. The chip performs all orientation processing at default configuration at 100 Hz. In addition to the accelerometer and gyroscope, it includes a magnetometer to eliminate long-term orientation drift. One problem is that the communication has to go through the Pozyx tag, so also through the Arduino's serial interface. When the BNO055 estimate for θ arrives at the computer, it has incurred latency from the BNO055 inbuilt sensor fusion (10 ms), a mismatch in phase for the readout (max. 10 ms), and communication delays (1 ms for I2C and 4 ms for Serial), for a total of 25 ms latency. The faster the robot turns, the lower its forward velocity must be, making the effect of a delayed heading angle smaller. Nevertheless, the MPU6050 is still configured for low-latency angular velocity measurements from the z-gyroscope and can be used if beneficial. When orientation measurements become time-critical the UWB estimates have to be planned around them and will incur extra latency. But a relatively small ($<$10 ms) additional delay to absolute x- and y-positioning events is not as significant as delaying orientation updates. So the Arduino is set up to always process requests for the heading angle with the highest priority. Whenever a request arrives from the computer, the Arduino will communicate the most recent orientation that is available. The Pozyx positioning events are handled after orientation requests are handled, at a lower priority. This is discussed in detail in Section 4.1.5.

With this setup of two sensors the Kalman filter has all inertial measurements it needs to realize good performance. The MPU6050 provides high-order inertial measurements at 1 kHz and extremely low latency. And the BNO055 fulfills the very important low-order measurements of absolute orientation at 100 Hz.

4.1.4 Platform driver

The platform driver, `neato`, is in charge of communications with the robotics platform. It connects to the serial port. The interface to the platform is documented[1] such that it instills confidence that it is suitable for use as a robotics platform. However several arbitrary restrictions show that this is slightly optimistic. The manufacturer has designed this interface as a means to debug and test the machine first. The ability to control the platform as a robot is more of a side effect and therefore somewhat limited. Existing drivers for the platform, such as the neato_robot project[2] or the NeatoPylot project[3] are not designed for performance or even deal with clearing the error buffer. A different approach is needed if a stable system with performance guarantees is desired.

[1] Programmer's Manual `https://www.neatorobotics.com/resources/programmersmanual_20140305.pdf`
[2] neato_robot by Mike Ferguson, `https://github.com/mikeferguson/neato_robot`
[3] NeatoPylot by Simond Levy, `https://github.com/simondlevy/NeatoPylot`

The platform has a serial command interface which appears to runs at 100 Hz internally, evaluating at most one command every tick. This disregards the length or amount of commands queued up in the buffer and implies a scheduling problem. This means the platform can be actuated at a constant rate, but also requires us to synchronize higher-rate processes with this external rate. Furthermore, the response to each command will be placed in the output buffer one tick later. So in a synchronous system, the effective rate is halved.

The Neato's commands come in various shapes, of which many are undocumented. As limitations are only imposed by the peculiarities of the firmware and not by the documentation, a lot is possible. For instance, the interface always echoes the given commands back. The response is appended, terminated with a typically unused delimiter (`0x1A`). This makes it possible to identify responses to specific commands and receive them atomically. It is possible to enable continuous streaming of binary data that represents the Lidar distance measurements. The data format is not documented and will stream interleaved with regular responses, making it difficult to distinguish them. But the consistent formatting and differing nature of these streams make it possible to split them. So a demultiplexer was written that identifies the binary data and textual serial responses and packs them into two different queues. The Lidar distance data then takes up no ticks in the command queue causing no interference, and will stream with as little latency as is possible. This is especially useful for scheduling as the Lidar data normally arrives at an unpredictable rate due to rotational variation. A little reverse engineering of the Lidar binary data reveals reflection strength and a series of error codes that gives another dimension to mapping the environment.

For performance reasons this system should be asynchronous. A separate thread manages the execution of commands and, once started, will keep reporting and updating internal state. This has as consequence that:

1. Velocity setpoints are sent to the wheels at 50 Hz and encoder positions are requested at 25 Hz (see Section 4.1.4.1.)
2. Encoder positions are received at the same time that a new velocity setpoint is sent, so it arrives too late to be useful to determine that new setpoint.
3. High-frequency position updates will have to come from the accelerometer and gyroscope to guarantee all setpoints.

The driver is threaded and uses unidirectional threadsafe queues to communicate. Such a system can quickly be decomposed into separate processes with different scheduling priorities to facilitate meeting computational demands for real-time performance.

4.1.4.1 Neato command schedule

A schedule has to be composed as there is not a single function that provides full system state. After the platform is initialized into the proper mode of operation, complete utilization of the platform's communication bus is desired as this is a bottleneck to performance. Commands are scheduled over subsequent discrete ticks, which have a duration of 10 ms. The following functionality is divided over the 100 Hz command rate:

Wheel actuation is the most significant. Wheels are velocity-controlled by the platform itself, so this allows a lower rate than when a filter would directly actuate the wheels. A major requirement is that the actuation happens at a constant rate because of the nature of the filter. Therefore, a rate of 50 Hz is chosen for wheel actuation. At this rate, the maximum distance travelled between commands is 6 mm at $0.3\,\mathrm{m\,s^{-1}}$.

Wheel encoder feedback is the second most important. The platform computes wheel position from encoder pulses at high rates, reporting with 1mm accuracy. As pulses are counted internal to the platform, a lower readout rate is necessary than if this was done on the computer. A constant rate is again desired and as space for auxiliary commands must be left, a rate of 25 Hz is allocated.

A schedule of 33.3 Hz each for wheel actuation and encoder readouts was also considered, but encoders have discretization noise at 1mm and do not benefit much from this increased rate. At maximum speed ($0.3\,\mathrm{m\,s^{-1}}$) at 25 Hz gives us 12 mm per update with less noise, compared to 9 mm at 33.3 Hz with higher noise. Note that the noise from these measurements is not incremental, as the discretization is done by the platform which reports absolute values. As a consequence, travelled distance between wheel actuations would increase from 6mm at 50 Hz to 9 mm at 33 Hz at maximum velocity, which is a significant loss in precision. Also, with the proposed schedule 0.75 utilization goes to commands which affect primary performance, and with the alternative only 0.67 utilization is expected.

This leaves 25 Hz for auxiliary commands, which consist of the following:

- Reading out digital sensor values, such as the collision bumpers.
- Reading out analog sensor values, such as battery voltage and lifting height.
- Reading out and clearing reported system errors (very incidental).

Each of these operations take one tick.

From these, *collision bumper state* is the most significant. The bumpers contribute to an essential safety feature, as an emergency stop mode. In part, a collision can be detected by an unexpected sudden change in acceleration. But as long as there is a choice, priority can be given without compromising performance. The collision bumper state does not need a constant rate, it is about minimizing latency (worst-case and average).

At 8.3 Hz (equal share between these three tasks) this is a worst-case latency of 0.12 s excluding processing delay. During those 0.12 s the robot can travel 36 mm into an object, during which it increases driving force to preserve velocity. Three ticks between commands are due to wheel actuation and encoder feedback, but with one other auxiliary command this becomes seven ticks. These seven ticks is the minimum of the schedule in the worst-case, which results in a worst-case latency of 0.08 s. At the highest speed this is 24 mm of travel after a collision could have been detected. More bumper state readouts can be scheduled, this will only reduce the average-case latency. When bumper state readouts are alternated with auxiliary commands, the average-case latency is 0.04 s. With two subsequent bumper state readouts, the average-case latency becomes 0.03 s. With three this becomes 0.025 s, and with four, 0.024 s. These only consider scheduling and command latency. Latency induced by processing is insignificant, and inertia and braking force of the system is not yet considered.

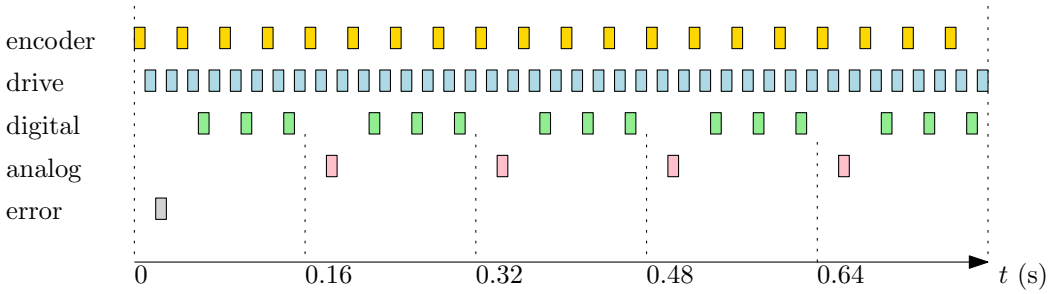

Figure 4.5: One period (0.8 s) of the 100 Hz platform command schedule.

Because of diminishing returns, it was chosen to have three subsequent bumper state readouts for each auxiliary command. This corresponds to an average 7.5 mm of controlled travel into an object after a collision occurs at maximum velocity. This is assuming that the accelerometer collision detection and Lidar collision avoidance both fail. Doing significantly better than this comes at the cost of performance or system complexity.

Leftover auxiliary command rate is now at 6.25 Hz, which is shared between the analog sensor readouts and the system error readouts. Another four analog sensor readouts can easily be performed for every possible system error, dedicating an average 5 Hz to this. The platform system errors are infrequent, but queued and typically occur in batches, so reading at 1.25 Hz seems fine. If system errors are not cleared, the platform may lock up and crash.

The total schedule can then be summarized as follows, repeating every 80 commands (0.8 s):

```
SCHEDULE:  encoder, drive, SECONDARY, drive.
SECONDARY:  TERTIARY, digital, digital, digital.
TERTIARY:  error, analog, analog, analog, analog.
```

One period of this schedule is illustrated in Figure 4.5.

4.1.5 UWB driver

From Section 4.1.3 it has become obvious that low-latency readouts of the Pozyx tag's orientation sensor are very valuable. But orientation values are used in the controller which has to satisfy real-time guarantee. If UWB ranging is performed at an unfortunate moment, during the time that the controller requests the orientation, it will likely lead to a missed deadline for the controller. So UWB positioning has lower priority, it comes second to orientation readouts. To solve this problem, the system must always be idling whenever an orientation readout is started. This is possible as orientation readouts happen at a constant rate of 100 Hz, with 10 ms in between requests. So all UWB-related operations are scheduled to occur right after the orientation request is handled. And the busy period must be $< 10\,\text{ms} - \epsilon$ in duration or the next deadline may be missed. In this case, ϵ represents maximum clock jitter, as the Arduino and the computer do not have synchronized clock sources. Out of caution $\epsilon = 0.5\,\text{ms}$, allowing for a deviation of up to 5% from the configured clock, which is rather tolerant.

The Pozyx API that runs on an Arduino-compatible microcontroller is essentially a high-level abstraction, a wrapper for the Pozyx tag's I2C interface. This interface can be described as a sequential memory region of configuration registers that toggle modes of operation, with direct read- and write operations. The Arduino is an I2C master to the Pozyx tag slave, with the result that the Arduino decides the exact time and duration of communication. The slave can introduce no delays or cause preemption, at worst it will indicate errors by inaction which reduces communication time. This makes it perfect for real-time scheduling with deadlines. The results are communicated over the UART serial interface to the computer at a speed of 115 200 bps, with all data packed in a deterministic fixed-width format. This data is buffered asynchronously so the Arduino does not need to wait for any response from the computer to write data, and the buffer can be emptied when it makes sense. For the details of how the controller reads this buffer, refer to the schedule of the controller in Section 4.2.4.

The API has two interesting functions for determining position. The `localize` function does multilateration with 4 anchors, which takes approximately 75 ms to complete. An assumption is made that the tag remains stationary during those 75 ms, but at $0.3\,\mathrm{m\,s^{-1}}$ the robot can move 23 mm which is an additional inaccuracy. The `range` function determines the distance between an anchor and a tag, and takes 15 ms to complete. For doing simple multilateration with four anchors as described in Secton 3.3, this ideally takes 60 ms to complete. Such incremental multilateration can be realized by calling the `range` function for each anchor with round-robin scheduling. Then every 15 ms a new estimate can be computed, should the anchor respond. Anchors that do not respond will have another chance after one scheduling cycle. This setup works for areas where the number of anchors is small and the anchors are always within range. For larger areas with more anchors, a local method for anchor selection and scheduling is necessary. A more thorough analysis is given in Section 4.1.5.1.

When the Pozyx API executes one of these functions, initially it writes several registers in burst, followed by a busy wait of reading the status register until the result is available in the corresponding register. A single call to `localize` takes 75 ms, which is too long to fit in the period. But if the busy wait is omitted, the initial configuration write easily fits within the schedule and needs not be segmented further. However reading the result of a 4-anchor localization takes significant time (more than 10 ms), due to the amount of data. This is unfortunate as all that is necessary is the x, y- and error-estimate. Unfortunately, the API does not offer a more fine level of access. The system is not mature enough to start optimizing the register access schedule, as this will likely lead to unexpected errors, as seen in Section 4.1.5.1. As a workaround, only the lowest-latency ranging algorithm with a single range- and error measurement is specified. This readout takes 3.5 ms, which is short enough to be scheduled. There are some significant advantages to using individual range measurements as well, as discussed in Section 3.3.

One additional low-priority task that also requires carefully timed performance is setting the RGB status LED. This is otherwise problematic as the specific type of LED used, a WS2812b, accepts the RGB configuration over fixed-frequency synchronized clock. The Raspberry Pi computer is poorly equipped to generate this signal, but standard libraries exist for the Arduino. This task is of very low priority but takes a relatively significant amount of time (≈ 2.2 ms). So it is only scheduled opportunistically, when time allows, and never interferes with other activity. If the LED value has not changed since the last time it was written, no write is performed and the duration is significantly reduced.

Operation	lower bound minimum	upper bound measured	upper bound reserved
Read orientation	0.7	0.9	1.5
Initiate ranging	0.7	0.8	1.5
Read ranging result	3.2	3.5	4.0
Read ranging result not ready	0.7	0.7	1.5
Status LED write	0.1	2.2	3.0

Table 4.1: Duration of operations on the Arduino microcontroller, in ms.

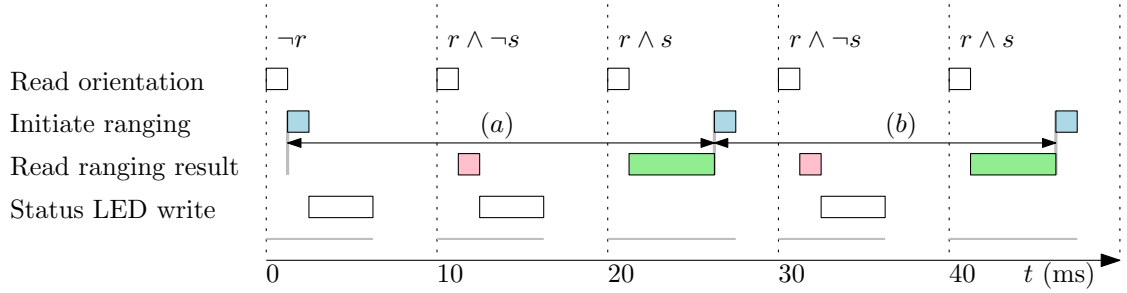

Figure 4.6: All cases of the scheduling of operations as they occur on the Arduino microcontroller, containing five periods with three different schedules.

The task durations to be scheduled are summarized in Table 4.1, and these include transmission of results back to the computer's serial buffer. The sum of all reserved operations exceeds the maximum busy period of 9.5 ms, so a sequential schedule is not permissible. From the start where ranging is initiated up until results have been read takes 15 ms, but the period is 10 ms of which the read itself takes 4 ms. The schedules are presented in Figure 4.6. The normal procedure is to always read orientation first to get the lowest-latency most recent orientation estimate. It is followed either by initiating ranging if it has not been started ($\neg r$, blue in the diagram), or if ranging has been started before (r) by attempting to read the ranging result. The result is either ready (s) or not ($\neg s$), in the latter case the read will complete early (pink in Figure 4.6). Either after ranging is initiated or the ranging read failed ($\neg r \vee (r \wedge \neg s)$), at most 3 ms have passed. So a status LED write can be scheduled opportunistically, which can take an additional 3 ms. After the result has been read, at most 5.5 ms have passed, which allows us to re-initiate ranging immediately ($r \wedge s$) giving one `range` result every 2 periods, or 20 ms. This results in an initialization with schedule $t = [0 \ldots 30)$, and continuous repetition of the region $t = [30 \ldots 50)$. Should some error occur, the system is resumed with re-initialization using $t = [0 \ldots 10)$.

Note that because of these different schedules, two possible durations for the ranging request can occur. Timing like (a) from Figure 4.6 is longer and occurs when the request has started at the beginning of the period ($\neg r$), which happens during initialization or whenever some communication goes wrong. Whereas timing like (b) is typically shorter as the request was made right after the previous read finished. The minimum length occurs at the latest start time of the blue request subtracted from the earliest end time of the green read two periods further. For (a) this is $(20+0.7+3.2)-(1.5) = 22.4$ ms, and for (b) this is $(40+0.7+3.2)-(20+1.5+4.0) = 18.4$ ms, which both give sufficient time for the request to complete. For (a) with only one

period in between, the maximum length is the earliest start time subtracted from the latest end time, which gives $(20 + 1.5 + 4) - 0.7 = 14.8\,\text{ms}$, which is still slightly too little to make the read succeed the first round. The maximum length of (b) with one period inbetween is necessarily shorter than for (a). This shows that one period inbetween is insufficient, so the schedule shown in Figure 4.6 is complete. In addition, as there must be one period with a failed read inbetween, the LED update task cannot be starved. This holds even if the read for (a) succeeds after one period as the LED status update is also done when $\neg r$.

The properties of the schedule presented in this section have been shown to hold in tests. If the extended debug output is enabled however, the timings are longer than presented in 4.1 and the schedule cannot be followed. But with minimally interfering debug output and practical evaluation, it was verified that the timing is as expected. The typical latency for UWB `range` measurements in this system is $\approx (15 - 5) + 5 = 15\,\text{ms}$, as there is 5 ms of additional waiting time, but the radio signal for TDoA ranging is not broadcast until 5 ms after the `range` measurement starts. In a commercial Ubisense system with a latency of 184 ms[74], at $0.3\,\text{m}\,\text{s}^{-1}$, the extra error due to latency is 0.055 m. With this method, inaccuracy due to latency is reduced to 0.0045 m, which is an order of magnitude better. So the proposed solution is quite favorable to the point that computational techniques such as delay compensation are no longer of significant benefit.

To conclude, a working deterministic schedule was found, implemented, and verified, that minimizes the latency for heading requests and allows the `range` function to complete once every 20 ms, working synchronously with the controller.

4.1.5.1 Pozyx evaluation

This first public version of the Pozyx system has been evaluated with the most recent firmware and Arduino API available at 2016-07-13. Note that as of 2016-08-25 version 1.0 of the Pozyx firmware has been released, which is a major change. Version 1.0 is not evaluated due to time constraints, so the evaluation that follows here concerns the previous older firmware version. The infrastructure consisting of six Pozyx anchors has been set up in the office of Segula Technologies Nederland B.V. as per Figure 4.7. Anchors are restricted in that they have to be positioned near power outlets and this configuration is among the best possible. The figure shows some interesting regions for evaluation. The yellow region is out-of-bounds, tags in this region have all anchors positioned on one side of the tag. This makes multilateration more ambiguous as the distance measurements are much less orthogonal. In pink regions two anchors are positioned almost behind each other, which makes one of the anchors redundant for multilateration. Similar reasoning holds for tags positioned on the blue lines, where distance from the blue line is inaccurate and should be determined by a different anchor.

The layout from Figure 4.7 ensures that for every yellow region, anchors are positioned on the furthest extents (the corners) to ensure maximum orthogonality. For every blue line and pink region, anchors orthogonal to the direction exist within range. Any three anchors are never on the same line, and anchors 4 and 5 can act in stead of the far anchors in the inevitable case of interference.

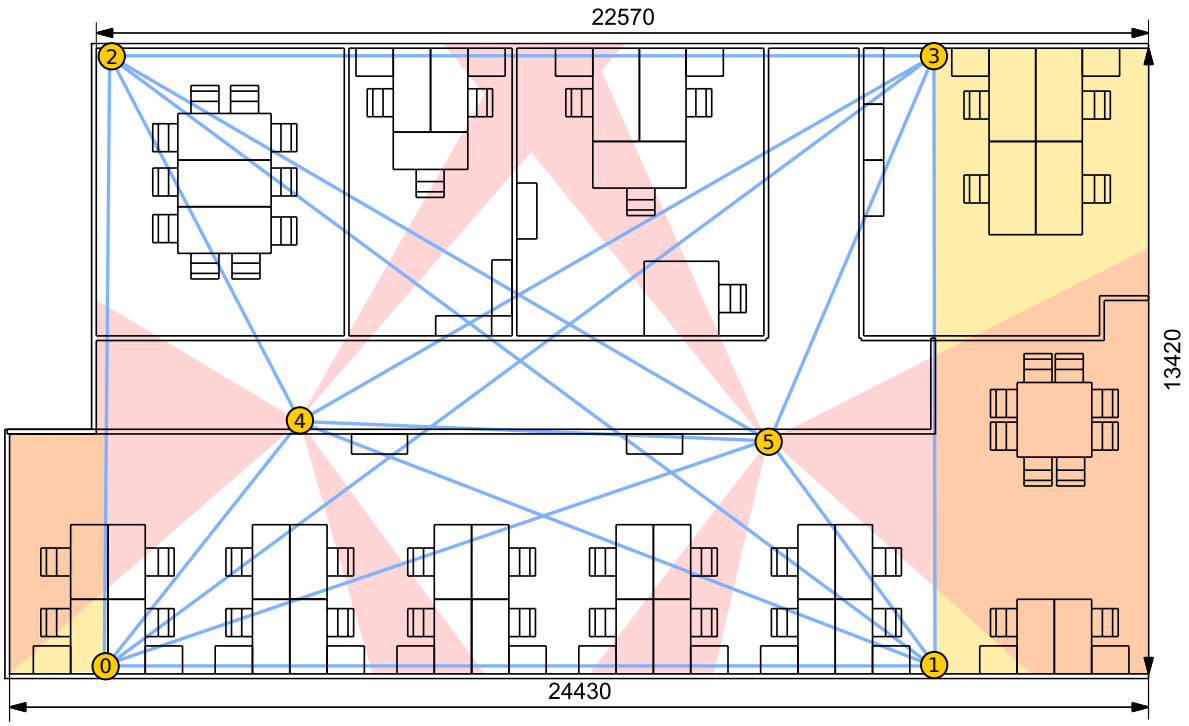

Figure 4.7: Six Pozyx UWB anchors with their configuration order number, positioned in an office environment. Dimensions in mm. All objects (black) are possible sources of interference. Interesting regions are colored.

Auto-calibration can be done over very long periods to stabilize the anchor position estimate, as long as an accurate height is provided. All anchors were placed at 2.48 m height from the floor. The exact location of the anchors has been measured and auto-calibration was used to verify the measurement.

The Pozyx system offers different native methods of localization. One configuration parameter is the type of localization, which is either 2D, 2.5D or 3D. The 2D mode is good for estimating with all anchors and tags in the same plane. This can not be true in the presented setup as tags move close to the ground, but for this evaluation a tag was moved approximately in the same plane as the anchors. The 2.5D mode takes known heights for all anchors and the tag, and computes coordinates in the plane of the same height as the tag. The 3D mode does full 3D multilateration including an estimate for the height, which is difficult as all anchors are positioned in the same plane. Another configuration parameter is the localization algorithm type, which is either UWB or Least-Squares (LS). The UWB algorithm is apparently an undocumented stateful filtering algorithm that is said to work well with non-line-of-sight between anchors. The Least-Squares algorithm is a direct linear method that minimizes the squared error using a pseudo-inverse. A third configuration parameter is how many anchors are used for the multilateration, which can be four or more. For this evaluation, both the advertised minimum of four and the whole set of six anchors are tested. An evaluation of the combinations of these parameters are presented in Table 4.2. A score of – – means the result is unusable, – signifies that the result is inaccurate to the point of unsatisfactory, + indicates that it works but with some negative aspects, and + + when the system functions

algorithm, # anchors		positioning			out-of-bounds			height		
		2D	2.5D	3D	2D	2.5D	3D	2D	2.5D	3D
UWB	4	+ +	+ +	+ +	+ +	−	−	− −	+ +	− −
	6	−	−	+	−	+	−	− −	+	− −
LS	4	+	− −	− −	+ +	− −	− −	− −	− −	− −
	6	−	− −	− −	+ +	− −	− −	− −	− −	− −

Table 4.2: Comparison of native modes of localization with the Pozyx UWB system.

as expected. During these tests, it was ensured that the tag has line-of-sight and can range with at least four anchors.

A good localization method has both good positioning capability, an ability to determine height correctly, and can deal with out-of-bounds situations reasonably well. From Table 4.2, it becomes immediately obvious that something is wrong with the Least-Squares algorithm, and that the 6-anchor version performs much worse than the 4-anchor version. For 2.5D and 3D, the Least-Squares algorithm produces either zeros or no output, so it must be defective. For all 6-anchor variants of localization, the individual range measurements start to error over time, where more and more anchors become perfectly correlated copies of others until only a singular value remains. Another notable defect is that the 3D mode fails to provide a height, as all returned heights are constant and wrong. This opposed to the 2.5D mode which was supposed to give the constant configured height, but provides a varying estimate anyway. The result of this mixup is that the 2.5D mode is less accurate than it could have been had it assumed constant height, and the 3D mode cannot be made to work like the 2.5D mode because there is no method to configure the tags height. These are obviously bugs.

The 2D mode requires all anchors and tags to be in the same plane. For the 2D mode with the Least-Squares algorithm, the performance is okay, and especially good for out-of-bounds estimates. But for the 2D mode with the undocumented UWB algorithm, accuracy is even better. However the assumption made for this mode cannot hold, because when tags are all placed near the floor, interference from metallic objects would decrease the range to the point where it no longer works, so the 2D mode is out.

The only mode that can possibly give a practical estimate is the 2.5D mode with the UWB algorithm with four anchors. Independent of how the algorithm is implemented, internal height ambiguity will decrease the x, y-positioning accuracy. Suboptimality of the algorithm follows from the presence of a height estimate, which indicates that the constant configured height is ignored during localization. The estimate always assumes that tags are positioned below the anchors, which is a deliberate choice that resolves at least that ambiguity. The 2.5D mode does not provide an accurate estimate out-of-bounds. Observed is when moving away from the center, when passing the bounds into the rightmost yellow region of Figure 4.7, the estimate suddenly moves in the opposite direction. This is likely due to some internal ambiguity resolution that gives preference to estimates within bounds. But unlike where it helps to resolve the height ambiguity, it now makes it impossible to do positioning in out-of-bounds areas, even several centimeters past the border. This severely limits the setup of a Pozyx network which currently only supports four anchors. This behavior is not observed in the 2D mode, which consistently gives good out-of-bounds estimates.

When observed from an implementation perspective these results are sign of a mistake. The 2.5D mode can be realized by projecting all range measurements into the constant-height plane and performing localization with the 2D mode and Least-Squares algorithm. But for some reason this is not happening and the 2.5D mode behaves completely different from the 2D mode, while the 3D mode is made useless. A possible explanation is that the 2.5D and 3D mode are switched in the firmware, which also explains why the 2.5D mode estimates height whereas the 3D mode height remains constant.

It is expected that after this firmware matures, more significant optimizations can be achieved, especially concerning latency. One parameter that is very tempting to tune is the data rate, as higher data rates have lower range but also significantly lower latency. This becomes especially significant as robots increase in velocity, or when many more robots join to communicate on the same channel.

Above all, the Pozyx algorithms have been seen to reuse values from prior range measurements to ensure better estimates when some anchors become unavailable. This way the system cannot produce statistically independent samples, which breaks an assumption for the controller. It also could affect several earlier analysis which assume low latency. The inherent unreliability of radio-based systems makes it tempting to cache old values rather than simply giving errors. More mature commercial systems also exhibit this behavior, such as the Ubisense UWB tags. This is evident from the fact that despite unreliable communications they always have a reliable constant update rate, which coincidentally explains the long signal latency. And the resulting properties are not friendly for integration of such tags into another positioning system.

After all no satisfactory solution is found for continuous out-of-the-box localization. Ranging however does work reliably and a positioning algorithm can easily be realized with this functionality. By doing a custom implementation several desirable properties can be ensured:

1. Statistical independence is a requirement for a Gaussian distribution, which the controller assumes.
2. Predictable latency provides upper bounds for certain types of positioning error.
3. Lowering latency to the same order of magnitude of other sensors reduces the need for complex compensation.

By reporting the raw periodic range measurements directly, the Spring algorithm described in Section 3.3.1 can be used instead, which exhibits all these properties. The initial rough localization is done from the stationary position, which can use the 2.5D mode with the 4-anchor UWB algorithm. From that position on, range updates are applied by the Spring algorithm, approximately every 20 ms. This ensures convergence of the position estimate during motion after two range queries from different angles, which has much lower latency than a full localization update would have taken.

4.1.6 Assembled prototype

Figure 4.8 shows the assembled prototype with the controller as a stack of circuit boards. This setup has the camera oriented towards the ceiling to verify localization accuracy with camera

Figure 4.8: Assembled prototype of the robotics system with mounted electronics, and a charging base on the background.

pose estimation. For a multi-robot system the camera will be pointed to the front of the robot to do pose estimation of neighboring robots when performing formation control. The cables are connected with right-angled connectors to the platform's ports and are sufficiently short as to not obstruct or allow the robot to get caught on objects. Currently, the robot and controller both have to be powered on manually before the system can become operational. The platform contains an LCD, but the platform's debug interface does not allow to use this as a data display. The controller is positioned such that the IMU is in the center of rotation of the robot. It is mounted with slight elevation on a transparent base to ensure that the Lidar measurements travel unobstructed underneath. This base acts as a lightguide for the status LED and rests on the top of the Lidar module. The overhang is supported by single thin pillar in the Lidar's dead region. The thickness of the base (8 mm) makes it sufficiently stiff, which reduces resonant vibrations.

4.2 Sensor integration

The drivers provide raw sensor values, of which only a subset is interesting. It is quite tempting to directly access the drivers from high-level logic, but this does not make the

platform very flexible or performant. So the `sensors` module provides a useful abstraction for universal access to sensors, irrespective of their origin and access mechanism. This enables faster simulations, allows for non-real-time processing, record and replay, etc. while still providing performance where necessary. Performance of the `controller` module would have been severely impacted by the introduction of an additional layer, but careful design has accommodated this integration so that it is hardly affected. The following sections go in-depth on the way that the `fakeworld`, `sensors`, `controller`, and sensor-specific modules function and have been interwoven.

4.2.1 Processing

Some sensor data needs processing to be useful at all, as is the case with Lidar and the camera, while others directly output perfectly usable values, for instance for the orientation and UWB position. The specifics to these are handled over several modules.

The `lidar` module contains algorithms for processing Lidar data. Update regions are extracted from the Lidar point boundary as described in Section 3.2.2.1. The resulting grid can be applied as an update to the space map W_S. This system works as designed if position error is below the grid resolution. If the position error is larger than the grid size, excessive toggling of cells on border regions is observed. Workarounds such as to build confidence before applying updates result in latency for detecting obstacles and planning around them, which is not desired. Instead the focus should be to use various additional techniques to improve position accuracy. A wall detection algorithm fits a line to noisy distance measurements close to the wall. Ideally this gives an accurate position and orientation for walls, which can then be compared to a previous frame to obtain accurate relative translation and rotation. This has the ability to stabilize the position estimates whenever straight walls are around, which is quite often in indoor situations. But the algorithm is not finished and performs a bad fit of the wall due to unknown bugs.

The `camera` module does image processing to realize pose estimation. Both with markers on the ceiling to determine a robot's own position, and to determine the relative position of neighboring robots for formation control. The camera used is a 5MP camera attached over a 2-channel MIPI CSI-2 interface to the Raspberry Pi. This interface can theoretically give 2Gb/s of bandwidth. It is very well-supported by the Raspberry Pi platform and its community. Aruco markers [22] have been chosen as 2D optical landmarks. The C++ Aruco library[4] offers a complete, highly robust method for pose estimation. To go from a location in an image to a position in the space, the camera's parameters need to be known. Normally camera calibration is a complicated process as there are many calibration parameters. The Aruco library also offers methods to perform camera calibration from a simple printed chessboard pattern with Aruco markers inside of the black squares. A wrapper is available for the Python language, `python-aruco`. A 5MP image is read from the camera module and repeatedly processed by `python-aruco`. The image will have been auto-adjusted for exposure and color by the camera module. A high-resolution image benefits the accuracy of the algorithm's pose estimate, especially with regard to the orientation. Since processing of large images takes quite a lot of time, a single core is dedicated to perform this task.

[4] ArUco, `https://sourceforge.net/projects/aruco/`

The setup of the inertial measurement units has been described in Section 4.1.3. The output of the drivers provides x- and y-accelerometer data at 1 kHz, and absolute orientation around the z-axis at a rate of 100 Hz. This data is fed without further processing to the `controller` module, which performs sensor fusion. The output of which is an estimate of the pose $p = (x, y, \theta)$ and its first two derivatives, $\dot{p}$ and $\ddot{p}$. This estimate is then offered back to the `sensors` module and is used as the best-known value for the pose.

4.2.2 Simulation

For simulation, the `fakeworld` module provides computational shortcuts and a virtual world with simulated physics. A selection of these shortcuts can be enabled, depending on the purpose of the simulation.

There are two obvious approaches to simulating a multi-agent system. One is to launch separate processes for each agent with true isolation. But then simulation has to occur real-time, or with some form of clock scaling and synchronization. Also global events have to occur in a shared world, which is not available during distributed simulation. For simulation purposes it is fine if it is predetermined, including dynamic behavior such as moving obstacles. Another approach is to create multiple instances of objects in the same process, multiple controllers and world models, etc. This method benefits from shared state and memory reuse, but requires the discipline of properly isolating variables in modules. It is also more predictable in behavior and offers more control over the whole system. This approach is chosen for the current implementation. It is still easy to switch to a multi-process simulation, but the reverse would not be as trivial.

For simulating the platform, an obvious need is a virtual world. Assuming that the collision avoidance algorithm works, there is no need for collision detection and all movement can be considered within C_f. Two setpoints for control are left- and right-wheel velocity. The wheels can be treated as ideal without slipping. As the platform has high torque and does internal velocity control of the wheels, changes in velocity setpoints propagate practically instantaneous. The physics model can then omit the mass in favor of a maximum Δv, simplifying the model. Position, orientation and the first two derivatives make up the true pose, for which the relevant equations can be found in Table 3.1. Forward acceleration is directly derived from the change in position and orientation. As these derived measurements can be numerically perfect, noise is added that is characteristic to each sensor. Sideways acceleration now only occurs in the form of centripetal force, which is a direct result of moving in a circular arc, which is derived directly from differing left- and right-wheel velocities.

The `pozyx` and `mpu6050` module are derived as described in the previous paragraph. The `neato` module which interfaces the platform has several on-board analog and digital sensors. Most sensors are unused and follow lazy evaluation, meaning they get computed only when an attempt at a readout is made. Note that all three of these have inertial sensors.

The Lidar data originates from the neato module, and can be computed by means of raytracing. For this, some obstacles in the environment need to be defined. Arbitrary lines and circles are supported for computing intersection points (distances) when raytracing. But the raw Lidar data may not be necessary if the goal does not include evaluation of the Lidar processing algorithms. The main output is a region as described in Figure 3.5 on page 40. Such a

region update can be determined by a square window function on a separate "physical world" space map. This physical world is provided as an image representation of the scene – a monochrome bitmap. Time-based dynamics is realized by superimposing images of obstacles with time-varying positions. Then, changes to the world model's space map are still only recorded after the robots are in proximity or actively communicate it to each other. Adding noise to the space map is done by edge detection on the monochrome image and to randomly toggle some of the resulting pixels.

The `camera` module's processing must be completely circumvented during simulation. The pose estimation output can be derived completely and very efficiently from the internal physics state. This frees up one core of the CPU previously dedicated to image processing. The rate of pose estimation was limited to the image processing speed, but now it needs an artificial limitation. This rate is chosen to be 5 Hz, similar to that of the Lidar update rate.

Robustness to errors can be simulated by perturbation of the virtual world and observing the resulting behavior. For instance, a collision could be done by changing a robot's inertia rapidly and leaving it stationary afterwards. The system supports the introduction of such events by conditional triggers, with arbitrary environment access. This makes it sufficiently flexible for any variant of experimental validation steps.

Some other modules need special consideration. Simulation is made faster than real-time by raising this rate to that which causes maximum CPU utilization. In this case virtual time must be provided in the sensor module as access to the real system-based time might make modules misbehave. The controller is synchronized with the 1 kHz accelerometer interrupt rate, which is determined in the sensors. All other I/O rates are derived from the accelerometer, so the system is quite scalable in this. Since the execution priority of the controller is much higher than that of for instance the planner, it is at risk of starving other modules. The solution is to also proportionally increase the tick rates for other modules and to scale back when reaching 100 % utilization. This is done by continuously monitoring process CPU utilization from all processes that were dispatched at the start of simulation and adjusting the master time scale proportionally. Note that since the controlling processes of many robots are being monitored, performance may become lower than real-time for some simulations. This is of course intentional and perfectly fine with simulated physics, otherwise simulation may not achieve accurate results.

4.2.3 Abstraction

This describes the interface of the `sensors` module and the reasoning behind it. The main purpose is to ensure propagation of the raw or processed sensor data to the interested modules. Dependent modules have different requirements and the interface should satisfy all of them.

The `controller` module is all about low-latency access and does not benefit at all from a separate module that caches results. The controller uses the accelerometer, the heading angle, UWB estimates, encoder distances, and some specific events. However if the controller absorbs certain sensor events then the rest of the system will not be able to react to them. As a solution, the data that the controller needs is retrieved through a lightweight wrapper, directly from the driver. The wrapper does not even take the opportunity to read or copy the value to a cache itself. The controller provides measurements to the sensor module explicitly after the

controller has used the value. Measurements are placed a thread-safe FIFO queue defined by the `sensors` module, which is consumed asynchronously. This isolates the low-latency part of the system from the unpredictable execution times of other processing algorithms.

The `worldmodel` module defines four data structures, $W = (W_S, W_E, W_N, W_C)$ as defined in Section 3.2.2. Their update mechanisms define the shape of the data, but not the method of delivery. Some sensor data is processed asynchronously and results in spontaneous update events, as is the case for Lidar updates to W_S and pose estimates on W_E. But decisions on the world model do not take place at frequencies above 10 Hz, so this data might not be needed until right before the decision logic occurs. In that case, such updates are queued to be processed together with incoming network updates (all of W) and periodic updates (W_C, W_N), so that the most recent data is available at the point of decision. This type of sensor data can be queued up in another thread-safe FIFO-queue. Exceptions to this are emergency events, but when detected these are passed straight to the controller without delay and other modules will know by polling. With other data, only the most recent value is of interest, for instance for the current pose p or the battery charge level. It does not make sense to process all intermediate values, so only the most recent raw measurement is cached. The interface for all such events is exposed and consists of a list of methods that perform sensor processing with lazy evaluation. After being processed the result is cached until a new measurement arrives.

Raw historic sensor data is very useful for debugging and optimization. As sensor handling is centralized, it becomes trivial to collect. So all values can be optionally logged to a database before any processing is done. The `fakeworld` module can read and replay such data for testing the processing algorithms without using the physical system or without even running real-time. In such a replay, the controller is unable to affect the system and will become unstable, so it is best left disabled.

To break the requirement that the system has to run at real-time speed a function for virtual time is introduced. The `sensors.time()` function is used in all modules instead of the standard Python `time.time()` function. During normal operation, `sensors.time()` is just an alias for `time.time()`. But for simulations the function increments slower or faster depending on the processing needs. In this way large networks with intensive algorithms can still be simulated on a single computer.

4.2.4 Controller

The primary purpose of the controller is to provide a state estimate of the pose p with better properties than any of the individual sensors. This is as much about approaching the readout of the sensors in an intelligent way, as it is about tuning the parameters of the filter. A secondary purpose is to provide velocity estimates for the wheels in order to move the robot closer to the next point on the path. A multi-rate Kalman filter implements a state observer and is the heart of the controller. The implementation uses the Python module `filterpy`, for which extensive documentation exists with regard to tuning and optimization[5]. System states are $p = (x, y, \theta)$, $\dot{p}$ and $\ddot{p}$. Derivation of measurements to this state are as described in Section 3.1.2.

[5]By Roger R. Labbe Jr, `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/`

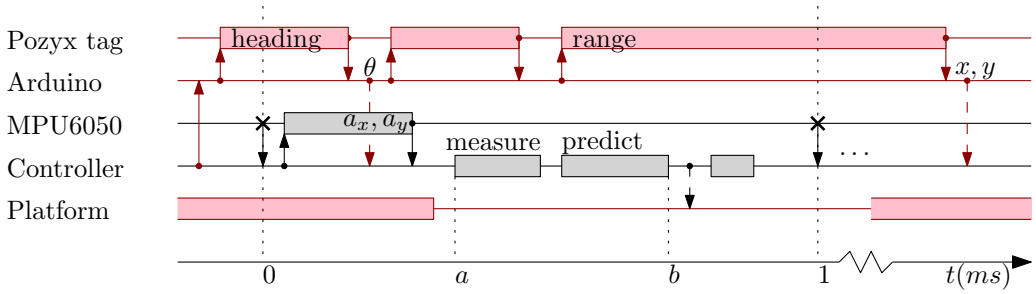

Figure 4.9: Communication order with the controller over a 1 ms period. Solid arrows indicate synchronized communication. Dashed arrows are asynchronous. Red parts observe a 10 ms period.

In the previous sections of this chapter, much attention was given to reducing latency. Without a proper integration in the last step these efforts will be in vain. Figure 4.9 outlines the strategy toward integrating these measurements. Not all requests occur every period. After the controller receives an interrupt from the MPU6050 IMU, it immediately performs a blocking read for the accelerometer a_x, a_y values over I2C. From time a the controller enters the measure stage of the Kalman filter and processes most recent values as per Section 4.1.5. Immediately after the controller continues with the prediction stage that finishes at time b. Now a new estimate is ready and sufficient information is known to actuate the wheels, so velocity setpoints are updated. These will be used asynchronously, the next time a `drive` command is sent per the schedule from Section 4.1.4.1.

After time b, the controller has some time left to prepare for the next interval. Every 10 periods it requests the heading angle from the Arduino over the serial connection, which in turn starts the 10ms schedule as described in Section 4.1.5, as illustrated in period $t < 0$ in Figure 4.9. Since this is an asynchronous serial connection no time is spent waiting for data to be transmitted. The next period ($t = 0$), θ should have arrived and can be read from the serial buffer and processed. The period after that ($t = 1$), potential UWB range events can be handled should they be available. For other periods various queues and buffers are emptied to prepare for the next measure step. This includes the encoders, platform bumpers, etc. and such updates vary for every period. Each such update has a maximum duration and care is taken to not exceed the 1ms period. This schedule is determined offline.

A practical evaluation is currently not available as the controller implementation is incomplete.

4.3 Logical system

From this stage on, the system has more similarities to an information system than an embedded system. Under this category fall the networking implementation and all aspects of inter-robot communication. This is of course closely related to the world model and the data it stores over time, which is visualized by a web-based monitoring system.

4.3.1 Networking

The networking implementation concerns itself with both physical and logistical aspects. Of infrastructure and network protocols, dealing with packet loss and congestion, to of course the message content. Infrastructure can be kept simple until the situation no longer satisfies. The same simplicity is applied to network protocols. Provisions for all of these must of course be made early on, or their future implementation becomes prohibitive. Therefore, most of these options are already reflected in the message format even if they may not seem directly useful.

The Pozyx UWB radio supports data transmission, but has very low bandwidth, in the order of kilobits as opposed to megabits per second for Wi-Fi. Even though the range is likely less, the maturity and bandwidth of narrowband IEEE 802.11b/g/n Wi-Fi is preferred. In addition, existing hardware, drivers and TCP/IP stack on the computer can be used. UWB radio may still be useful for sending long-distance homing information when Wi-Fi coverage breaks down, but is currently not used for anything other than positioning.

The implementation of mesh networking can be realized completely independent of any application-level decision on communication. This is typically done through a virtual network interface with either a kernel-based (BATMAN-Adv, Babel) or userspace (OLSR, [12]) routing system. If implemented, the choice goes to BATMAN-Adv as it is in Linux mainline and is optimized for robotic teleoperation [27]. But as the system is evaluated in an office network, connectivity over an existing Wi-Fi network is likely. This means there is no need to implement any of the mesh networking discussed before. So point-to-point signal between robots is no longer of a concern as access points are typically mounted in more optimal locations. Also latency is reduced to two wireless hops in all cases, from robot to access point to robot. Mesh networking can still be implemented for evaluation purposes, and is still necessary for scenarios such as cooperative lawnmowing. But such an implementation is not realized in this thesis.

Whereas reliability and retransmission are possible to deal with manually using UDP, TCP can do the same for free at only a cost of latency. Latency has an inverse relation with link reliability. As the test environment was far from congested, latency was low and predictable and there was no direct need for an alternate implementation. One problem is that TCP channels cannot be shared, leading to a quadratic decrease in network capacity in the number of robots that join. For now multi-robot simulations have not been network-bound as the loopback interface is used, so none of the broadcast-based transmission setups previously designed have yet been implemented. For congestion control, the standard TCP algorithms of slow start, congestion avoidance, fast retransmit, and fast recovery are inherited. Their properties are not very desirable as there is little feedback and they may only increase latency, but at least this offers protection from the channel being flooded.

Websockets are a modern take on classical sockets, unlike raw TCP sockets that listen on dedicated ports, they are initiated over HTTP and can be routed with URLs. These have the advantage of being easily used with web-based clients. All while incurring no major disadvantages. Many backend technologies are possible but all that is needed is a low-latency and working combination. For this Bottle is chosen as a low-overhead web service and request router, with the bottle-websocket extension for maintaining websockets. Internal to these libraries, Gevent is used for low-latency handling of potential multiple simultaneous clients. A HTTP service is running on port 7622, with the `/` location providing service identification and

the `/messages` location a websocket instance of the shared communication channel. Initially some library problems were experienced as binary data is not typical for websocket streams, but this was overcome.

The exact representation of data structures on the line is handled by serialization of the messages defined in Section 4.3.1.1. Deserialization can only be done for individual messages with known length, but the length is not included in the serialized format. Since messages are placed in a TCP stream for sequential transmission, each will be prefixed with the length as a 32-bit unsigned integer in network byte order (big-endian). The per-packet TCP checksum allows error detection on the data, and subsequent TCP retransmissions will ensure integrity of the stream.

Network performance has been evaluated in a test setup by sending direct relative motion commands at 20 Hz to the robot by manipulating a 6-DoF USB input device from a Wi-Fi-connected laptop. The input device is a Space Navigator from 3DConnexion and registers rotational and translational offset of a disc-shaped object. The angle of the z-axis and forward motion vector were mapped to rotational and forward velocity respectively. These were encapsulated in *move* messages and follow the complete processing pipeline, but skipping the processing intervals of the world model and controller, and instead directly actuating the platform. Nominal latency from input to robot motion was observed to be below 100 ms with no correlation to distance, with occasional spikes likely due to interference on the wireless channel.

4.3.1.1 Communication messages

The complete communication language has been defined in the Protocol Buffer domain-specific language, version proto2[6]. This gives interchangeable and native-feeling Python, C++ and JavaScript objects that take care of type conversion and (de-) serialization. The proto2 serialization format is optimized for size, which results in low latency and high throughput. A description of the message format and typical usage is presented here.

The parser assumes a known message type, so the Message message is a generic container for top-level messages. This global message includes fields to identify the sender (origin) uniquely, the sequence number and a timestamp of transmission. Each sender maintains the number of prior messages of own origin sent in this sequence number, it can serve a purpose similar to that of TCP sequence numbers. If messages are relayed or retransmitted they will maintain their original sender and timestamp.

The protocol allows for some complicated retransmission and synchronization behavior that is not yet utilized. In theory ensuring that all past messages arrive at all clients can lead to a world model that is forever consistent when updated deterministically. This is a very attractive thought as it enables consistent distributed decisions based on state without any leader election or mutual exclusion algorithms. But it also comes at the cost of maintaining a separate copy of the world model for distributed planning decisions, as it will update much more slowly. So for now only the provision is given by means of the Retransmit message type and the handling functionality is not yet realized.

[6]Protocol Buffers Language Guide, proto2, `https://developers.google.com/protocol-buffers/docs/proto`

The most frequently occurring message type will have to be the Update message. It contains updates to the world model state and corresponds directly with the respective data structures as containers. For each of the NavigationGraph, EntityMap, SpaceMap and CoverageMap there are add, update and delete lists. For NavigationGraph messages, edges lack update but have separate add and delete lists. These make full transmission of the world model data structures possible.

Then the robots will still want to communicate to each other on the highest level, for this the Command message has been provided. Commands are directed at a target, which is a robot's unique identifier. A subset of the commands is typically queued as tasks with individual ids in a schedule until they finish executing. The following commands are defined:

- **debug** Configures the target for some specific behavior useful for debugging purposes. The debug string will be parsed by the target.
- **clear** Immediately clears the current target's schedule, does not cancel any task already in progress.
- **cover** Schedules the target to execute coverage path planning over the specified region. The optional width and offset leaves space for multiple followers, the optional threshold determines the specific path.
- **explore** Schedules the target to execute the wall-following exploration algorithm within the specified region.
- **follow** Schedules the target to do formation control at a fixed distance from another robot.
- **home** Schedules the target to navigate independently to its charging base, and dock to charge.
- **join** Schedules the target to join a collaborative behavior, which may involve terminating the local manager.
- **leave** Schedules the target to resume independent behavior, after which the robot's own manager will independently fill the schedule.
- **move** Schedules the target to follow a specific trajectory.
- **navigate** Schedules the target to navigate independently to a waypoint.
- **restart** Tells the target to immediately stop and restart all services.
- **shutdown** Schedules the target to power down all hardware and disable itself. Manual action is required to resume.
- **start** Immediately asks the target to start the next task of the schedule or resume an interrupted task. If a time is provided, it will wait until the internal clock passes the specified time.
- **status** Immediately requests the current status of the target, for polling.
- **stop** Immediately ends the current task and stops further motion.
- **pause** Immediately interrupts the current task and stops further motion.

The robots also have a need to express themselves, to provide event-based feedback that does not fit the world model. This is done by means of Status messages, which consist of the following:

- **completed** Indicates that the currently queued task id has been completed. Includes the length of the remaining schedule and next task id, if any.
- **debug** Any text-based or JSON-formatted debug information for temporary use. These will never be sent unless the target is configured to do so by a debug command.
- **error** There was an error executing the current task and the robot was automatically paused. The task id and a status is provided, with a specific error description.
- **status** Response to a status request. Contains the current collaborative behavior, execution status, id of current task and an estimated time for completion, and the length of the schedule.

4.3.2 World model

The world model implementation contains the data structures $W = (W_S, W_E, W_N, W_C)$ from the design in Section 3.2.2. They are implemented by the following classes:

- **WorldModel** is the container W. In essense, an instance of WorldModel is a complete perspective from a single robot. It has methods for handling all updates from the `sensors` and the `network` modules.
- **NavigationGraph** represents W_N. It uses the NetworkX library[7] for the underlying graph representation.
- **EntityMap** represents W_E. Name-based lookup is implemented with hash tables using native Python dictionaries. Region-based lookup uses the SciPy KDTree library[8].
- **SpaceMap** represents W_S. Implementation is realized by a space- and computation-efficient NumPy ndarray[9].
- **CoverageMap** represents W_C. As it is also a logical grid like the SpaceMap W_S, NumPy ndarray is selected again for the data structure. Computation of partial contours is done with a custom iterative Marching Squares algorithm that uses linear interpolation.

Any Command type messages from the `network` module are relayed via a callback that was set by the `planner` module. All Update messages are consumed by the WorldModel class directly, they are decomposed and passed on to the respective data structure classes. After each update from the sensors is processed, heuristics decide whether Update messages must be sent to the network. This depends on preferred intervals, message priority and current network congestion, which ensures minimal latency on the propagation of network updates. As the network is not congested in the test setup, only the preferred interval is honored with

[7] NetworkX, `https://networkx.github.io/`

[8] KDTree by SciPy, `http://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html`

[9] ndarray by NumPy, `http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html`

all messages having equal priority. Currently all such intervals are configured at the default 0.1 s, or 10 Hz.

As of this writing, the implementation has not been completed so no evaluation can be done.

4.3.3 System monitor

The primary purpose of the system monitor is to inspect the state and data structures of all robots to the level that this is useful for debugging. But not all state that is interesting for debugging is automatically transmitted over the network, so this is also an interface to trigger certain debugging features and perform rudimentary control of the system. In essence it is an application-specific visualization of the network traffic, with several interactive controls.

The shared communication between the robots can be collected by simply connecting to any participating robot over the published Websocket. This subscribes the client to all communications. Packets that arrive will contain network Messages objects encoded with the Protocol Buffers proto2 language.

So all that is needed is a means to connect to the stream, decode those messages and present them in a visualization. This is done with a static HTML5 page using JavaScript and a Canvas element, which is served directly from the robot. Now any client with a modern web browser that is connected to the same network as the robots can visualize their data structures, and optionally control the robots.

For decrypting the binary proto2-encoded messages the protobuf.js library[10] is used. And since websockets by default do not support binary messages, a special binary datatype is introduced, which is offered by bytebuffer.js[11]. In addition, JavaScript's Number datatype used for all arithmetic does not allow all operations on 64-bit long integers, which are quite common in the proto2 language. So the Long datatype from long.js[12] is used instead. Connecting to the stream can then be done by initiating the Websocket and changing the underlying datatype. Each time a Websocket event arrives, the first two bytes represent the length of the message which is read in its entirety. Using the message specification from Section 4.3.1.1, protobuf.js then parses the message into a native JavaScript object. This makes all of the prior defined messages available on the client, without writing any wrappers in JavaScript.

Visualization of the messages is simplified greatly by using the D3.js library[13]. All data elements can be shown overlaid on a map of the environment, like the office map of Figure 4.7. The D3 Floor Plan library[14] ofers pan- and zoom functions with layer selection. Layers exist for each of the four main data structures, and a selection can limit updates to one robot. A text window on the bottom provides a log-like summary of the incoming data and shows selected debug information. All command messages can be sent to the robot directly from the interface, for which the JQuery UI library[15] provides convenient functions and also helps to make the interface touch-friendly. These control buttons turn the the `monitor` module

[10] protobuf.js by Daniel Wirtz, `https://github.com/dcodeIO/protobuf.js`
[11] bytebuffer.js by Daniel Wirtz, `https://github.com/dcodeIO/bytebuffer.js`
[12] long.js by Daniel Wirtz, `https://github.com/dcodeIO/Long.js`
[13] D3.js by Mike Bostock, `https://d3js.org/`
[14] D3 Floor Plan by David Ciarletta, `http://dciarletta.github.io/d3-floorplan/`
[15] JQuery UI by Scott González, `http://jqueryui.com`

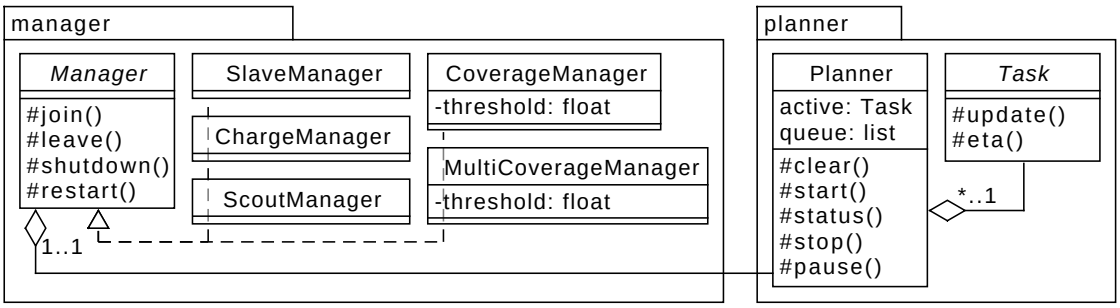

Figure 4.10: UML class diagram of the high-level system modules.

into a simple but manual version of the `manager` module. Having these versatile libraries available provides much flexibility, which is useful for quickly visualizing otherwise numeric debug messages.

The mix of modern technologies and libraries makes this interface feature-rich and useful, yet without significant cost for development. As an added benefit, it enables control from a wide range of platforms, including tablets, while remaining completely independent from the rest of the robot's codebase.

4.4 High-level system

The preceding work has defined the world to the point where various high-level algorithms can be described. There is a clear separation between the previous implementation work and the work that follows. Before the focus was on sensor performance, which has now been fixed by the performance of the dependencies. From this point on, the focus moves toward goal effectiveness, which is expressed in units that depend on the specific goal that the system is assigned. Due to the incompleteness of both the `controller` module and the `worldmodel` module at the time of writing, implementation of functional algorithms is not yet feasible. Therefore, the focus in this section will lie on the framework to realize such implementations.

4.4.1 Motion planning algorithms

These are defined in the `planner` module. Two classes from Figure 4.10 form the framework for all algorithm implementations.

The *Planner* class is a per-robot queue to manage the execution of tasks. It has methods to handle the corresponding Command messages, documented in Section 4.3.1.1, and registers with the `worldmodel` module to receive these. The debug, join, leave, shutdown and restart commands are forwarded to the `manager` module. The remaining non-task commands manage the queue, consisting of clear, start, status, stop, and pause. The Planner class has corresponding methods that can also be called directly as the `manager` module deems fit. It has a field for the current active task and a FIFO queue for the scheduled tasks, implemented as a Python list.

The *Task* class is an abstract base class whose instances represent tasks for the robot to execute. Its methods and properties provide a standardized interface for interacting with the rest of the robot. Motion planning classes inherit the Task class, of which the behavior is fully configured during instantiation. Of particular note is the task's *update* method, which returns a section of the path in proximity of the current position; a short sequence of setpoints for the controller. There is a mismatch as the planner recomputes new setpoints at a rate of 10 Hz, but the controller module responds at 1000 Hz. So instead the planner gives a partial path that is longer than the controller can possibly traverse at maximum velocity within one period, even considering position error. And the controller can determine the optimal course during the entire next period from this local plan, irrespective of minor oscillations in position. The task also has an *eta* method which gives a rough and quick estimate of the remaining time. This estimate is only used incidentally and need not be accurate.

At the start of execution of a task, that task becomes a Planner's active task and is removed from the task queue. The active task's update method is called at 10 Hz. This continues until the end of the task is reached, or until the task is interrupted.

The motion planning algorithms to handle the tasks of move, follow, navigate, home, explore, and cover have not yet been implemented.

4.4.2 Orchestration

Orchestration of all robots is handled by the `manager` module. The main purpose is to handle all dynamic events that may occur to guarantee completion of the goal. Different types of managers are supported, of which the *Manager* class is the abstract base class. This class receives and handles five types of Command messages from the `planner` module: debug, join, leave, shutdown, and restart.

Managers are like use cases for the robot, they are a composition of tasks toward a certain goal. Several managers are defined to handle the most generic high-level behavior, as can be seen in Figure 4.10. Note that no definite behavior is enforced by this system and it is designed for extension.

Behavior of managers is typically to keep the Planner queue filled with tasks. And if any exceptions occur, to find a workaround for the cause and subsequently re-plan the tasks. This repeats until a fatal error occurs or the goal is completed.

- **ChargeManager** keeps sending the robot back to the charging base and keeps it charging forever. It is the default manager of the system.
- **SlaveManager** does nothing except handle the Command messages. It ensures the system does not interfere with external managers.
- **ScoutManager** continuously explores the area by following all walls around all nodes of the navigation graph. After it completes visiting all nodes, the process repeats. This ensures that the map is continuously kept updated.

- **CoverageManager** performs continuous coverage path planning with a single robot. A coverage threshold is determined based on the current minimum observed in the scene. After this value is reached, the coverage threshold is increased to re-start the process. The system does not terminate until the battery runs out.
- **MultiCoverageManager** puts all detected robots except itself on their slave manager. It alternates performing coverage path planning with formation control in large regions, and distributed coverage path planning for remaining regions. Similar to the CoverageManager, the coverage threshold is incrementally increased and the system does not terminate until the battery of the manager itself runs out. At which point it sends all robots the leave Command message and lets the default manager take over for itself.

The leave Command message replaces the current manager by the default manager. The join Command message replaces the current manager by a SlaveManager which allows another manager to control it. When the platform indicates a low battery alert, the current manager will be replaced by the ChargeManager. Currently other managers can not be started during operation, they require manual startup.

The collection of managers can be seen as a state machine like Figure 4.11, where each robot with a manager assigned is a state. A system of multiple robots exhibits a parallel composition of the state space, for which the join and leave Command messages allow collapsing the state to a known value. There are some limitations to such a setup, but the most important thing is that the system can be effectively described with process algebras for analysis. And the restriction to a simple conditional state machine on the highest level makes this possible.

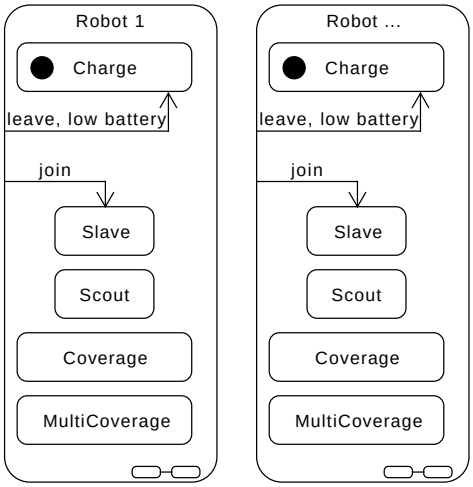

Figure 4.11: UML state diagram for a multi-robot configuration of managers.

These managers have not been implemented yet and therefore cannot be evaluated for total system performance. A metric of performance for the CoverageManager and MultiCoverageManager would be the amount of time that passed in a fixed region until some minimum coverage certainty is reached. Statistics of the percentage of time spent doing certain tasks and the amount of area covered over time with various strategies would give much insight.

Chapter 5

Conclusions

The main goal of this thesis is to realize a low-cost robotics platform for the purpose of evaluating cooperative motion control algorithms, specifically area coverage. This problem is broken down into several requirements in Section 1.1 and subgoals in Section 1.2 that specify the solution in detail.

In particular, area coverage algorithms often assume localization to be ideal, but no single sensor exists that comes close. Algorithm performance can be reduced to localization accuracy, which can only be achieved by integration of many contributing sensors. A hardware selection then leads the design and implementation process.

Novel techniques are possible when the localization error has become smaller than the area covered by the robot. The most significant of which is a shared view of the world, to which each robot contributes measurements. As a consequence algorithms are no longer restricted to a single robot's observations.

But the remaining localization error, no matter how small, still requires special considerations in many of the motion planning algorithms. This principle of position uncertainty guides the behavioral design which is used to ensure a practical solution.

5.1 Summary of Results

This work has succeeded in presenting a thorough design of a versatile robotics platform for the purpose of algorithmic evaluation of multi-robot systems. The design is heavily optimized toward achieving high positioning accuracy, but results that show the combined accuracy by evaluation are unfortunately not yet available. The goals that were set in Section 1.2 have been met for the largest part.

The first goal was to deliver a suitable hardware design. Research in Chapter 2 evaluates the many aspects of various solutions. Relevant selections are made that optimize toward low cost, reliability, performance, and capability. The detailed design that is proposed in Section 3.1 offers a wide variety of sensors leading to comparatively high environmental awareness at a price that is lower than the cost of many individual sensors currently on the market. This includes ultra-wideband radio, rotational scanning Lidar, a high-resolution camera, inertial

sensors (two IMUs), a digital compass, contact bumpers and wheel encoders, at a unit cost of approximately €500. These sensors follow modern and robust technologies that have low computational requirements, suitable for embedded real-time solutions. Furthermore, the construction of a single prototype including the necessary infrastructure is completed in Section 4.1.

The second goal was to deliver a design for sensor fusion and trajectory tracking. Research on various sensor fusion methods in Section 2.1.5 shows that an extended Kalman filter is suitable for a real-time position estimate if certain properties of the measurements are optimized. These concern the assumption of zero latency, the sensor distribution being Gaussian with a tight error estimate, and fully determined internal state variables to counter for instability from bias. Trajectory tracking is often an integral part of motion planning, but under position error direct control becomes invalid. The solution researched in Section 2.2.1 and described in Section 3.3.2 involves a local potential field that prefers following the path's direction, with orientation corrections increasing with the distance from the path. This approach allows all motion planning to be oblivious to the motion constraints specific for unicycle mobile robots. But these algorithms have not been implemented and no evaluation has been done.

The third goal was to design for system performance, which is derived to minimizing localization error and measurement latency. System performance has been a big concern throughout the design and implementation. The software architecture assigns performance requirements and priorities to a modular subdivision, of which individual modules have seen much optimization. The UWB driver from Section 4.1.5 has seen extensive analysis and an offline schedule is computed and verified. The platform driver and the IMU (Sections 4.1.4 and 4.1.3) have achieved the optimal measurement rates and latency possible for the hardware. Practical tests show that individual timing characteristics fall within expectation, which validates the schedules and shows that the implementation is sufficient. The low measurement latency is an important requirement for minimizing localization error and achieving high position accuracy. Their combined efficiency and latency in the sensor fusion design has also seen careful analysis in the controller implementation.

The fourth goal was to realize cooperative data structures and communication. In Section 3.2.2 data structures have been designed that match closely with the problems that are to be solved. The grid-like space map is used for path planning and obstacle avoidance, and the grid-like coverage map stores completion state of area coverage. The geospatial entity map stores accurate positions of arbitrary objects in the scene, and the graph-based navigation graph represents a cellular decomposition that offers fast complete navigation for long distances, and can assist area coverage. Simultaneously the requirement of communicating these data structures in a distributed manner is taken into account. The space map has a reliability score to merge updates, the coverage map uses commutativity of the maximum operator, the navigation graph depends on a commutative replicated data type (CDRT) for graphs, and the entity map uses managed ownership constraints. The networking implementation from Section 4.3.1 guarantees reliability and keeps and latency low, but assumes a lack of congestion. The communication protocol from Section 4.3.1.1 keeps payload size small.

The fifth goal was to perform unambiguous localization in the world frame. This goal has been met with the successful implementation of the UWB infrastructure and driver from Section 4.1.5, which provides the full pose as absolute position and orientation. However the properties of this system alone still do not satisfy the speed and accuracy requirements for the

data structures. The design for a low-latency method of localization is given by the Spring UWB ranging localization algorithm from Section 3.3.1, for which the required timings have been experimentally verified in Section 4.1.5. This makes it suitable for use with sensor fusion as described in Section 2.1.5. An implementation of this sensor fusion is expected to exceed the requirements for localization accuracy.

The sixth and final goal was to design behavioral algorithms for navigation and area coverage. Several behavioral algorithms are defined as high-level tasks in Section 4.4.1. A local potential field enables path planning that is oblivious to the constraints in motion that unicycle mobile robots experience. The approach for navigation in Section 3.3.3 uses a combination of an AD* search in an obstacle grid and graph searches on generalized Voronoi diagrams, which enables fast real-time and complete navigation in a cooperative system. A practical solution for the Coverage Path Planning problem has been proposed in Section 3.3.4, that uses a spiral pattern by following contours directly derived from a shared probabilistic map of coverage. A strategy for formation control is presented in Section 3.3.5 that does not depend on low-latency reliable communication channels. This strategy is compatible with the coverage path planning solution and can turn it into the multi-agent system envisioned in the introduction. Furthermore, the supporting system is designed for environmental awareness. The practical consequences of environmental uncertainty are taken care of by the algorithm design. The resulting high-level data structures fulfill the prerequisites for future distributed algorithms. They convey significant environmental information useful for complex algorithmic implementations, close to what can only be achieved in simulations. And many of the concerns with integration of these details are already solved in Chapter 4, such as data rates, processing times, and consistency. In addition, when the budget was cut concern was given to the ability to test distributed algorithms even when multiple physical robots are not present. This resulted in both a virtual simulation platform and a method to realize live visualizations of the shared data structures. Both of which are useful for accelerated algorithmic development.

In conclusion, the presented system meets most of the goals that are set. The extensive design solves many practical considerations and shows feasibility of the complete solution. A substantial amount of the implementation has been completed and several individual modules have been tested. A full sensor schedule is validated and tested, which enables all sensors to operate simultaneously at maximum data rates under minimized latency. And finally, the resulting platform is generic and extensible for future applications in research.

5.2 Future work

The implementation is still incomplete and some parts need to be finished:

- The controller's Kalman filter needs to be implemented while ensuring realtime performance.
- The Navigation Graph's online iterative construction needs to be implemented and verified.
- Most world model update methods need to be implemented, especially concerning data transformations from the sensors.
- The planner's task algorithms for move, follow, navigate, home, explore, and cover.
- The planner's queue mechanism requires better integration.
- The different managers need to be written and proper strategies need to be chosen that guarantee completion.

The system also requires some serious evaluation. Previously this thesis had a verification chapter, of which the results have now been incorporated elsewhere. It could contain at least the following extra points:

- A structured approach to latency analysis on a module basis, finishing with end-to-end latency analysis.
- Statistical characteristics of each sensor, to optimally tune the Kalman filter in the controller.
- A characterization of dynamics of the system, useful for optimizing velocity profiles to prevent system vibrations.
- Controller performance optimization, which has many aspects.
- Algorithmic performance under various scenarios, first of individual tasks and then of each of the different managers.
- The evaluation of algorithm performance will likely hint at many problems caused by communications latency. Investigating these problems will likely lead to a recommendation on algorithmic best practices and a better understanding of the practical performance of comparable distributed systems.

Next are possible options for more involved optimizations that may involve hardware changes:

- Using two accelerometers with different configurations. Connect a BNO055 IMU directly to the Raspberry Pi in addition to the MPU6050 IMU, which would reduce especially orientation latency significantly.
- Re-implement the Pozyx API on the Raspberry Pi for direct communications, to circumvent the additional latency of using the Arduino MPU.
- Turn the schedule for the Pozyx IMU into an online version instead of offline, which still correctly prioritizes heading requests but reduces the latency of range requests further.

- Full use of SLAM techniques with the Lidar for relative pose corrections.

And finally follow some more involved and experimental ideas:

- UWB localization in a multi-robot system can work much better as TDoA transmissions intended for some tags can also be used on all tags. This involves a whole redesign of the UWB RTLS architecture dedicated to multi-robot performance.
- Each robot's UWB tag can also act as an anchor if their position and position error are properly known. This may lead to significantly higher positioning rates as the UWB network becomes busier.
- Tuning of the cover task to incorporate specific shapes of the robot's brush, but only near obstacles. This will result in a higher true coverage percentage for generic robots with complex shapes.
- Break the assumption of a reliable network and optimize the type and volume of traffic given dynamic network performance, as is typical of many WiFi networks.
- Incorporate aspects of truly distributed systems, where no central manager exists yet actions are still cooperative, possibly without explicit communication.

Bibliography

[1] Ercan U. Acar, Howie Choset, and Prasad N. Atkar. "Complete sensor-based coverage with extended-range detectors: A hierarchical decomposition in terms of critical points and Voronoi diagrams". In: *International Conference on Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ*. Vol. 3. IEEE, 2001, pp. 1305–1311.

[2] Ercan U. Acar et al. "Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods". In: *The International journal of robotics research* 22.7-8 (2003), pp. 441–466.

[3] Sung-Joon Ahn. "Geometric fitting of parametric curves and surfaces". In: *Journal of Information Processing Systems* 4.4 (Dec. 31, 2008), pp. 153–158. ISSN: 1976-913X. DOI: 10.3745/JIPS.2008.4.4.153.

[4] Esther M. Arkin et al. "Optimal covering tours with turn costs". In: *SIAM Journal on Computing* 35.3 (2005), pp. 531–566.

[5] Tim Bailey and Hugh Durrant-Whyte. "Simultaneous localization and mapping (SLAM): Part II". In: *IEEE Robotics & Automation Magazine* 13.3 (2006), pp. 108–117.

[6] A. Benini et al. "A biased extended Kalman filter for indoor localization of a mobile agent using low-cost IMU and UWB wireless sensor network". In: *Tc* 1 (C2 2012), p. 4.

[7] Charles Boucher and Robert Lensch. *MEMS Technology*. Boucher-Lensch Associates, 2010.

[8] Yang Quan Chen and Zhongmin Wang. "Formation control: a review and a new consideration". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 3181–3186.

[9] Y. H. Choi et al. "Online complete coverage path planning for mobile robots based on linked spiral paths using constrained inverse distance transform". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2009, pp. 5788–5793. DOI: 10.1109/IROS.2009.5354499.

[10] Howie Choset and Philippe Pignon. "Coverage path planning: The Boustrophedon cellular decomposition". In: *Field and Service Robotics*. Springer, 1998, pp. 203–209.

[11] Sanjiban Choudhury and Sanjiv Singh. *A coverage planning algorithm for agricultural robots*. IIT Kharagpur, India, 2009.

[12] Thomas Clausen and Philippe Jacquet. *Optimized link state routing protocol (OLSR)*. 2003.

[13] S. A. M. Coenen and M. Maarten Steinbuch. "Motion planning for mobile robots—A guide". Master Thesis, Mechanical Engineering, Eindhoven University of Technology, Eindhoven, 2012.

[14] Jorge Cortes et al. “Coverage control for mobile sensing networks”. In: *IEEE International Conference on Robotics and Automation, 2002. Proceedings. ICRA'02.* Vol. 2. IEEE, 2002, pp. 1327–1332.

[15] Paraskevas Dunias. *Autonomous robots using artificial potential fields*. 1996. 120 pp. ISBN: 978-90-386-0200-4.

[16] T. Fraichard and A. Scheuer. “From Reeds and Shepp's to continuous-curvature paths”. In: *IEEE Transactions on Robotics* 20.6 (Dec. 2004), pp. 1025–1035. ISSN: 1552-3098. DOI: 10.1109/TRO.2004.833789.

[17] Carmelo Di Franco and Giorgio Buttazzo. “Energy-aware coverage path planning of UAVs”. In: IEEE, Apr. 2015, pp. 111–117. ISBN: 978-1-4673-6991-6. DOI: 10.1109/ICARSC.2015.17.

[18] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. “Visual simultaneous localization and mapping: a survey”. In: *Artificial Intelligence Review* 43.1 (2015), pp. 55–81.

[19] Enric Galceran and Marc Carreras. “A survey on coverage path planning for robotics”. In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1258–1276.

[20] Enric Galceran et al. “Coverage path planning with real-time replanning and surface reconstruction for inspection of three-dimensional underwater structures using autonomous underwater vehicles”. In: *Journal of Field Robotics* 32.7 (Oct. 2015), pp. 952–983. ISSN: 15564959. DOI: 10.1002/rob.21554.

[21] Santiago Garrido et al. “Robot navigation using tube skeletons and fast marching”. In: *Advanced Robotics, 2009. ICAR 2009. International Conference on.* IEEE, 2009, pp. 1–7.

[22] S. Garrido-Jurado et al. “Automatic generation and detection of highly reliable fiducial markers under occlusion”. In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2014.01.005.

[23] Shuzhi S. Ge and Yun J. Cui. “Dynamic motion planning for mobile robots using potential field method”. In: *Autonomous Robots* 13.3 (2002), pp. 207–222.

[24] F. Gulmammadov. “Analysis, modeling and compensation of bias drift in MEMS inertial sensors”. In: *4th International Conference on Recent Advances in Space Technologies, 2009. RAST '09.* 4th International Conference on Recent Advances in Space Technologies, 2009. RAST '09. June 2009, pp. 591–596. DOI: 10.1109/RAST.2009.5158260.

[25] K. R. Guruprasad and T. D. Ranjitha. “ST-CTC: a spanning tree-based competitive and truly complete coverage algorithm for mobile robots”. In: ACM Press, 2015, pp. 1–6. ISBN: 978-1-4503-3356-6. DOI: 10.1145/2783449.2783492.

[26] Guangjie Han et al. “Localization algorithms of wireless sensor networks: a survey”. In: *Telecommunication Systems* 52.4 (2013), pp. 2419–2436.

[27] Abraham Hart, Narek Pezeshkian, and Hoa Nguyen. “Mesh networking optimized for robotic teleoperation”. In: *SPIE Defense, Security, and Sensing.* International Society for Optics and Photonics, 2012, 83871E–83871E.

[28] Mahdi Hassan et al. “Task oriented area partitioning and allocation for optimal operation of multiple industrial robots in unstructured environments”. In: *Control Automation Robotics & Vision (ICARCV), 2014 13th International Conference on.* IEEE, 2014, pp. 1184–1189.

[29] Noam Hazon, Fabrizio Mieli, and Gal A. Kaminka. “Towards robust on-line multi-robot coverage”. In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on.* IEEE, 2006, pp. 1710–1715.

[30] Islam I. Hussein and Dusan M. Stipanovic. “Effective coverage control for mobile sensor networks with guaranteed collision avoidance”. In: *IEEE Transactions on Control Systems Technology* 15.4 (July 2007), pp. 642–657. ISSN: 1063-6536. DOI: 10.1109/TCST.2007.899155.

[31] Jung-Hoon Hwang, Ronald C. Arkin, and Dong-Soo Kwon. “Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control”. In: *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. Vol. 2. IEEE, 2003, pp. 1444–1449.

[32] Invensense Inc. *MPU-6000 and MPU-6050 product specification revision 3.4*. 2013. URL: https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf (visited on 08/13/2016).

[33] Sven Koenig and Maxim Likhachev. “Improved fast replanning for robot navigation in unknown terrain”. In: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 1. IEEE, 2002, pp. 968–975.

[34] Kurt Konolige et al. “A low-cost laser distance sensor”. In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 3002–3008.

[35] Yoram Koren and Johann Borenstein. “Potential field methods and their inherent limitations for mobile robot navigation”. In: *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*. IEEE, 1991, pp. 1398–1404.

[36] D. Kostić et al. “Collision-free tracking control of unicycle mobile robots”. In: *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*. IEEE, 2009, pp. 5667–5672.

[37] Joaquin D. Labrado et al. “Proposed testbed for the modeling and control of a system of autonomous vehicles”. In: *IEEE SoSE 2016* (2016).

[38] Lionel Lapierre, Rene Zapata, and Pascal Lepinay. “Simulatneous path following and obstacle avoidance control of a unicycle-type robot”. In: *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 2617–2622.

[39] Jean-Claude Latombe, Anthony Lazanas, and Shashank Shekhar. “Robot motion planning with uncertainty in control and sensing”. In: *Artificial Intelligence* 52.1 (1991), pp. 1–47.

[40] Steven M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[41] Jonathan K. Lawder and Peter J. H. King. “Querying multi-dimensional data indexed using the Hilbert space-filling curve”. In: *ACM Sigmod Record* 30.1 (2001), pp. 19–24.

[42] Maxim Likhachev et al. “Anytime dynamic A*: an anytime, replanning algorithm.” In: *ICAPS*. 2005, pp. 262–271.

[43] L. Ljung. “Asymptotic behavior of the extended Kalman filter as a parameter estimator for linear systems”. In: *IEEE Transactions on Automatic Control* 24.1 (Feb. 1979), pp. 36–50. ISSN: 0018-9286. DOI: 10.1109/TAC.1979.1101943.

[44] Chaomin Luo, Simon X. Yang, and Deborah A. Stacey. “Real-time path planning with deadlock avoidance of multiple cleaning robots”. In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Vol. 3. IEEE, 2003, pp. 4080–4085.

[45] R. Mahkovic and T. Slivnik. “Constructing the generalized local Voronoi diagram from laser range scanner data”. In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 30.6 (Nov. 2000), pp. 710–719. ISSN: 1083-4427. DOI: 10.1109/3468.895894.

[46] Eitan Marder-Eppstein et al. “The office marathon: Robust navigation in an indoor office environment”. In: *2010 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2010, pp. 300–307.

[47] S. Mastellone et al. “Formation control and collision avoidance for multi-agent non-holonomic systems: Theory and experiments”. In: *The International Journal of Robotics Research* 27.1 (Jan. 1, 2008), pp. 107–126. ISSN: 0278-3649. DOI: 10.1177/0278364907084441.

[48] David McNeil Mayhew. “Multi-rate sensor fusion for GPS navigation using Kalman filtering”. Virginia Polytechnic Institute and State University, 1999.

[49] Emmanuel Mazer, Juan Manuel Ahuactzin, and Pierre Bessiere. “The Ariadne’s clew algorithm”. In: *Journal of Artificial Intelligence Research* 9 (1998), pp. 295–316.

[50] James McCrae and Karan Singh. “Sketching piecewise clothoid curves”. In: *SBM*. 2008, pp. 1–8.

[51] Kenneth Meyer, Hugh L. Applewhite, and Frank A. Biocca. “A survey of position trackers”. In: *Presence: Teleoperators and Virtual Environments* 1.2 (Jan. 1, 1992), pp. 173–200. ISSN: 1054-7460. DOI: 10.1162/pres.1992.1.2.173.

[52] Suruz Miah. “Design and implementation of control techniques for differential drive mobile robots: An RFID approach”. Université d’Ottawa/University of Ottawa, 2012.

[53] Suruz Miah et al. “Nonuniform coverage control with stochastic intermittent communication”. In: *IEEE Transactions on Automatic Control* 60.7 (July 2015), pp. 1981–1986. ISSN: 0018-9286, 1558-2523. DOI: 10.1109/TAC.2014.2368233.

[54] Winston Nelson. “Continuous-curvature paths for autonomous vehicles”. In: *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*. IEEE, 1989, pp. 1260–1264.

[55] Esha D. Nerurkar and Stergios Roumeliotis. “Power-SLAM: a linear-complexity, anytime algorithm for SLAM”. In: *The International Journal of Robotics Research* (2011), p. 0278364910390539.

[56] Andreas Olofsson, Tomas Nordström, and Zain Ul-Abdin. “Kickstarting high-performance energy-efficient manycore architectures with Epiphany”. In: (Dec. 17, 2014). arXiv: 1412.5538 [cs].

[57] Udi Peless. *Robomow company website*. 2016. URL: http://robomow.com/ (visited on 08/29/2016).

[58] Ioannis Rekleitis et al. “Efficient Boustrophedon multi-robot coverage: an algorithmic approach”. In: *Annals of Mathematics and Artificial Intelligence* 52.2-4 (Apr. 2008), pp. 109–142. ISSN: 1012-2443, 1573-7470. DOI: 10.1007/s10472-009-9120-2.

[59] Anna Sadowska et al. “A virtual structure approach to formation control of unicycle mobile robots using mutual coupling”. In: *International Journal of Control* 84.11 (Nov. 2011), pp. 1886–1902. ISSN: 0020-7179, 1366-5820. DOI: 10.1080/00207179.2011.627686.

[60] Anna Sadowska et al. “Distributed formation control of unicycle robots”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1564–1569.

[61] T. Sanpechuda and L. Kovavisaruch. “A review of RFID localization: Applications and techniques”. In: *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2008. ECTI-CON 2008. 5th International Conference on*. Vol. 2. IEEE, 2008, pp. 769–772.

[62] Tom Schouwenaars, Jonathan How, and Eric Feron. “Receding horizon path planning with implicit safety guarantees”. In: *American Control Conference, 2004. Proceedings of the 2004*. Vol. 6. IEEE, 2004, pp. 5576–5581.

[63] Stephen Se, David Lowe, and Jim Little. “Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks”. In: *The international Journal of robotics Research* 21.8 (2002), pp. 735–758.

[64] Fernando Seco et al. “A survey of mathematical methods for indoor localization”. In: IEEE, Aug. 2009, pp. 9–14. ISBN: 978-1-4244-5057-2. DOI: 10.1109/WISP.2009.5286582.

[65] Marc Shapiro et al. “Conflict-free replicated data types”. In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by Xavier Défago, Franck Petit, and Vincent Villain. Red. by David Hutchison et al. Vol. 6976. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 386–400. ISBN: 978-3-642-24549-7 978-3-642-24550-3.

[66] Bruno Siciliano et al. *Robotics*. Red. by Michael J. Grimble and Michael A. Johnson. Advanced Textbooks in Control and Signal Processing. London: Springer London, 2009. ISBN: 978-1-84628-641-4 978-1-84628-642-1.

[67] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. Springer Science & Business Media, Aug. 20, 2010. 644 pp. ISBN: 978-1-84628-641-4.

[68] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[69] Christoph Sprunk. “Planning motion trajectories for mobile robots using splines”. In: *University of Freiburg* (2008).

[70] Anthony Stentz. “Optimal and efficient path planning for partially-known environments”. In: *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE, 1994, pp. 3310–3317.

[71] Rui-Jun Yan et al. “Natural corners-based SLAM with partial compatibility algorithm”. In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* (2014), p. 0959651814533533.

[72] S.X. Yang and C. Luo. “A neural network approach to complete coverage path planning”. In: *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)* 34.1 (Feb. 2004), pp. 718–724. ISSN: 1083-4419. DOI: 10.1109/TSMCB.2003.811769.

[73] Mohammadreza Yavari and Bradford G. Nickerson. “Ultra wideband wireless positioning systems”. In: *Dept. Faculty Comput. Sci., Univ. New Brunswick, Fredericton, NB, Canada, Tech. Rep. TR14-230* (2014).

[74] Risang Gatot Yudanto and Frederik Petre. “Sensor fusion for indoor navigation and tracking of automated guided vehicles”. In: *Indoor Positioning and Indoor Navigation (IPIN), 2015 International Conference on*. IEEE, 2015, pp. 1–8.

[75] G. X. Zhang. “A study on the Abbe principle and Abbe error”. In: *CIRP Annals - Manufacturing Technology* 38.1 (Jan. 1, 1989), pp. 525–528. ISSN: 0007-8506. DOI: 10.1016/S0007-8506(07)62760-7.

[76] Xiaoming Zheng et al. “Multi-robot forest coverage”. In: *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 3852–3857.