Eden Attenborough, Robert Stevenson, Tye Buckingham

Registration number 100301654, 100301804, 100271864

2023

Complete Coverage Within Virtual Boundaries for Autonomous Robots Using RTK-GNSS Positioning

Supervised by Dr Edwin Ren

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

Cutting the grass of large areas such as golf courses is a difficult and tedious task. Several robotic products to do this are already on the market, most of which use a buried guide wire for boundary definition. In this project, we present a prototype autonomous robot that uses RTK-GNSS for localisation, a companion mobile app for boundary definition, and a novel algorithm for coverage planning and obstacle avoidance for navigating a user specified a work area.

Acknowledgements

We would like to thank Dr. Edwin Ren for being our supervisor on this project. We would also like to thank CHC Tech in Norwich for providing us with an RTK chip and giving us use of their RTK base station.

Contents

1. Introduction

The ability to automate tedious and repetitive tasks has been a common problem, and the use of autonomous robots for performing these tasks has also been a common solution for these tasks. If we think about tasks such as, plant/crop watering, environment surveying, warehousing, and factory production lines they all involve some type of repetitive task that has either already seen robots used in these areas or are currently being explored. Applications for the automation of outdoor tasks in environments that require high precision for navigation are also growing in interest, particularly in the agricultural sector and businesses that have to maintain large outdoor environments. That is what this paper looks to explore by using RTK-GNSS positioning to give our robot platform centimetre level GPS precision. With this level of accuracy that we expect, we look at a range of products and scenarios and found that autonomous lawnmowers are an example of a product that could leverage this technology to perform a better job, and with clients/business like a golf course having large areas of greenery that would need frequent lawn maintenance as part of their business.

That is why this projects aims to create a robot that can use the high precision that RTK-GNSS can offer and perform accurate path navigation to carry out complete coverage of a pre-planned path using a purely virtual defined operating area, much like how a robot lawnmower would do to cut all the grass in its work zone. This means we also created an mobile application that we can use to define these operating perimeters, along with the ability to define areas that we are not allowed to enter, and a robot platform to dev and test with. This robot has to be equipped the capabilities to operate in an outdoor test environment and be able to properly orientate itself and make use of RTK-GNSS for localisation. Additionally, we look at the using LiDAR for reaction to undefined obstacles in the working areas and attempt to navigate the robot around them to reach its goal.

Therefore, this paper will give an overview of existing autonomous lawnmowers, with references and discussion about research related to a methods or technology being done during the relevant section of this paper. We will next be following up with details on the design and implementation of the robot platform, going into detail on the hardware and how we make use of ROS framework for the base of our system's software. After, we will explore the system's architecture, with information on the robot's architecture and the network architecture for the whole system. We then go into detail on the RTK-

GNSS localisation implementation, along with information of the use of sensor fusion with RTK. Our system application then follows next where we go into the details on core feature require for our proof-of-concept implementation and how we implemented these features, with the primary focus being on the user's ability to define the virtual boundaries by drawing on satellite images in the app. To follow this up, the next few sections look at coverage planning, path traversal, obstacle avoidance and navigation method required to carry out the complete coverage path of our defined area which we can then move on with discussing the test result. Finally, we can evaluate the functionality of this proof-of-concept system and determine the shortcomings and limitation with the techniques and hardware used by the current system, which we can then use to suggest how we could improve these faults.

2. Related Works

Using autonomous lawnmowers as the one of the core product examples that could benefit from the technologies used in this project, we can look at existing automatic lawnmowers to compare what we will be doing vs what currently exists.

2.1. Similar Systems

When look at the existing lawnmower, two of them being open-source, which are available we can we variety of different methods for defining boundaries that the mower should be operating, such as the use 'Guide Wires' which involves the user installing wire under the surface of their grass (with some implementation running a low current through them) which sensors on the robot can detect and use it detect the edges if their work zones and prevents the lawnmower from driving out of these boundaries. Another method which appears to be rather common is the '*Walk-around*' method that requires the user to drive the robot around the borders of the no-go and works zones in order to define these for later. We are not a big fan of this method as it would make deploying these robots in large scale environments, like a golf course, very cumbersome to deploy and could be hard to scale if we wanted to run fleets of these machines on an area.

A number of systems use one or more sensors to detect pre-existing physical boundaries, for example walls, fences, or paths. Once these boundaries have been detected, a virtual map of the area is presented to the user. Adjustments can then be made to the

map if needed. The main drawback of this method is the requirements for a well-defined physical boundary. If the area the user wishes to cut is not physically confined, then the lawnmower will continue moving until it finds one. This is generally fine for most home gardens which are generally confined by fences, but commercial land often has less limiting boundaries. When compared to other methods such as the '*Walk-around*' or 'Guide Wires' options, the work required by the user is much less, with no installation of physical boundaries required (if they exist already), and no need to walk the robot around, these methods provides an easier option for most users. Unfortunately, when considering commercial applications, it appears the installation of boundaries would be required. Unless the systems begin to allow the user to define what is a boundary for example, when the grass changes shade or height, this method is limited in scope.

When we looked for any that use 'Aerial Photography' (or 'Aerial Imagery'), whether that is using drone imagery or satellite images, we were unable to find a product that used this approach for perimeter/zone definitions. This is important as this is the approach that this paper's project looks to explore to see if whether we can use this to define a "pure" virtually perimeter that does not require any physical interaction with the robot to define the perimeter edges or need the user to setup/install any equipment for defining the perimeter (e.g. Guide wire/Perimeter wire).

If we also look at the method of obstacle avoidance that are used across existing systems, we see this more in higher-end product they typically have multiple sensing methods, whether that is LiDAR, Ultrasound sensors, Vision, Collision/Bump sensors and/or AI object detection. For our project we are only using LiDAR, which only appears in one other system out of those that we are reviewing in this paper, for detecting objects in front of our robot. The use of Computer Vision and AI detection is an interesting addition for carrying out obstacle avoidance, however it would require a lot of time to train these detection models and would also require our robot to be ale to process the data faster enough, which unfortunately we do not have hardware that is capable enough to offer any reasonable real-time AI processing.

You can find a summary comparison table for the lawnmowers (or the manufactures) boundary and obstacle avoidance methods within Table 1. What is interesting to us though is that out of the two open-source solution, 'ArduMower' and 'OpenMower', we see that (at the time of writing this paper) 'OpenMower' does not feature any form of obstacle avoidance according to their GitHub and can only avoid obstacle that were defined during the perimeter setup phase (ClemensElflein, 2023).

3. Design and Implementation of the Robot Platform

With this project, we have several core parts that come together to form the complete system. These core parts consist of; the physical robot platform (which is the focus of this section), our system and network architectures (Sec. 4), RTK-GNSS localisation (Sec. 5), a mobile application (Sec. 6), the coverage planner (Sec. 7), the path traversal and Navigation (Sec. 8 and Sec. 9). This section in particular will focus on creating a robot test platform that we can use to develop and test our project work so that we are capable of evaluating the method used in this project and determine if where we can further improve upon these ideas or need to find an alternative method of approach.

3.0.1. ROS2 - Robot Operating System

Over time, many software platforms have been proposed and even have grown to have developed a rich selection of tools and features, but few rival Robot Operating System (ROS 1) in its significance on the maturing robotics industry (Macenski et al., 2022). This is why we chose to design and implement our software solution to operate using the 'Robot Operating System (ROS)' framework for our project's robot, as it "has become influential in nearly every intelligent machine sector" and offer a large set of software libraries and tools for building robot applications (Macenski et al., 2022). We specially use 'ROS2' (Macenski et al., 2022), as is the latest (major) revision of the ROS API whilst taking advantage of more modern libraries and technologies in the core ROS functions, such as updating the targeted python version from 'Python 2' to 'Python 3.5 or later' and also changing the targeting for 'C++' from 'C++03' to 'C++11' (some parts from C++14). Additionally, it addresses the lack of support for real-time systems in the original ROS version of the framework by adding support for real-time code and embedded system hardware.

We also took into consideration that this project may be used for future testing and development that either extends the work of this paper or be extended with new features. By using ROS2 over ROS1, we ensure the software framework we are using and the software we have written has the highest probability of remaining comparable over the next few years (at least till "May 2027" according to the ROS2 distribution list for 'Humble Hawksbill'[1] as the last distribution for ROS1 ('Noetic Ninjemys') reaches

[1] *https://docs.ros.org/en/humble/Releases.html*

end-of-life (EOL) in “May 2025"[2], with no new distribution being released afterward this.

More on how we use ROS can be found later in this paper in Sec. 3.1.1, Sec. 4, Sec. 5.

3.1. Robot platform

To develop and test the coverage planning algorithm and the impact of using RTK-GNSS technologies for outdoor localisation, we required an autonomous robot platform that we will use these techniques on. That section will go into the design and construction of our system, with details on the hardware used, some tools and software framework used, and the system architecture.

3.1.1. Robot Hardware

The hardware used for our robot consists of:

- Devastator Tank Mobile Robot Platform[3]
- Raspberry Pi 4 (4GB)
- ESP32 Dev Module[4]
- BNO085 9-DOF IMU (Adafruit 9-DOF Orientation IMU Fusion Breakout - BNO085)
- ICM-20948 9-DOF IMU (SparkFun 9DoF IMU Breakout - ICM-20948)
- Cytron 3A 4-16V Dual Channel DC Motor Driver
- Geared DC Motor with Magnetic Encoder Outputs
- Roblox NEO-M8P (GPS-RTK module)

The list excluded information on the powering hardware, as the design for this lies out of scope for this paper and will only be discussed briefly in Sec. 4.

[2] *http:// wiki.ros.org/ Distributions*
[3] The 2nd revision of the chassis that came with ‘Metal DC Gear Motor’.
[4] 30 pin variant of the ESP32 dev board, also commonly referred as ‘DOIT ESP32 DEVKIT V1’.

We chose the 'Raspberry Pi 4' single board computer (SBC) for the main control boards of our system for a range of reason. The first being that it supports running the Linux operating system 'Ubuntu'. This is especially useful as it makes integrating and using 'ROS' in or project easier, as by using the 'Ubuntu Server' distribution we give as little resources as we can to the operating system and allows use to Maximise e the amount of processing resources for our application's software. This is particularly useful as it makes integrating and using 'ROS' in our project easier, as by using the 'Ubuntu Server' distribution we give as little resources as we can to the operating system and allows use to Maximise e the amount of processing resources for our application's software. Additionally, we wanted as much have as much RAM as possible for our system as the process from the coverage planning (more in Sec. 7) result in the path being provided a series of point that are used as virtual checkpoints and targets that the robot navigates with in order to perform the complete coverage off an area.

The chassis that we chose for our platform did come with its own set of DC motors, however earlier in the development of the project we saw that the use of motor encoders could be beneficial for two major reasons. The first is that we can use the encoder's feedback to calculate how much the motor has turned. Using this, in combination with some characteristics of the robot, we can provide our robot a target distance to move forwards (or backwards) with a precious down to millimetres (with an error +/- 10mm).

The second benefit is that we can use the encoder feedback to calculate the approximate revolutions-per-minute (RPM) of the motor, which we can use to control the speed of the robot more accurately, allowing for more consistent motor behaviour. This is performed by using a 'Proportional Integral Derivative' (PID) controller for controlling our motor speed. The use of PID controllers is popular for feedback control system and seen in use for a range of application within robotics (and other engineering related systems) for controlling a range of different signals/values to reach a target specific value (Åström and Murray, 2021; Blake et al., 2022; Philippart et al., 2019). This allows us to set a target RPM and get a required 'pulse-width modulation (PWM)' signal to send to the motor driver so that we can drive the motors with enough power to get them to rotate at the requested target speed.

We also found that midway through development, that the robot would easily get stuck when trying to rotate on the spot on surfaces that should not be challenging to run on (e.g. carpet, artificial grass and a moss covered floor), and would therefore not be able to operate for testing. This suggested that we need to increase the movement power

of our robot and therefore increase the power, or more specifically the torque, of the motors. From reading product information and the spec of the motor that came with the chassis and the ones we are using with motor encoders attached to the motor shaft, we can see that the motor that came with the chassis have a gear reduction ratio of “45:1" and the motor (with encoders) came with a gear ratio of “1:20". Through inspection of the motors and comparing their respective gearboxes, we found that we could swap the “45:1" gearbox on to our motors with the encoders install. This way the motors were capable of provider much more torque than before and the robot was no longer getting stuck when trying to turn on the easier operating environments/surfaces when running at the maximum rated voltage of 7.4V.

The robot is also equipped with two inertial measurement units (IMU), the BNO085 and the ICM-20948, are allows us to get the current robot’s orientation (more details on methods used can be found later in Sec. 3.2) and use it to reliable robot the robot to face a particular bearing, or by a specific number of rotation. In addition, we are capable of using the IMU data in combination with the RTK and LiDAR sensors to improve the Localisation techniques that used these sensor’s data when the robot it deployed in operation.

The ICM-20948 is connected and used by the ESP32 on the robot when the robot is sent a command to rotate to face a bearing, or if need we could send a command to rotate by a specific angle. The robot’s BNO085 IMU is connected directly to the Raspberry Pi via I^2C and is accessed thought a dedicated ROS node that publishes the IMU data to the ROS network at a rate of approximately 4Hz. For the BNO085 IMU’s I^2C connection with the Raspberry Pi, we were required to use the ‘Software I^2C’ implementation over the hardware I^2C implementation on the Raspberry Pi. This is because the Raspberry Pi has had a bug in its I2C implementation, which was discovered in 2013, that prevents I2C communication with some devices and leads to data corruption (both in read and write direction) when the connected device is reliant on clock stretching (Advamation, 2013). What significant is that despite the chipset used in the Raspberry Pi 4 being improved since 2013, the issue remains present on the SBC, with an open GitHub issue from 2022 still describing the bug (Raspberry Pi Foundation, 2022). Therefore, we use the ‘Software I^2C’ implementation to interface with the BNO085 because we experience crashes due to corrupt and/or missed data over the I^2C bus when relying on the hardware implementation.

3.2. Orientation

Through the use of both of our IMU sensor, which we introduced back in Sec. 3.1.1, to get data related to the current orientation of the robot. Primarily both IMU sensors are used to determine how much the robot rotates, such as the ICM-20948 used by the ESP32 that allows us the robot to execute rotation related commands by providing an angle for rotation, which when we are approaching our target orientation we accept an error of ± 1 for matching our current rotation to the target value. The robot's BNO085 IMU is used by the Raspberry Pi, as mentioned before, by the robot's path navigation programs to also provide the approximate heading of the robot (this heading is the bearing from North), but it also provides the robot's current linear acceleration and angular velocity vectors, in addition to the quaternion representation of the robot's orientation.

One of the advantages to using quaternions is that they are great at representing the current rotation state of a robot in 3D space and is targeted more at robots (along with other systems) that move freely in 3D space, instead of on a planar surface. By having a quaternion representation of the orientation, we can avoid running into 'Gimbal Lock'. Jung et al. (2013) explains 'Gimbal lock' as the condition when the IMU sensor "cannot uniquely represent the orientation by Euler angles". This occurs when "the pitch angle is near to ± 90 degrees" and as the angle approaches 90 degrees, the yaw, and roll values from the sensor move in that same direction (Jung et al., 2013). However, our system will have access and use both orientation representations. We do this by taking the quaternion representation and recalculating the Euler angles for the current orientation, as it is both more intuitive for humans to understand the orientation in terms of yaw, pitch, and roll angles, and we are also interested in the heading of the robot to determine to direct the robot is facing when operating in its environment. Because we are using the Euler representation we do re-introduces the issue of 'Gimbal Lock' for our robot, however because 'Gimbal Lock' only occurs then the pitch angle reaches 90 degrees the robot would have to be orientated pointing with the front (forward face) pointing directly up to the sky in which we would therefore experience a more serious mechanical and operation problem with the robot, and its operating environment.

3.2.1. IMU Processing

We get the quaternion representation for the orientation from combining the accelerometer, gyroscope, and magnetometer data from the IMU. It is common across other works,

like those from Ludwig (2018); Ludwig and Burnham (2018), to use fusion algorithms like the one proposed by Mahony et al. (2008) (commonly referred as 'Mahony Algorithm') or Madgwick et al. (2010)'s fusion algorithm (commonly referred as 'Madgwick Algorithm').

However, for this project we have chosen to use a different approach for getting the fussed outputs that the 'Mahony' or 'Madgwick' algorithms and instead try out the IMU's integrated fusion hardware. These features are typically found in more exercise IMU sensor units that are targeted towards industry and profession clients, but with the more affordable IMU sensors that are used in this project we see that these features have manage to become more accessible, with a lower barrier of entry.

The BNO085 runs the 'CEVA's SH-2' firmware that includes the 'MotionEngine' software that provides signal processing algorithms which process the sensor data and provide a precise real-time 3D orientation, heading, calibrated acceleration and angular velocity vectors (along with many more outputs) for easy access and use (CEVA, 2023). The ICM-20948 has its 'Digital Motion Processor (DMP)' firmware that does a similar job at processing the IMU data as the BNO085 so that we can also more easily obtain the 3D orientation, heading, calibrated acceleration and angular velocity vectors from this sensor. The BNO085 and ICM-20948 sensor do require calibrating to be able to use them effectively. This calibrating is required to counter the poor, or sometime missing, factory calibration found on low-cost IMUs that results in the measures from these sensors being coupled with non-negligible systematic errors (Tedaldi et al., 2013). A more commercial IMU, like XSens's 'MTi-10/100' series, come with a factory calibration. This factory calibration typically means the IMU is sold with its calibration parameter already calculated and stored in the sensor firmware or EEPROM (or some other form of non-volatile memory) allowing for accurate measures from off-the-shelf because the calibrated output of the sensor has been verified with known references (Tedaldi et al., 2013). However, this factory calibration process generally requires additional time and high-cost equipment during manufacturing, along with the conducted the calibration under a clean and perturbation free environment (Tedaldi et al., 2013; Peng et al., 2022). However, both the BNO085 and ICM-20948 respective firmwares can perform automatic calibrations for their sensors and can therefore adjust their calibration parameters based on their surrounding operating environments, and the robot is self. This is especially important as the robot will generate interfering signals that will have an influence on the IMU's sensor that can lead to inaccuracies, and even drifting in values, should

the sensor not be correctly calibrated. So with the auto calibration running full-time the robot is capable of adjusting to the changes in real-time and as a result should be able to counter the impact from the sudden changes.

4. System Architecture

With the size of this project, we have several subsystems that make up the complete system. The simplest separation of this complete system that allows use to divide and tackle its challenges is to look at the architecture for the physical robot platform that was developed for testing, and the network architecture that allows all the other pieces in the system (The robot, the processing server, and mobile application) to communicate with each other.

The challenges of the system require there to be a server-side aspect of the client phone app, which saves and manages user-defined areas through an HTTP API. We have implemented a server-side application to manage this, written in Python Flask. User data is stored in a MySQL database. An entity relationship diagram for the database structure is shown in Fig. 3. There is also a docker container that runs ROS2.

To aid in deployment and isolation of server-side tasks, we use a technology called *Docker*. Docker provides OS-level virtualization, in which separate tasks of the server-side system are isolated from each other (Ratan, 2017). Docker also aids deployment: the entire system can be deployed with a single `docker-compose` directive file. It also aids networking, since networking can be controlled by docker. This is shown in Sec. 4.1.

Fig. 2 depicts the robot's system architecture for the hardware that was described in Sec. 3.1.1. With the Raspberry Pi connected and communicating with, the ESP32 to control the robot use serial commands, receiving the LiDAR's scan data, the RTK-GNSS chipset providing the positioning data of the robot, the IMU orientation and sensor data, and finally the 4G network adapter.

The ESP32 is connected to the ICM-20948 IMU, our motor driver board, and the motor encoders. This way the ESP32 can avoid a bi-direction communication chain of information in the Raspberry Pi for the BNO085 and instead only need to use the ICM-20948 for determining how much of an angle the robot has rotated. The encoders are connected as interrupts to the ESP32 to allow us to create the close-loop feedback control loop for the robot's motor speed, as mentioned in Sec. 3.1.1, which then lead

to the ESP32 using the motor driver to control the motor speed by sending the PWM signals to the robot.

Powering the robot is relatively simple considering the level of hardware we are using. With the few embedded sensors, e.g. ICM-20948 IMU, have very lower power requirement and are powered directly from the ESP32's 3.3V pin. The only piece of hardware that the ESP32 provided power to the motor's encoder which, like the ICM-20948, pull very little current and is therefore does not require a dedicated power circuit. The motor driver is connected to a power pack consisting of two Samsung 18650 lithium-ion cells that run in series with a capacity of 2150mAh with the cell rated for a max discharge current of 10A (Continuous discharge), and its voltage is regulated down to 7.4V so that we can drive the motors at their highest rated power and maximise the speed and torque available to the motors. Using the datasheet of our motors (Fig. 1) we can see that our motors are rated at a max stall current of '$\leqslant$ 2A' per motor. Therefore, the cells are more than capable to handle these motors and our driver board is also capable of providing 3A (continuous) per channel/motor (5A peak). Additionally, there is a 5A fuse between the batteries and primary power switch so that we minimise the potential of damaging our components in the event that too much current is drawn. The final safety measure we have is that the robot includes two power switches in the circuit, the primary switch that is located on the top of the robot and the secondary switch connected directly to the battery holder. This dual switch setup was chosen for our platform to avoid accidentally powering the system on during transport when carrying the robot to various test location and therefore discharging the robot's batteries too far when the robot is to being used/monitored, but also reduced the risk of the robot accidentally driving when preforming over-the-air code updates when developing and testing the robot.

The Raspberry Pi's connected USB devices (LiDAR, ESP32, 4G dongle and RTK-GNSS chipset) and BNO085 (connected via I2C) are powered from a 10000mAh battery bank. This made powering of these devices simpler for the current level of the project, and we are getting power from a clean power source that is not receiving any noise through the power rails from components like the motors.

4.1. Network Architecture

The system is designed in such a way that at the centre there is a central server that receives HTTP requests from the client's mobile app, stores them, and generates the map.

This generated map can then be sent to the robot. However, this presents challenges. The server runs its own instance of ROS2, but since the robot is connected to a 4G dongle it will be on a separate network to the server. Since ROS2 nodes on the same network can communicate natively, with the aid of a Fast DDS discovery server (Open Robotics, 2023), we have implemented a WireGuard VPN that the robot and server will connect to on boot time. This enables seamless communications between the two as if they were on the same network. The VPN connections are shown as dotted lines in Fig. 4. On the server, there is a docker container that acts as a VPN client, this container's network stack is duplicated by the ROS2 docker container.

5. Localisation

5.1. Real-Time Kinematics

At the core of our project is our localisation implementation. We use a system of real-time kinematic positioning (RTK) in order to localise our robot. RTK is a system by which 'corrections' from a static base station are sent to a moving GNSS receiver on a moving robot, or 'rover'. The GNSS system alone has a level of accuracy of multiple meters, by using a base station with a long survey period to correct ionospheric and tropospheric delays, as well as multipath and ephemeris errors, RTK is theoretically able to achieve centimetre-level accuracy (Mannings, 2008) (Chen, 2014). A diagram showing the RTK methodology is shown in Fig. 5.

RTKLIB, Takasu et al. (2007), Takasu (2009), and Takasu and Yasuda (2009), is an open-source GNSS toolkit that can be used to do real-time RTK localization. It is the framework from which we do our RTK localizations with.

In our implementation, RTKLIB must integrate into our existing ROS2 framework. We are aware of two libraries that implement RTKLIB in ROS. Stahl (2013) and Ferreira et al. (2020). Of these, Stahl (2013) only works in older editions of ROS and hasn't been updated in the last ten years. Ferreira et al. (2020) is an excellent project, but since it was made as an academic project it was made only for ROS 1 with no fork for ROS2. This doesn't make it useful for our purposes.

Consequently, we decided to make our own wrapper implementation, which eventually proved trivial. Our implementation creates a thread to run the RTKLIB RTKRCV process, with the main thread using the TELNET protocol (Carr, 1969).

`nmea_navsat_driver` (Perko et al., 2013) is used to read the resulting NMEA message and converts it into ROS standard messages (specifically `sensor_msgs/NavSatFix`) this thankfully is a library that was recently forked to work in ROS2. Libraries not working in ROS2 was a common problem in our project.

Our RTK localization was thoroughly tested. This process is described in Sec. 11.1.

5.2. Sensor Fusion

We were disappointed with the accuracy of RTK alone. So we investigated whether it is possible to supplement the localisation estimate with additional sensors, a process known as *sensor fusion*.

Sensor fusion is a common task in the field of robotics. It typically involves using an *Extended Kalman Filter (EKF)*. The extended Kalman filter is an adaption of the traditional Kalman filter that can handle non-linear estimators (such as data provided by a GNSS sensor). In the *update* portion, information from multiple sensors can be merged together (Thrun et al., 2005).

Kalman filtering can be considered analogous to the *hidden Markov chain model* (Roweis and Ghahramani, 1999), (Hamilton, 2020) (See Fig. 26).

There is no shortage of literature about using extended Kalman filters for outdoor localization- since it is a common task in the field of car localization, which is important for GPS navigation systems and self-driving cars, both huge fields.

Castanedo (2013) is an excellent work that describes the different version of sensor fusion opportunities. Liu et al. (2021) uses extended Kalman filters in conjunction with an IMU, RTK and a LiDAR in order to do localization in the context of vehicles. Cai et al. (2018) adds visual odometry with a camera, also in the context of cameras. Most relevant, however, is Cai et al. (2019), who use GPS, IMU, wheel encoders and Visual Odometry in their localization estimate, in the context of robotics.

When it comes to ROS, the 'state of the art' is Moore and Stouch (2016), an advanced localization library using extended Kalman filters, and an almost infinite number of sensors (provided they can give messages in standard ROS formats) and has recently been ported to work with ROS2. If we were to implement this in our robot, the workflow would be as described in Fig. 27.

Unfortunately, sensor fusion with only a GPS and an IMU 'may not be especially useful' (Moore and Stouch, 2016), so without additional hardware changes, sensor fusion

is out of scope for this project. Hardware changes would primarily include the inclusion of wheel encoders to provide a source of odometry (the most critical change), additionally visual odometry with a camera (with placed location markers or landmarks visible in the physical environment), or a higher-end IMU. We do not think LiDAR would help the localisation estimate in this outdoor context. A novel idea would be to put a camera in a high-up location away from the robot that monitors the robot's location so that it works even in dark conditions, an infra-red reflector could be used to pinpoint the location precisely.

6. Companion Application

To facilitate the definition of virtual boundaries, a mobile application was developed. The application lacks features that would be seen in a commercial release such as scheduling, setting blade height, and telemetry. It was decided to not focus on such features as these were not the main focus of this project, instead focusing on the definition of virtual boundaries. The application uses the Google Maps API to produce a map view for the user to visualise the area they are defining. In Fig. 12 a golf course hole has been chosen to demonstrate the view when defining an area. The user can zoom and pan when editing boundaries, along with undo and delete options. It became clear that these options were necessary due to the difficulty of being accurate when defining boundaries using a small touchscreen device. Again, in a fuller release of such an application a more complete system for drawing boundaries, allowing different line types and capabilities to edit lines or points on the line, would be required. The functionality is suitable to demonstrate the premise of virtual boundaries in the context of this project.

Along with the difficulties in drawing boundaries on a small touch screen without more advanced tools, the use of GPS-based maps also limits the accuracy. GPS readings are generally within 1-5m accurate. This may be enough when driving along roads, but it is insufficient when trying to accurately mow lawns or stay within a user's garden. Unfortunately, the alternative methods require mapping an area, typically with a drone. These mappings can produce accuracies of 0.5-1m, considerably better but still not ideal. To produce a sufficient map, Ground Control Points (GCPs) need to be utilised. A GCP is an object which is visible in a taken aerial photo and knows its position with a high degree of accuracy. With the use of RTK, the requirement of a base station can be useful in this instance. The base station knows its position to cm level accuracy

and could be made to be visible in the aerial photos. Though the use of multiple GCP is required for higher levels of accuracy, the high accuracy of the base station itself could prove to produce comparable accuracy. With the base station in view, and the knowledge of its position, the boundaries of the given images can be determined using the camera's viewport and focal length as shown in Fig. 18. The use of these high-accuracy geo-referenced images alongside the map improvement methods discussed in Sec. 9 would result in significantly higher accuracy and coverage. Considering this, and the chunking method described in Sec. 7, this would allow for a fleet of lawnmowers to cut a substantial area with varying lengths. The user would be able to define multiple areas for each section of the course, and each lawnmower could tackle a set of chunks from each of these sections. Understandably, with the higher cost and more in-depth procedure requiring drone mapping, ideally 3D LiDAR mapping, would not be suitable for the standard user. It would be possible to provide two application views for each customer type with little change to the application itself as they would both, at their core, be simply handling a geo-referenced image with drawing capabilities. Additionally, the scheduling, telemetry and cutting height would only need to be expanded to allow the user to handle multiple lawnmowers, but this would not be a significant change either.

7. Coverage Planning

Complete Coverage Planning (CCP), also known as Coverage Planning (CP), is the process of defining a route that traverses the entire given area. The use of CCP covers aerial applications such as drones for mapping an area, or ground vehicles such as farm equipment or landmine clearance. Depending on the application of CCP the requirements may differ, a standard set may include *time-to-complete*, *total distance travelled*, and *number of rotations*, Mier et al. (2023). Energy consumption may be added to this set however, energy consumption is generally thought of as a product of these three. The number of rotations may not be as obvious as the other two, but in most applications turning is less energy efficient and more time consuming, Höffmann et al. (2022).

CCP typically takes one of two forms, cell decomposition and grid-based, Galceran and Carreras (2013), Gareau et al. (2022). Cell decomposition, as shown in Fig. 24, divides a complex shape into simpler, more manageable shapes, a path is then generated for each of these shapes. A new cell is created when the number of intersects with the scanning line increases or decreases. Grid-based methods, as shown in Fig. 22, and

demonstrated in Fig. 6, divide the area into a grid, most commonly using a bounding box with the grid sector's dimensions mapped to the dimensions of the traversing subject. Choosing which of these methods depends on the traversing subject. Cell decomposition is most often accompanied by a swaths and headlands path generation, Mier et al. (2023). Swaths and headlands are the most common method for agricultural applications, their use predates the use of modern machinery with their usage continuing due to their efficiency, simplicity, and intuitiveness. As shown in Fig. 23, the swaths and headlands approach produces parallel tracks capped by semi-circular turns. These turns must be shallow due to the often poor turning radius of farm equipment, often a longer turn is more efficient than a shorter, but sharper turn.

Grid-based methods are a relatively newer approach and are more commonly used in applications with robots and drones due to their generally better turning radius which negate a lot of the issues with sharp turns, and their use in more cluttered environments. Agricultural land will seldom contain obstacles, and those which do often contain few comparatively large obstacles. Robots will often traverse areas that contain a larger number of obstacles, and often varied sizes. This increases the complexity of the cells using cell decomposition and introduces a new problem of how to connect each cell most efficiently. Grid-based methods handle this problem from the start. Once a grid has been produced, each sector is connected so that a graph is produced. With this graph, the coverage problem can now be solved using any number of graph traversal algorithms. Common methods include Wavefront and Minimum Spanning Tree, Gareau et al. (2022), which focus on finding an efficient but suboptimal route in the fastest time; both having a time complexity of $O(n)$. These methods are commonly used when computed online due to the limited processing power of embedded devices. Other methods, often computed off-line, focus more on providing a more optimal route with less focus on computation time. It should be noted that CCP is a sub-problem of the well-established Travelling Salesman Problem (TSP), which is known to be NP-hard, meaning there is no known method to produce a fully optimal solution in polynomial time. Methods which focus on providing the most optimal route will therefore be suboptimal but can still provide a significant improvement. Two common methods are Nearest Neighbour and Christofides which have a time complexity of $O(n^2)$ and $O(n^3)$ respectively, Nilsson (2003). The trade-off between computation time and optimality is displayed clearly when comparing these two algorithms, whilst Nearest Neighbour is significantly quicker it produces a result within 25% of the lower bound

whereas Christofides will produce one within 10%. This difference may not appear too significant, but it raises the decision of how important traversal time is compared to computation time.

In the context of robotic lawn mower coverage path planning (CPP) is defining a route which will enable the lawn mower to cut all grass within a given area. In the proposed scenario this area is a perimeter with 'no-go' zones within. The perimeter will define a boundary in which the robot must not leave, and no-go zones define an area in which the robot must not enter. The most optimal route would provide complete coverage in the shortest distance, the shortest time, with the least power consumed, and no overlap of work. Though the link between these variables is clear, they are distinct. A lawn mower could complete a course within the shortest time by increasing its speed, thereby increasing power consumption. When planning a route, one must keep these in mind and not let them take precedence over the main goal, complete coverage.

A planned route can only be optimal if the lawn mower follows it accurately and so awareness of the shortcomings plays a key role when configuring a route. In this case, the lawn mower can localise itself within $\pm$20cm and can orientate itself within $\pm 10^{\circ}$. Knowing this, allowances are made in optimal variables to provide complete coverage. As the user will not be waiting for the robot to finish options can be considered to alleviate these in-accuracies namely, overlapping paths and crossing routes. For these purposes, overlapping paths are when side-by-side paths cause the lawn mower to partially go over an area that has already been traversed. Crossing routes are two or more routes that favour different directions, thus going across the same area in different directions.

The shortcomings of the lawnmower's accuracy need to be balanced with the appropriate mitigation. Simply having the lawn mower complete twice the amount of work is an easy solution. It provides twice the chance a piece of lawn will be mowed greatly increasing the chances of a successful complete coverage. However, there is no way to adjust the amount of work needed to match the given accuracies. Overlapping paths provide a variable, the amount of overlap, which can be adjusted and tested to find an optimal parameter that consistently provides complete coverage without inducing unnecessary work.

7.1. Methodology

As discussed in Sec. 7 there are two main methods when dividing a shape for coverage planning, grid-based and cell decomposition. Due to the size, turning capabilities of the lawnmower, and potentially cluttered nature of the environment, a grid-based method was chosen. Therefore, the problem of *covering a given area* can be thought of as the mower going to a set of neighbouring positions which, when traversed, ensures all possible positions are covered. As the mower uses UTM coordinates, which defines the area as a 2-dimensional plane, the possible locations can be the result of dividing this plane. An optimal solution would see the plane divided into cells of the same height and width of the mower's blades but, as the mower cannot precisely locate or orientate itself the grid is divided such that each cell overlaps slightly with its neighbour.

Producing these cells first sees a bounding box placed around the given perimeter. This bounding box is scanned from left to right, bottom to top, with the scan line moving up the y-axis every L centimetres and moving along the x-axis every $(W \times O)$ centimetres. Where L is the length of the mower, W is the width, and O is some overlapping factor. This will result in a tessellated grid of cells.

This initial stage of the process is shown in Fig. 6, where a simple shape is used as a demonstration.

To begin producing a path for the grid, the centroid of each cell is taken. A weighted, undirected graph is then constructed with the centroids as the nodes. Each node has a connection to any neighbour directly up, down, left, or right. The weight of these edges is the distance between them using the 2-dimensional UTM coordinates. Initially restricting the mower to four directions is due to the use of tank tracks and the presumption that this would result in slower turning. As mentioned, sharper turns are often less optimal even if they result in shorter distances due to the time taken to perform such turns.

Once the graph is constructed the problem can now be thought of as a graph traversal. The goal of the mower is to visit each cell, or graph node, in the shortest distance, returning to the start, and optimally, visiting each node once. This scenario is analogous to the commonly known Travelling Salesman Problem (TSP) in which, a travelling salesman needs to visit a set of cities, travelling to each once, and do so in the shortest distance.

For small areas such as the one shown in Fig. 7, this provides a sufficiently optimal route in an acceptable amount of time. However, as shown, the route has two issues

namely, gaps within the covered area, and overflow around the perimeter. The issue of gaps can be solved by increasing the amount of overlap for each path, for the example in Fig. 8 the overlap was set to 50%. Doing so greatly increases the number of cells and therefore nodes in the graph, which will become a point of discussion in later sections, but for now one can assume some level of overlap is necessary for this process.

Perimeter overflow is not solved by path overlap and becomes more severe with more complex shapes. To prevent overlap an inner perimeter is produced and traversed first before traversing the rest of the route. This way, any overflow will be into the path of this inner perimeter and not the user- defined perimeter.

The same method of producing an inner perimeter to reduce overflow can be applied to no-go zones. As any form of localisation will not be completely accurate it's desirable to account for this. Producing an outer no-go zone creates a buffer to reduce the chances of going over any no-go zones. The issue with this method is creating an overly pessimistic buffer zone will reduce the overall coverage whereas one which is too optimistic will have a higher chance of entering a zone the user does not want it to.

Due to the area's size, these examples contain a limited number of cells and thus do not yet expose the issues with this method in its current described state. Whilst TSP is a suitable method for coverage planning, it's unsuitable for large areas due to its time complexity. A naive, but optimal, approach has a time complexity of $O(n!)$ whereby every possible solution is checked. This can be improved by using an approximation as discussed in Sec. 7. Due to the computation being performed offline, and the greater importance placed on traversal time and energy efficiency, it was decided to use the Christofides algorithm. Additionally, the user would not notice the slower computation time as it would be performed in the background however, the user experience would be improved by providing a shorter, for efficient route.

When considering large areas such as golf courses, the graph can be composed of over 100,000 nodes. To perform even an approximation would take an unreasonable amount of time and would require a considerable amount of memory. To alleviate some of this work the proposed method sees the larger area chunked into larger cells in which each will contain the initial mower-sized cells. Each of these chunks will produce a TSP route. Thus, reducing the time complexity of the Christofides approximation for example to $O(4^{\frac{n}{m}} \times m)$ where m is the number of macro chunks. These macro chunks are adjustable, containing as few points as needed. Once all chunks are complete, a final TSP is performed to connect all chunks. In the example shown in Fig. 10, over

200,000 original points are converted to less than 1,500 chunks, with no more than 450 points in each. As these speed-ups are pre-TSP it becomes difficult to compare running times of commonly used methods however, it is common for optimal solutions for a TSP containing tens of thousands of points to require hundreds, if not thousands, of CPU years. With this, the proposed method represents a significant improvement.

When considering the example, one can see that the majority of the area is composed of identically sized cells. And with these macro cells and the original microcells aligned to the same bounding box, it is fair to say that the contents of each macro cell are the same. Knowing this, if the maximum amount of microcells is known one can assume that if a graph contains said amount, it is a whole macro cell and therefore, the same as any previous. Using this, one needs only perform the TSP algorithm for one and can then copy the result for every other, leaving only the partial cells. With this change to the method completes in under 15 minutes running on an Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz with 16GB of RAM. This method is limited by the potentially different vertical topography of each chunk is unknown. The lawnmower would still be able to traverse the route and arrive at each checkpoint due to it continuously checking if the point has been reached, and if not continuing to adjust course however, this would likely not be the most efficient path. Accounting for elevation requires the use of a Digital Elevation Model (DEM). A DEM provides information about the elevation of an area, most map APIs will provide access to such information. The drawback of these provided DEMs is the lack of information on smaller objects; data about trees, houses, or landscaped terrain will generally not be available. Achieving this higher level of detail would require a scan of the chosen area to provide a Digital Surface Model (DTM). The comparison of these model types is shown in Fig. 19. These are often completed using a drone or UAV fitted with LiDAR. The benefit of marginally better performance over uneven terrain, which is uncommon on golf courses, was decided to not be a priority within the scope of this project. The lawnmower's ability to work outside business hours gives the user the time to complete the work, which also means that we do not require use having the highest level of optimisation at this current state of the project.

Due to the process of generating a path from a grid sequential 'checkpoints' are often traversed in a straight line, and as the implemented path traversal method uses a point-to-arc system the mower need not store each point as it primarily considers the line between said checkpoints. This allows the removal of intermediate thus reducing the memory requirements of the path. Removing checkpoints also reduces the chances of

some being inaccessible due to obstruction, and due to the increased time consumed by the method which determines this, described here 9.2, any reduction in the need for this method is desirable. Removing all such intermediate points produces a subset that is confined almost entirely to the perimeter as shown in Fig. 11

8. Path Traversal

Once a path has been generated the lawnmower must now traverse it. As the path is a list of coordinates the lawnmower must determine the distance and angle between its current position and the checkpoint in the path. As the coordinate system being used (UTM) assumes the area to be a 2D plane this makes calculations much simpler. The distance is determined using the lawnmower's current position as provided by the RTK, and the bearing using the IMU. Though both of these sensors are accurate, they are not perfect. Using an IMU to determine the lawnmower's bearing can produce a reading within $\pm 10°$. This inaccuracy is acceptable for small local movements, but over time this inaccuracy will produce a continuously worsening drift. The RTK therefore, is used to correct this drift and inform the lawnmower to adjust its course.

As discussed numerous methods can be used to determine whether a subject is off-course using location data such as GPS, or in our case RTK. Most of these methods focus on cars. A more complex scenario in which the subject has agency, multiple route options, and the localisation must conform to the subject. In our case, the lawnmower only has one route, it cannot choose a different route, and if off-course the lawnmower must conform to the intended route. This scenario allows for much simpler, and therefore, less computationally demanding algorithms.

8.1. Methodology

The method implemented is a relatively basic one, demonstrated first in Fig. 25. In which, A and B are points in the path, and a line L is drawn between them - L being the ideal route for the robot to take. As it is known the lawnmower has inaccuracies in localisation due to the IMU and RTK, two more lines B_0 and B_1, are created which run parallel to L. These new lines represent a buffer that is acceptable for the lawnmower to be within. Without this buffer, the lawnmower would adjust its course every time it does not follow the perfect route, and with the known inaccuracies, this would be all the

time, resulting in little progress. This results in the lawnmower being off-course only if it is outside the defined buffer.

In addition to known accuracies, one must also consider erroneous readings. These are different from inaccuracies as they're non-continuous and outside the normal inaccuracy range. To address these points, one must consider potential scenarios which may occur. In Fig. 25 three scenarios are shown. In the first, one point is outside the defined buffer, in this case, one would likely assume it to be erroneous as all other points stay within the buffer. If the lawnmower corrected its course at this point by turning, then all subsequent points would be outside the buffer. Thus, the lawnmower must not adjust course for a single reading. In the second case, multiple points are outside the buffer, each point increasing the likelihood that the lawnmower is off-course. The problem is at which point should the lawnmower adjust its course, and how likely is four erroneous points in a row. The third case represents the biggest challenge, in this case, multiple points are inside and outside the buffer, some being on opposite sides of the buffer. Cases one and two can be handled by waiting for enough points to confirm if the lawnmower is off-course for example, if 10% of the last 100 points are off-course one might determine the lawnmower to be off-course. However, in the third case, this would appear to be insufficient. Enough points may be outside the buffer to confirm the lawnmower is off-course however, these points are on opposite sides of the buffer, so which way should the lawnmower turn to adjust its course. More importantly, the erratic movement shown in the sharp turning angles and large distances are in theory impossible given the lawnmower's properties and programmed movement commands. Knowing this one can remove these erroneous points by determining if moving this distance, at this angle during this time is impossible. If the movement is impossible, then the reading is removed. Removing such erroneous readings will affect the accuracy of the path traversal, but with the route overlap, performing multiple routes, and the frequency of RTK readings this effect is considered negligible.

9. Obstacle Avoidance

Traversing a path without prior knowledge of the environment presents the issue of unknown obstacles. Whilst the CCP algorithm will handle any *no-go* zones defined by the user, it is impossible for the off-line path planner to handle unknown obstacles. Without handling these unknown obstacles, the lawnmower will get stuck behind, crash

into, or in the worst case; drive over them. The last scenario is especially important to handle when considering the hazard of the lawnmower's blades. Autonomous devices, such as floor cleaners (Robot vacuums), only require obstacle avoidance to continue traversing a route without getting stuck. In these scenarios, the device can use a bump detector, which detects when the robot has collided with an object or short-distance sensors such as infrared or millimetre wave. These methods are acceptable for use in these scenarios as the robot has little to no potential of harming the object or person they may collide with as they are typically small, slow-moving, and with no harsh edges. An autonomous lawnmower requires the ability to detect objects at a further distance to help in removing the chance of colliding entirely.

To avoid obstacles the lawnmower must first be able to detect obstacles, more directly, the shape, size, and distance of the obstacle must be known. The most commonly used sensors are cameras and LiDAR. Whilst cameras can provide significant detail on the shape of the object, distance and size can be a challenge. Without a reference point, a single camera cannot determine this information on its own. Using two cameras can provide this information, much like how most animals and humans perceive depth and size with two eyes. The use of two cameras begins to present issues with performance, in some cases requiring desktop hardware for real-time avoidance, Phan et al. (2022). Light Detection and Ranging, more commonly known as LiDAR, uses the reflections of a laser to detect objects. The laser is able to provide distance values at specific intervals, commonly at set angles thus allowing the user to determine the distance and direction of a detected obstacle. Though 3D-LiDAR, which uses multiple arrays of lasers to produce high-resolution imaging exists, obstacle avoidance for robots more commonly uses 2D-LiDAR, Madhavan and Adharsh (2019), Peng et al. (2015) Ghorpade et al. (2017). Using 2D-LiDAR allows for much more efficient algorithms, reducing the amount of memory required and reducing the time handling readings. Knowing an obstacle's distance and direction is enough for avoiding it, using 3D-LiDAR or higher resolution LiDAR usually presents modest improvement as it can only be used to determine the height and fine details respectively.

With the sensor chosen, an appropriate avoidance method must be decided. Camera-based methods typically use some form of machine learning, or more specifically, Q-Learning, Kim and Lee (2022), Wenzel et al. (2021), Tai and Liu (2016). Q-Learning is a form of reinforcement learning whereby the model gets rewarded for performing correct actions and penalised for performing incorrect actions, in this case, they would

be reaching the end goal and colliding with obstacles. The main benefit of using Q-Learning is the ability to train a model specific to the robot and its environment. Unfortunately, this can also be its biggest drawback. Training a model takes a significant amount of time, and requires real-world values to train with, and any omitted data will require re-training the model. This omitted data could be a situation not thought of before training or one which did not arise during training. In the case of Tai and Liu (2016), the robot had issues driving in a straight line towards its goal because it did not encounter driving in a straight line much during training. In addition to requiring a

LiDAR is often used when efficiency is required over detail thus methods utilising LiDAR are often more classical, relying less on computationally heavy machine learning models and more on well-defined logic. The methods described in Ghorpade et al. (2017) and Peng et al. (2015) detail similar models for avoiding obstacles:

1. Face the direction of the target location
    - If no detected obstacles, move towards the target
2. Collect all detected LiDAR points in the desired range
3. If adjacent points are close enough such that the robot cannot pass by them collect them into a single group
    - If the points are not close enough, start a new grouping
4. Once all groupings are found, take the outer-most points to define the bounding box of each group
5. Turn to the left or right
    - The direction may alternate or depend on the location of the obstacles
6. Repeat step 5 until no more obstacles are detected

This method is simple but cost-efficient. Taking only the outer-most points for obstacles and merging close obstacles further improves the efficiency. This method may not be able to identify obstacles, but when the only goal is to avoid obstacles quick decision-making is preferred over detail. The main limitation of such a method is the inability to handle dynamic obstacles. With this method, if an obstacle is moving in the same direction as the robot, it is likely the robot will begin following the obstacle, unable to pass

it. To avoid moving obstacles the model must determine the obstacle's speed and trajectory, Wei et al. (2022). Whilst Q-Learning can handle this with enough training, the simple LiDAR method would require the addition of tracking obstacles between scans. It would also result in groupings being determined not just by proximity, but by moving in the same direction at the same speed. This becomes more complex when factoring in the robot's speed and trajectory, and the potential for obstacles to obscure others.

9.1. Methodology

Whilst the off-line coverage path planner handles any known obstacles defined by the user, there is no way for it to handle unknown obstacles. Therefore, some form of obstacle avoidance is needed to prevent colliding with, or driving over, obstacles. LiDAR was chosen as the obstacle detection sensor due to its low computational requirements. The use of cameras for obstacle avoidance has often been accompanied by desktop hardware with dedicated GPUs and as such with the given hardware it would prove too computationally demanding to provide real-time detection in an acceptable time limit. Gareau et al. (2022) gives a reaction time requirement of less than a quarter of a second based on a typical walking speed of 1.4m/s and a proximity of 30cm. Though this work focused primarily on busy dynamic environments, the robot poses less harm than a lawnmower. As such the lawnmower should aim to have a comparable, if not faster, reaction time. To this end, the obstacle avoidance method has focused more on computational speed to allow for adequate reaction times rather than accuracy. The method is based on those described in Madhavan and Adharsh (2019), Peng et al. (2015), and Ghorpade et al. (2017). The sensor used is the LDS-01, this sensor provides a detection range of 120mm - 3,500mm with a full 360° field of view, with one point collected per degree of rotation. As the mower only moves forwards and will only do so if no obstacles are detected, only points 30° either side of the centre line is considered. This range proved successful in simulated tests, which are detailed more in Sec. 10, with the lawnmower staying close enough to obstacles allowing for close cutting but would prevent the lawnmower from attempting to fit through gaps too small.

The method, described in Sec. 9, was found to get stuck on obstacles that were perpendicular to the path being traversed. This was due to the combination of facing the target each iteration and turning opposite to the side of the obstacle. Each movement would result in the obstacle changing sides and thus creating a loop in which the lawn-

mower would get stuck. To prevent this, the lawnmower no longer turns to face the obstacle directly and iteratively turns until its view is free of obstacles. The lawnmower now gradually turns towards the target, stopping once obstacles are in view. Not only does this prevent the issue of becoming stuck, but it reduces unnecessary movement as the lawnmower's turning now follows the shape of the obstacle. This process is applied to real obstacles using the sensor and virtual boundaries using a virtual LiDAR array which mimics a real sensor by using virtual lines extending from its position. If these virtual lines intersect the given boundaries, then the intersection is returned just as a detected LiDAR reading.

9.2. In-accessible Checkpoints

A significant challenge considering path traversal with unknown obstacles is determining if a point is in-accessible. At what point does the lawnmower decide it cannot reach its intended target. In this case, a point is accessible if it is completely surrounded by an obstacle. This may mean the obstacle is on top of the target position or multiple obstacles surround the target. A simple method may determine this by comparing the assumed time to the point given distance and speed however, this does not account for wheel slippage, localisation in-accuracies, or obstacles that obstruct a direct path but don't completely surround the target. Even when taking these into account this problem remains difficult when considering the lawnmower may not take the most optimal path. To avoid obstacles the lawnmower may at first go in the opposite direction but will eventually arrive at the target.

To solve this problem, it has first been split into two namely, when to consider a point *may* be in-accessible and when to determine a point *is* in-accessible. The first problem was addressed by first observing what the simulations show when a point is in-accessible. In almost all cases the lawnmower was shown moving back and forth over the same location. This is in line with the method as the lawnmower will try, where possible, to go directly to the target, but when an obstacle is in the way it will be pushed away; only to retry the same step. This behaviour is defined as a rule such that if the lawnmower's position is close to its position after a given number of attempts. This allows the lawnmower to travel in directions away from the point in an attempt to find a valid path, whilst detecting when it is stuck. Once the lawnmower has determined the point *may* be in-accessible, it now must determine if it *is* in-accessible. To do this the

lawnmower collects and stores points collected by the LiDAR. Using the lawnmower's position along with the angle and distance of the reading, these detected points can be added to the existing virtual boundaries to build a virtual map of the unknown obstacles. Using its current knowledge of the environment the lawnmower produces a grid and performs an on-line path finding algorithm. The result of this algorithm results in one of three scenarios. If the algorithm returns no path, then the target is completely surrounded by obstacles, the target is determined in-accessible and skipped. If the algorithm returns a path, it does not necessarily mean the point is accessible, it may mean the lawnmower is unaware of all obstacles. In this case, the path will take the lawnmower to these undiscovered obstacles, the lawnmower will get stuck again, and the process will be repeated until a valid path is returned and the target reached, or all obstacles are mapped, and the point is determined as in-accessible. This process can be seen in Fig. 13 where the lawnmower gradually moves along the obstacle, improving its knowledge as it does so.

Using this method however quickly uses a considerable amount of memory to store this information about the environment. This can be an issue for embedded devices that have limited memory capacity. Due to the use of a grid for the online path-finding algorithm, the coordinates need to be scaled to account for the robot's width otherwise the algorithm would assume the lawnmower can fit through impossibly small gaps. As the collected points are only used for this grid, this scaling can be used to reduce the memory used when mapping the area. If each point on the grid is 10cm apart then any points less than 10cm apart would become merged into one when creating the grid. Therefore, it is only necessary to store those 10cm apart to recreate the required grid. To reduce these the collected points an initial point is chosen. The distance from the chosen point and all remaining points is then compared. If the distance is less than the acceptable width for the lawnmower to pass through, then it is added to the group. Once no more points are found to be within the group, the process is repeated for all remaining points until each is part of a single group. Each group will represent the outline of an obstacle or part of an obstacle. Reducing the points for each group simply requires taking the endpoints of the outline and inserting a point at an interval equal to the grid's resolution - in this case 10cm. In the simulation result shown in Fig. 14, the lawnmower executed the same number of movement commands. The number of points was reduced from 6897 to 329, a reduction of 95%, and although this method uses a selection sort method which is $O(n^2)$ time complexity, the test completed in 45

minutes with this method compared to 40 minutes without. Though this difference is assumed to become increasingly great as more points are collected. The main drawback of this method can be seen clearly in the difference in progress between Fig. 13 and Fig. 14. Without using this method, the lawnmower traversed more area and subsequently mapped more of the obstacle, this is because the method reduced the appears to forget some of the areas it has mapped and retries the same path multiple times.

9.3. Limitations

There exist two major limitations with the current implementation of obstacle avoidance in this project namely the inability to directly handle dynamic objects and to adjust the course when new obstacles are found. With the current implementation, the lawnmower is unable to plan a route that effectively avoids moving obstacles however, this does not mean it cannot avoid them at all. The lawnmower currently moves in steps, scanning the area each time. This means, although the object is assumed static, it is updated regularly. Only dynamic obstacles which move in the path of the lawnmower in between scans would be at risk of collision. This would ideally be rectified by building on the method to account for the direction and velocity of each obstacle though this would require constant scanning and computation, and with most obstacles in lawn-cutting scenarios being static this would likely be redundant the majority of the time. A more energy-efficient method may see an additional sensor or less intensive, but constant, LiDAR scanning whereby the system is told to halt when an object gets too close to the lawnmower. Once a nearby object is detected then the lawnmower can either wait for it to exit the area if it is determined to be dynamic or perform a full scan and re-compute a new path. This would act more like an emergency stop rather than a full avoidance algorithm. The latter limitation, adjusting the course given new information, was intended to be handled with the off-line planner. This method would see the detected points being sent back to the server where they would be processed. Each point would need to be classified as dynamic or static as only static obstacles should be considered in the global planner. This step could be solved online should the lawnmower implement the aforementioned method of directly handling dynamic obstacles. Once determined to be static, the route can be re-computed. Initial plans saw this method happen after multiple runs of an area where data could be collected and averaged to produce a more accurate map. Though this method would still be valuable to implement, it does not improve

the current traversal. Therefore, an additional method would see updates to parts of the route as it is being traversed. When the lawnmower detects an obstacle and successfully traverses it for the first time. This new path taken should augment the intended route, adjusting it to match the successful path. The method described in Freitas et al. (2009), sees the use of multiple traversals from numerous vehicles to determine the location of roads unknown to the current digital map. Simply, if a large number of similar routes are recorded by vehicles that do not correlate to any known routes, then it would be assumed to be a new route and the map updated with this new route. The idea can be applied to the lawnmower, but instead of multiple vehicles, the same vehicle would perform multiple routes. This method can also be employed to improve the given digital map's perimeter. As the initial map is produced using a GPS-based API the accuracy of the map is not ideal for accurate tasks such as mowing. The combination of RTK and LiDAR, which are both accurate to the cm level, can be used to improve this.

10. Navigation

The combination of coverage planning, path traversal and obstacle avoidance creates the navigation system for the lawnmower. As these methods were developed alongside the development of the hardware platform and localisation methods it was not possible to perform real-world testing for the majority of the project. However, it was important to perform some form of testing to determine the validity of methods and discover any issues that may be present. To this end, a 'simulation', developed in Python was created. Though this program will be referred to as a simulation it is important to note that it does not take into account some important values such as the lawnmower's dimensions and weight, the in-accuracies caused by the IMU and RTK sensors, or slippage caused by the lawnmower's tracks. Though some in-accuracies were partially addressed by adding some noise this is not thought to be sufficient enough to yield any valuable insights. Implementing methods that would account for these real-world properties were outside the scope of this project and would have left little time for further development of more important methods, especially as it was intended to perform real-world testing on the platform which would have yielded far more important results. Despite its limited capabilities it proved useful to visualise any issues the lawnmower may have in given scenarios for example, the lawnmower would often become stuck on obstacles that lay perpendicular to the desired path, as discussed in 9. With this visualisation it was clear

to see the steps which were being taken and provided insights into how the problem occurred, and how it might be fixed. Though this problem may be constrained to this simulated environment and may not occur in the real world, the development of a fix would allow both methods to be tested once the platform was completed. An example of the simulation can be found in 21, in which the lawnmower is following two crossing paths within an environment with two unknown obstacles. Each frame is the lawnmower moving in the chosen direction, no frames are exported for turning as the final decision was considered the most important to visualise. The lawnmower can avoid obstacles and return to the desired path when it is off-course. Though it is clear to see that returning to the desired path happens later than would be ideal. This is in part due to the lack of multi-processing in the simulation. A real-world platform, would perform off-course calculations and obstacle scanning at the same time as movement, but in this simulation, these are done in steps. Each step determines if it is off-course, if obstacles are in the way, and then performs the appropriate move. In the real-world application, the platform might see one process scanning and detecting obstacles, another recording RTK readings and determining if it is off-course, and another controlling the movement based on these processes' output. The result of this simple simulation can be seen in Fig. 20 where to dimensions of the lawnmower's blade are accounted more and the area cut is highlighted. The figure also demonstrates the benefit of using two crossing paths as mentioned in 7. It is clear to see that when a obstacle is avoided the optimal route is not followed, but by performing a second route in a different direction this improves the chances this area will not be affected by this movement. Again, this is only a simple simulation as these results would of course differ in the real world. As a result, the implementation is unable to provide smooth, continuous movements and requires stopping at intervals to perform computations. In the flow diagram shown in Fig. 15, with the 'BTrack' and 'ATrack' sections are shown in Fig. 16 and Fig. 17 respectively, it is clear to see that the four main steps 'OffCourse', 'NoProgress', 'Move', and 'Turn', could be handled by separate processes. Though 'Turn' would be re-configured to encompass 'ObstacleAvoidance' which in this case would output turn commands but in the case of handling dynamic obstacles may output a new local path to take.

11. Testing

11.1. RTK Testing

To test the RTKLIB and ROS RTK setup, we took the RTK system on a walk around Chapelfield park in central Norwich. The setup is shown in Fig. 29. Data was collected in ROS bags for post-processing and logging and plotted in Fig. 28, and cross-checked with OpenStreetMap data. The conclusions from this test are as follows:

- The objective of this test is to make sure that the overall *shape* is accurate, not the accuracy of individual points or the global accuracy.
- The overall shape appears okay. The 'ground-truth' should not be considered too high, since we do not know how accurate the OSM data is, it is open-source data too, but nonetheless, we will have to consider fixing global inaccuracy. This could involve doing multiple runs on the area and training a machine-learning model to fix the global inaccuracy. Unfortunately, we did not have enough time to do this additional data collection.

At this stage, we were disappointed with the RTK accuracy. So we tested RTK accuracies in various locations across Norwich. By keeping the antenna in a static location, and measuring the standard deviation in points received, we can create a metric of inaccuracy. This is shown in Fig. 31. The geometric median was calculated with an implementation of Vardi and Zhang (2000).

We are very disappointed with the accuracy available. The only test in which RTK accuracies were close to ideal were in Fig. 31d. We believe that the straight line in which erroneous points appear as a result of **multipath interference** (see Fig. 30), an error which should be corrected with RTK.

We believe that the poor RTK accuracy is a result of factors outside of our control. Namely:

- The RTK system with which we were provided, the u-blox NEO-M8P was a first-generation product in which RTK became available 'to the masses' but by now is very out of date. It can only receive signals from a single band, which reduces the accuracy is capable of, compared to, for example, the ZED-F9P.
- The RTK base station we used was outside of our control- it was accessed through a raw TCP stream in a proprietary format. The 'standard' way to do this would

be to use a system called NTRIP with a format such as RTCM. Other formats are capable of sending the exact location of the base station in them, which improves accuracy.

- A better-suited metal plate to act as our grounding plate for the antenna could improve accuracy. u-blox (2016) describes a setup with a manufacturer-provided plate for the antenna which would reduce multipath errors. They also used multiple base stations, which we didn't have access to either.

11.2. Navigation Testing

The navigation methods require orientation and location as input from the IMU and RTK respectively. Whilst the IMU provided an imperfect heading, this was expected and is the reason the methods described in Sec. 8 were developed and the use of RTK was so important. Unfortunately, as described above, the RTK was unable to provide cm-level accuracy. Even without considering the methods were developed with the assumption this could be achieved with continuous readings whilst moving, the task of mowing a lawn cannot be done effectively with such accuracies. The lawnmower's movement is based heavily on its current position, bearing, and target. When one of these values changes frequently, and significantly, the lawnmower moves as if it is lost; no progress is made, and it constantly moves in the wrong direction. With such a significant issue the simple task of staying within a 2x2 meter area and traversing a straight 1m line was not possible. The lawnmower was able to detect physical and virtual boundaries, but without accurate location data, it was impossible to tell if it was acting on this knowledge correctly, other than us looking at some console outputs.

12. Evaluation

The RTK chipset that we were using for the project is relatively old in terms of more affordable, consumer-level hardware and unfortunately lead to unexpected inaccuracies that limited the amount of testing that we available to do outside of simulations for testing real-world performance because of the problems mentioned in Sec. 11.

An observation made during the navigation tests that the robot was struggling again with stationary rotations, however unlike the previous instance of this issue the problem

was with the robot chassis design. The tracks of the robot were getting stuck and causing the robot's chassis to twist. From this we determined that robot was making too much contact with the ground and would therefore have to overcome a much higher level of friction than expected which unfortunately would periodically get stuck, especially when rotating on uneven surfaces. Therefore, a quick solution was to alter the shape and tension that the robot's tracks where previous set at (see Fig. 32a) to a configuration that allowed us to rotate on the spot (see Fig. 32b) so that we could back to testing as quickly as possible. Because of these types of issues occurring that are related to the robot's chassis (and method of locomotion) we should consider whether we should try alternative chassis and/or methods of control for the robot (More in this in Sec. 13.1).

13. Conclusion

In conclusion, we aimed to develop a robot lawnmower that could use RTK-GNSS to get precise position and be capable to perform accurate path navigation of a path plan that was the result of a novel complete coverage planning algorithm from a defined operating area. We did create a minimal application that we can use to define our operating area and use this data for path planning. Additionally, we have simulated the use of LiDAR for reacting to undefined obstacles in the working areas and explored the process involved with navigating the robot around them to reach its goal.

We have unfortunately been unable to achieve the centimetre level accuracy that RTK-GNSS can provide and therefore left us unable to perform more extensive tests for the robot's navigation and obstacle avoidance features. This was due to limitations faced this the hardware we had access to and with not having enough time or access to better equipment we were unable to complete the testing for the project.

Along with being unable to test the performance of the navigation methods, it was not possible to test the accuracy of the GPS-based maps. Whilst it can be assumed the use of aerial images with GCPs as discussed in Sec. 6 would provide significant improvement, it is unknown whether this would be necessary for general consumer use, or only commercial use. With the main focus of this project being operating within virtual boundaries, it was unfortunate that due to the limitations of the RTK this was not explored or tested fully.

13.1. Future work

The possibilities for expanding the work of this paper can seem limitless, however, we can separate the categories of this work as being focused on either the 'Hardware' of the robot or the 'Software' written for the system, whether that is the robot's firmware, the navigation and planning algorithms, or the mobile application. By taking the issues and limitations that we have faced when working on this project we can suggest some starting points for future additions and/or changes.

13.1.1. Hardware

Some areas for change and/or improvements with the robot platform's hardware are:

- having a better RTK chipset that is capable of providing the expected centimetre level of precision for the robot localisation and would therefore allow us to apply and test the coverage planner and robot path traversal algorithm without the current location values jumping around too much.
- Switching the chassis used for the robot would help tackle a range of issues and open the potential to try new hardware configurations and techniques, such as:
  - The robot had trouble turning on the spot, as mentioned in Sec. 3.1.1, due to the motors and what we observed with the issue in Sec. 12 regarding the tracks we so should consider either modifying the robot's chassis to solve this , or look at using a different chassis that does not use tracks, and even differential drive for movement, on our test platform. Therefore we could look at using a different chassis that doesn't use tracks, and even differential drive for movement, on our test platform.
  - An alternative chassis could allow us to explore different forms of locomotion, such as wheel-based robots instead of the continuous tracks that we see in our robot.
  - With a Wheel-based robot chassis it would be able to integrate wheel encoders on the robot to be able to record odometry data that could be used in later works, as this is currently hard to do with our current chassis.
- If we switched to a wheel-based robot chassis design we could look at using 'Omni-directional' wheels as an alternative form of movement and could allow

for the exploration of different navigation algorithms that could exploit the advantages associated with a 'Holonomic' (can freely move in any direction and the controllable DOF equals the total DOF) drive system.

- Scaling the robot up to be larger would allow us to mount a larger antenna for the RTK-GNSS, which would allow us to get accuracy, and/or mount a larger ground plan underneath the antenna.
- The addition of a camera on the robot would enable us to explore using AI for object detection and avoidance. The camera's data could also be processed and used to provide us with another source of odometry (visual odometry) that we could fuse with the rest of the data to provide better navigation, localisation, etc...
- Improving the local processing power on the robot would also allow us to be able to perform more intensive calculations without having to rely on cloud computing (and therefore reliant on an internet connection). This would be especially important if we were to do any AI and/or image processing on the robot locally, as the Raspberry Pi would be too slow to do any real-time detection, so swapping the SBC out with a more powerful option, like one of the NVIDIA Jetson series, we would have the resources to develop and test these ideas.
- Refactoring the system to have a single IMU would help centralise some of the software controls and systems, but by also upgrading IMU to a more precise sensor, perhaps with a factory calibration to guarantee a more reliable baseline performance, and mounted on the robot in a place far from other components/systems on the robot to minimise interference would allow us to get a more accurate reference to north which would allow use to get a more accurate representation of the robot forward direction and orientation.

13.1.2. Software

When looking at what could be done for the robot's software, we have both the upper "control" layer where all the code running on the ROS network (and the Raspberry Pi) and the more "lower" level code that runs on the ESP32 that interfaces with the motors and other sensors to provide access for the SBC.

The code for the ESP32 could be refactored to eliminate the need for the ICM-20948 IMU from the system and use a single IMU that can be directly interfaced with the

SBC for easier data access in the ROS network. Additionally, we could further refactor the ESP32 code to use micro-ROS[5] to allow the microcontroller direct access the ROS networks by creating and running node that can both publish and subscribe to other ROS topics in the network. This way we could not need to route all the data going to and from the microcontroller through a dedicated ROS node like we had to do for this implementation of the system and instead would have more freedom to handle data across the devices.

As discussed in Sec. 9, the obstacle avoidance methods would need to be reworked to handle dynamic obstacles, have an emergency stop, and need the methods of detection, movement, and traversal to be split into their own processes. These changes may not significantly improve avoidance or coverage of the given area, but they would greatly improve safety. A lawnmower is unlike other robots such as automatic hoovers, they have the potential to seriously injure or damage anything, or anyone, it comes into contact with and therefore if this system were to be developed fully there would need to be more focus put on safety.

Improvements relating to map accuracy such as those discussed in Sec. 6 relating to the use of drone mapping and GCPs would perhaps be the most important if this platform was to be used for industry purposes. The accuracy of GPS-based maps is not sufficient, and although the cost of aerial mapping is high, it would likely be a one-off cost which companies would likely conclude is acceptable given the ability to perform such tasks at any time, especially outside business hours. A substantial improvement to the efficiency of the platform would be utilising multiple lawnmowers, with each being assigned its own section. This idea was explored briefly when developing the chunking method described in Sec. 7 however, it was determined to be outside the scope of this project given we were unable to have a single complete platform, and even then, would be unlikely to develop multiple.

References

Advamation (2013). Advamation - know-how - raspberry pi i2c bug. Accessed: 19/04/2023.

[5] micro-ROS Website: *https://micro.ros.org/*

Åström, K. J. and Murray, R. M. (2021). *Chapter 11 - PID Control*, pages 11–26. Princeton university press.

Blake, K., Huey, R., Menezes, D., Root, J., Tran, L., and Chen, H. (2022). Robot navigation using ultra-wideband indoor localization and dead reckoning algorithms. In *2022 Opportunity Research Scholars Symposium (ORSS)*, pages 12–15.

Cai, G.-S., Lin, H.-Y., and Kao, S.-F. (2019). Mobile robot localization using GPS, IMU and Visual Odometry. In *2019 International Automatic Control Conference (CACS)*, pages 1–6. IEEE.

Cai, H., Hu, Z., Huang, G., Zhu, D., and Su, X. (2018). Integration of GPS, monocular vision, and high definition (HD) map for accurate vehicle localization. *Sensors*, 18(10):3270.

Carr, C. S. (1969). Network subsystem for time sharing hosts. RFC 15.

Castanedo, F. (2013). A review of data fusion techniques. *The scientific world journal*, 2013.

CEVA (2023). *BNO08X Data Sheet*.

Chen, W. (2014). Geo-positioning, gps, dgps, and positioning accuracy.

ClemensElflein (2023). ClemensElflein/OpenMower: Let's upgrade cheap off-the-shelf robotic mowers to modern, smart RTK GPS based lawn mowing robots! Accessed: 02/10/2022.

Ferreira, A., Matias, B., Almeida, J., and Silva, E. (2020). Real-time GNSS precise positioning: RTKLIB for ROS. *International Journal of Advanced Robotic Systems*, 17(3).

Freitas, T. R., Coelho, A., and Rossetti, R. J. (2009). Improving digital maps through gps data processing. In *2009 12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–6. IEEE.

Galceran, E. and Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, 61(12):1258–1276.

Gareau, J. C., Beaudry, É., and Makarenkov, V. (2022). Fast and optimal branch-and-bound planner for the grid-based coverage path planning problem based on an admissible heuristic function. *Frontiers in Robotics and AI*, 9.

Ghorpade, D., Thakare, A. D., and Doiphode, S. (2017). Obstacle detection and avoidance algorithm for autonomous mobile robot using 2d lidar. In *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, pages 1–6. IEEE.

Hamilton, J. D. (2020). *Time series analysis*. Princeton university press.

Höffmann, M., Clemens, J., Stronzek-Pfeifer, D., Simonelli, R., Serov, A., Schettino, S., Runge, M., Schill, K., and Büskens, C. (2022). Coverage path planning and precise localization for autonomous lawn mowers. In *2022 Sixth IEEE International Conference on Robotic Computing (IRC)*, pages 238–242.

Jung, P.-G., Lim, G., and Kong, K. (2013). A mobile motion capture system based on inertial sensors and smart shoes. In *2013 IEEE International Conference on Robotics and Automation*, pages 692–697.

Jyothsna, K. S. and Raju, K. P. (2021). Irnss based estimation and mitigation of multipath error for strategic terrestrial applications. *Journal of Electrical Engineering and Technology (IJEET)*, 12(7):120–128.

Kim, H. and Lee, W. (2022). Dynamic obstacle avoidance of mobile robots using real-time q-learning. In *2022 International Conference on Electronics, Information, and Communication (ICEIC)*, pages 1–2. IEEE.

Liu, Q., Liang, P., Xia, J., Wang, T., Song, M., Xu, X., Zhang, J., Fan, Y., and Liu, L. (2021). A highly accurate positioning solution for c-v2x systems. *Sensors*, 21(4):1175.

Ludwig, S. A. (2018). Optimization of control parameter for filter algorithms for attitude and heading reference systems. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8.

Ludwig, S. A. and Burnham, K. D. (2018). Comparison of euler estimate using extended kalman filter, madgwick and mahony on quadcopter flight data. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1236–1241.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074.

Madgwick, S. et al. (2010). An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 25:113–118.

Madhavan, T. and Adharsh, M. (2019). Obstacle detection and obstacle avoidance algorithm based on 2-d rplidar. In *2019 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–4. IEEE.

Mahony, R., Hamel, T., and Pflimlin, J.-M. (2008). Nonlinear complementary filters on the special orthogonal group. *IEEE Transactions on Automatic Control*, 53(5):1203–1218.

Mannings, R. (2008). *Ubiquitous positioning*. Artech House.

Mier, G., Valente, J., and de Bruin, S. (2023). Fields2cover: An open-source coverage path planning library for unmanned agricultural vehicles. *IEEE Robotics and Automation Letters*, 8(4):2166–2172.

Moore, T. and Stouch, D. (2016). A generalized extended kalman filter implementation for the robot operating system. In *Intelligent Autonomous Systems 13: Proceedings of the 13th International Conference IAS-13*, pages 335–348. Springer.

Nilsson, C. (2003). Heuristics for the traveling salesman problem. *Linkoping University*, 38:00085–9.

Open Robotics (2023). *Using Fast DDS Discovery Server as discovery protocol*.

Peng, C.-C., Huang, J.-J., and Lee, H.-Y. (2022). Design of an embedded icosahedron mechatronics for robust iterative imu calibration. *IEEE/ASME Transactions on Mechatronics*, 27(3):1467–1477.

Peng, Y., Qu, D., Zhong, Y., Xie, S., Luo, J., and Gu, J. (2015). The obstacle detection and obstacle avoidance algorithm based on 2-d lidar. In *2015 IEEE international conference on information and automation*, pages 1648–1653. IEEE.

Perko, E., Venator, E., and Martin, S. (2013). *nmea_navsat_driver ROS documentation*.

Phan, T.-D., Duong, M.-T., Nguyen, C.-T., Ly, H.-P., Le, M.-H., et al. (2022). Sensor fusion of camera and 2d lidar for self-driving automobile in obstacle avoidance scenarios. In *2022 International Workshop on Intelligent Systems (IWIS)*, pages 1–7. IEEE.

Philippart, V. Y., Snel, K. O., de Waal, A. M., Jeedella, J. S. Y., and Najafi, E. (2019). Model-based design for a self-balancing robot using the arduino micro-controller board. In *2019 23rd International Conference on Mechatronics Technology (ICMT)*, pages 1–6.

Raspberry Pi Foundation (2022). I2c clock-stretching bug · issue #4884 · raspberrypi/linux. Accessed: 29/05/2023.

Ratan, V. (2017). Docker: A favourite in the devops world.

Roweis, S. and Ghahramani, Z. (1999). A unifying review of linear gaussian models. *Neural computation*, 11(2):305–345.

Stahl, M. (2013). Rtklib package summary.

Tai, L. and Liu, M. (2016). Deep-learning in mobile robotics-from perception to control systems: A survey on why and why not. *arXiv preprint arXiv:1612.07139*, 1.

Takasu, T. (2009). RTKLIB: Open Source Program Package for RTK-GPS. In *FOSS4G 2009*, Tokyo, Japan.

Takasu, T., Kubo, N., and Yasuda, A. (2007). Development, Evaluation and Application of RTKLIB: A program library for RTK-GPS. In *GPS/GNSS Symposium 2007*, Tokyo, Japan.

Takasu, T. and Yasuda, A. (2009). Development of the low-cost RTK-GPS receiver with an open source program package RTKLIB. In *International Symposium on GPS/GNSS*, International Convention Center Jeju, Korea.

Tedaldi, D., Pretto, A., and Menegatti, E. (2013). A robust and easy to implement method for imu calibration without external equipments.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press.

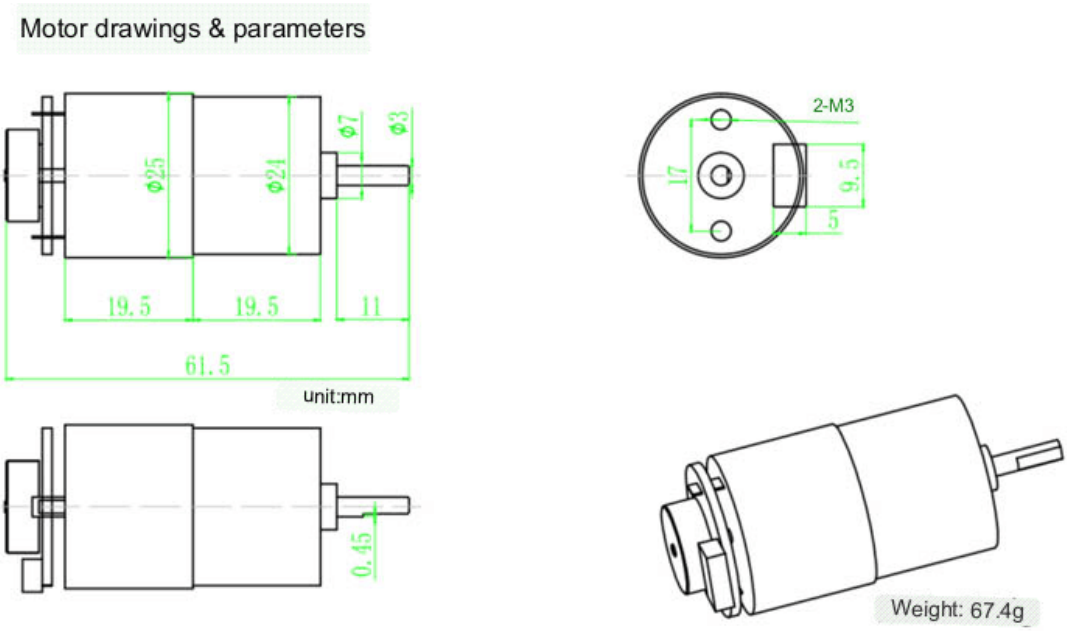
u-blox (2016). Achieving centimeter level performance with low cost antennas - white paper.

Vardi, Y. and Zhang, C.-H. (2000). The multivariate l 1-median and associated data depth. *Proceedings of the National Academy of Sciences*, 97(4):1423–1426.

Wei, S. X., Dixit, A., Tomar, S., and Burdick, J. W. (2022). Moving obstacle avoidance: A data-driven risk-aware approach. *IEEE Control Systems Letters*, 7:289–294.

Wenzel, P., Schön, T., Leal-Taixé, L., and Cremers, D. (2021). Vision-based mobile robotics obstacle avoidance with deep reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14360–14366. IEEE.

A. Appendix A: Datasheets



MODEL model	Voltage 电压	Gear Ratio	No Load (Empty)		Rated (load)				Stall	
			Speed 转速	Current current	Torque 扭力	Speed 转速	Current current	Output power	Torque 扭力	Current current
	IN		game	mA	Kg. cm	rpm	mA	IN	Kg. cm	A
25GA20E260	7.4V	1:20	500±13%	≤220	0.4±	400±13%	≤500	1.6	1.5±	≤2.0

Figure 1: Datasheet of the replacement DC motors (with encoders attached) with translated text (Chinese to English) using Google Translate

B. Appendix B: Figures and Tables

Table 1: A comparative summary of the Boundary and Obstacle Avoidance methods used by various existing autonomous lawnmowers and our own system developed as part of this paper.

Lawnmower	**Boundaries**				**Obstacle Avoidance**				
	Guide Wire	**Walk-around**	**Physical (virtual mapping)**	**Aerial Photography**	**LiDAR**	**Sonar/ Ultrasound**	**Vision**	**Collision Sensor**	**AI**
Luba	✗	✓	✗	✗	✗	✓	✗	✓	✗
Nova	✗	✗	✓	✗	✗	✗	✓	✗	✓
Conga	✗	✗	✓	✗	✗	✗	✓	✓	✓
Ardumower	✓	✓	✗	✗	✗	✓		✓	✗
RoboUP	✗	✓	✗	✗	✗	✓	✓	✗	✓
LawnMeister	✗	✗	✓	✗	✗	✗	✓	✗	✓
HonyMow	✗	✗	✓	✗	✓	✗	✓	✗	✓
MowRo	✓	✗	✗	✗	✗	✗	✗	✓	✗
OpenMower	✗	✓	✗	✗	✗	✗	✗	✗	✗
Our Solution	✗	✗	✗	✓	✓	✗	✗	✗	✗

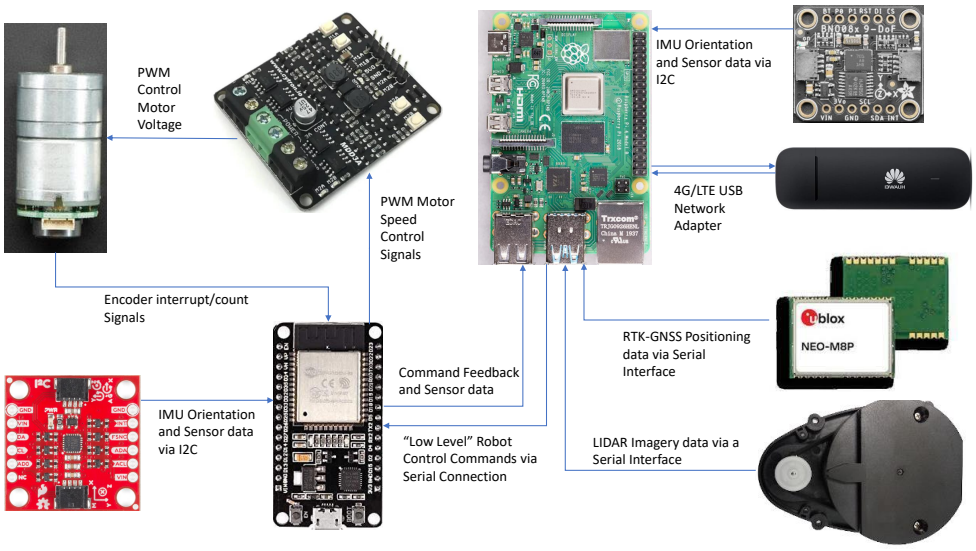


Figure 2: System Architecture of the Robot Platform

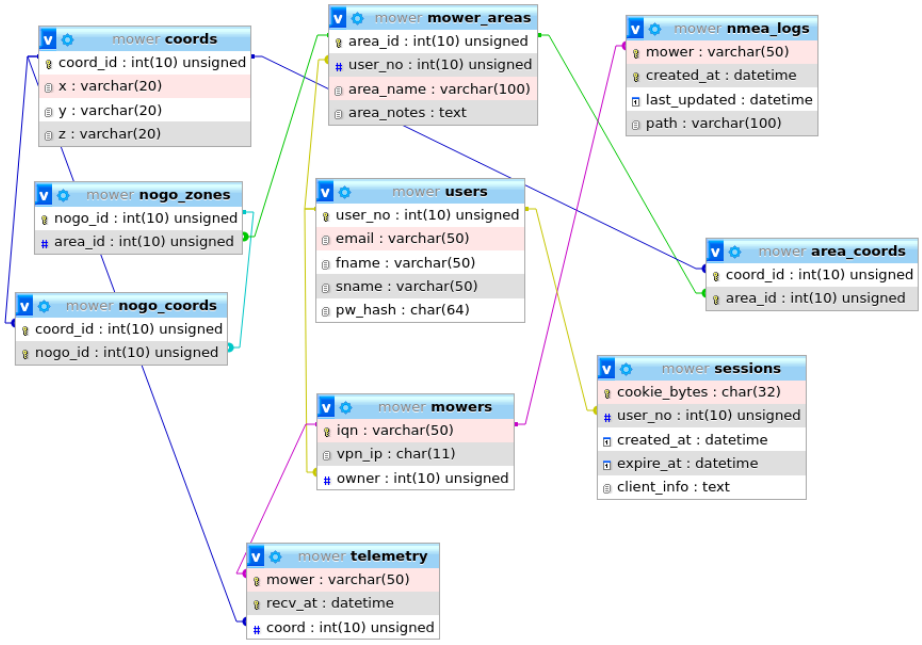


Figure 3: Database Entity Relationships Diagram

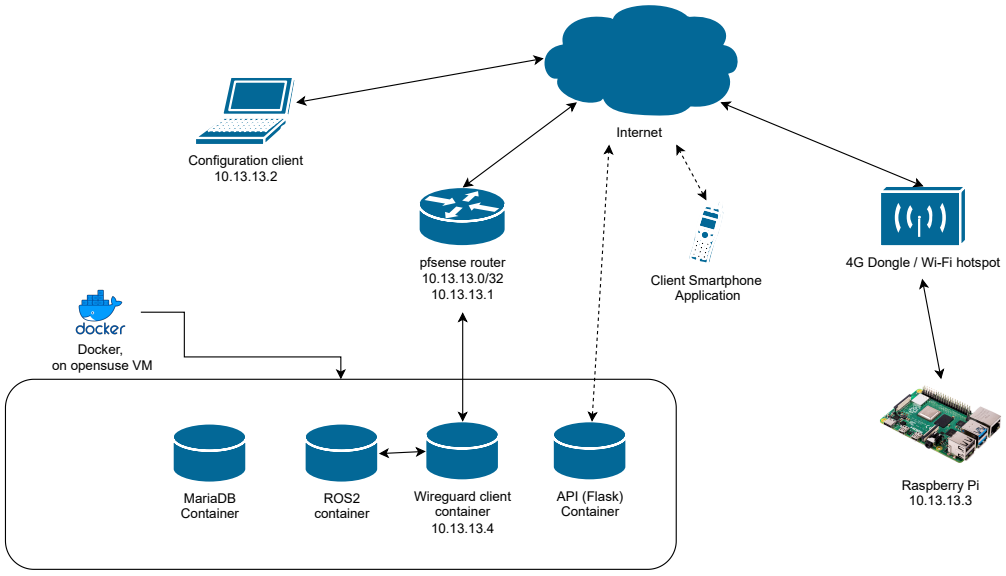


Figure 4: Network Architecture

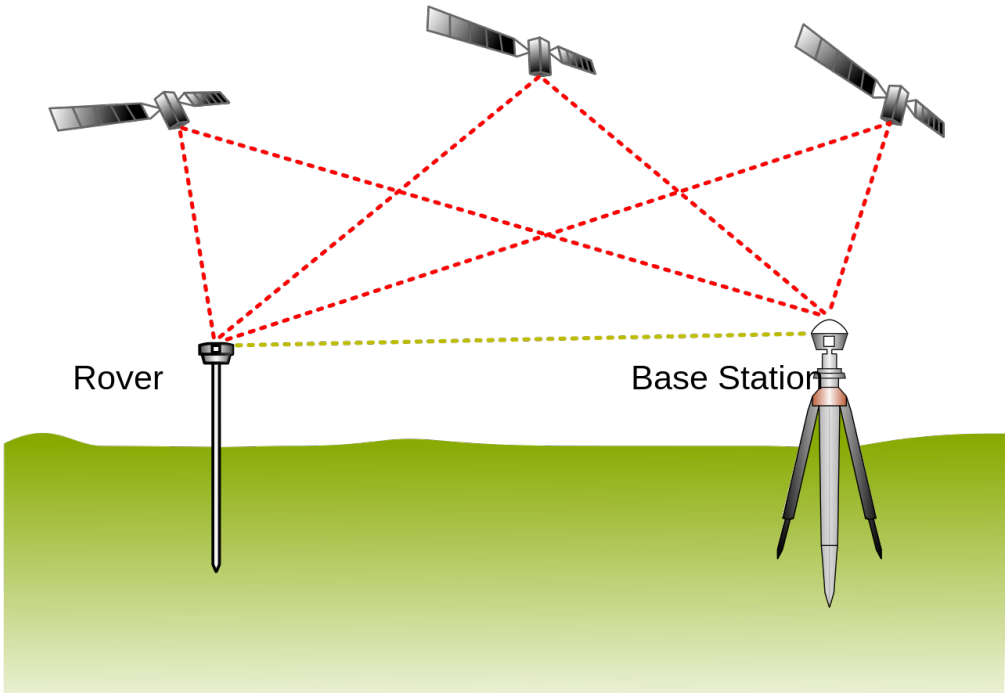


Figure 5: RTK Concept. Image released under CC BY-SA 4.0 by TS Eriksson

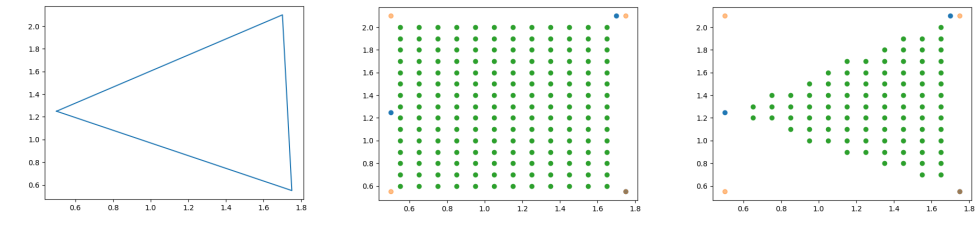


Figure 6: Generating Grid Points

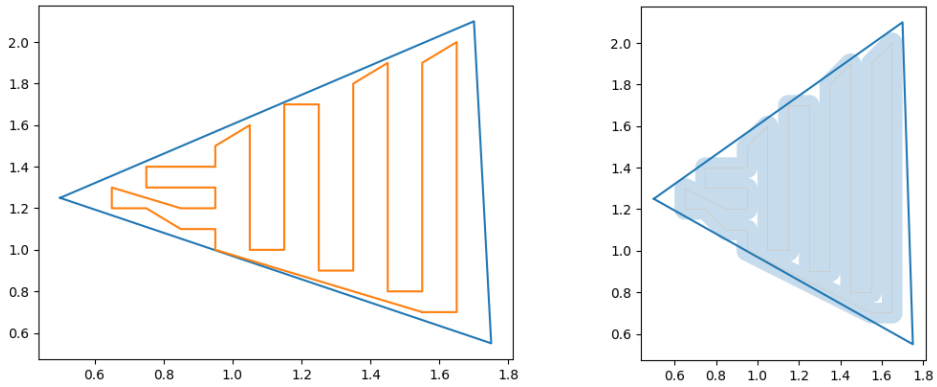


Figure 7: Using TSP to Produce Path

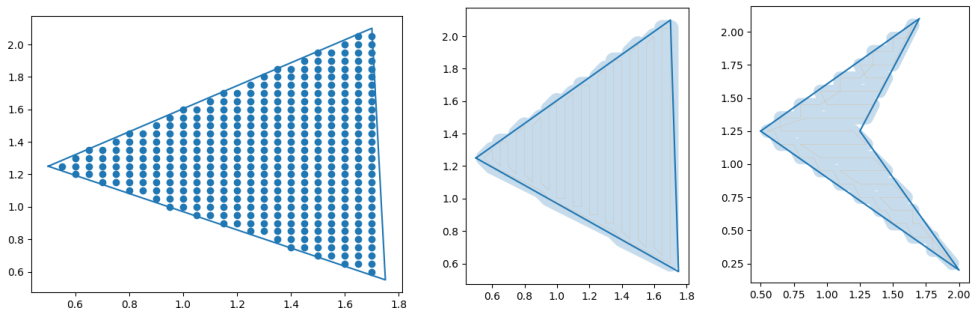


Figure 8: Increasing the Grid Resolution

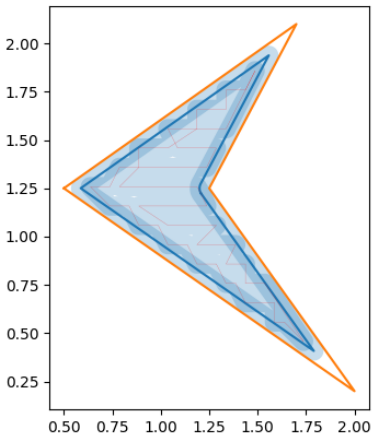


Figure 9: Using an Inner Perimeter to Reduce Overflow

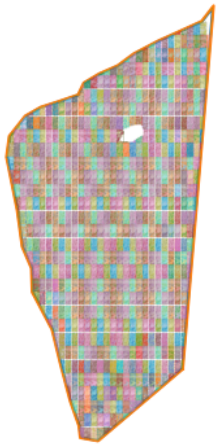


Figure 10: Golf Course using Macro Chunks

Figure 11: Removing Intermediate Points in the Path

Figure 12: Defining Virtual Boundaries Using the Mobile Application

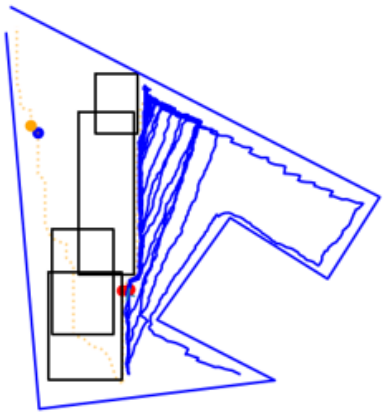


Figure 13: Inaccessible Target Without Reducing Points

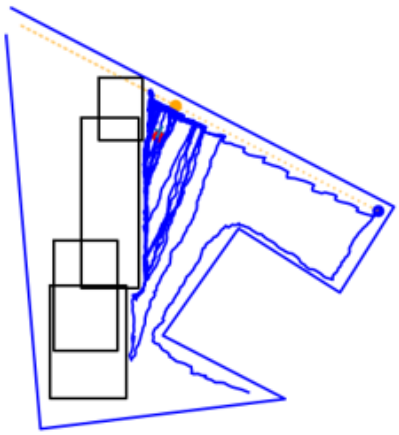


Figure 14: Inaccessible Target With Reducing Point

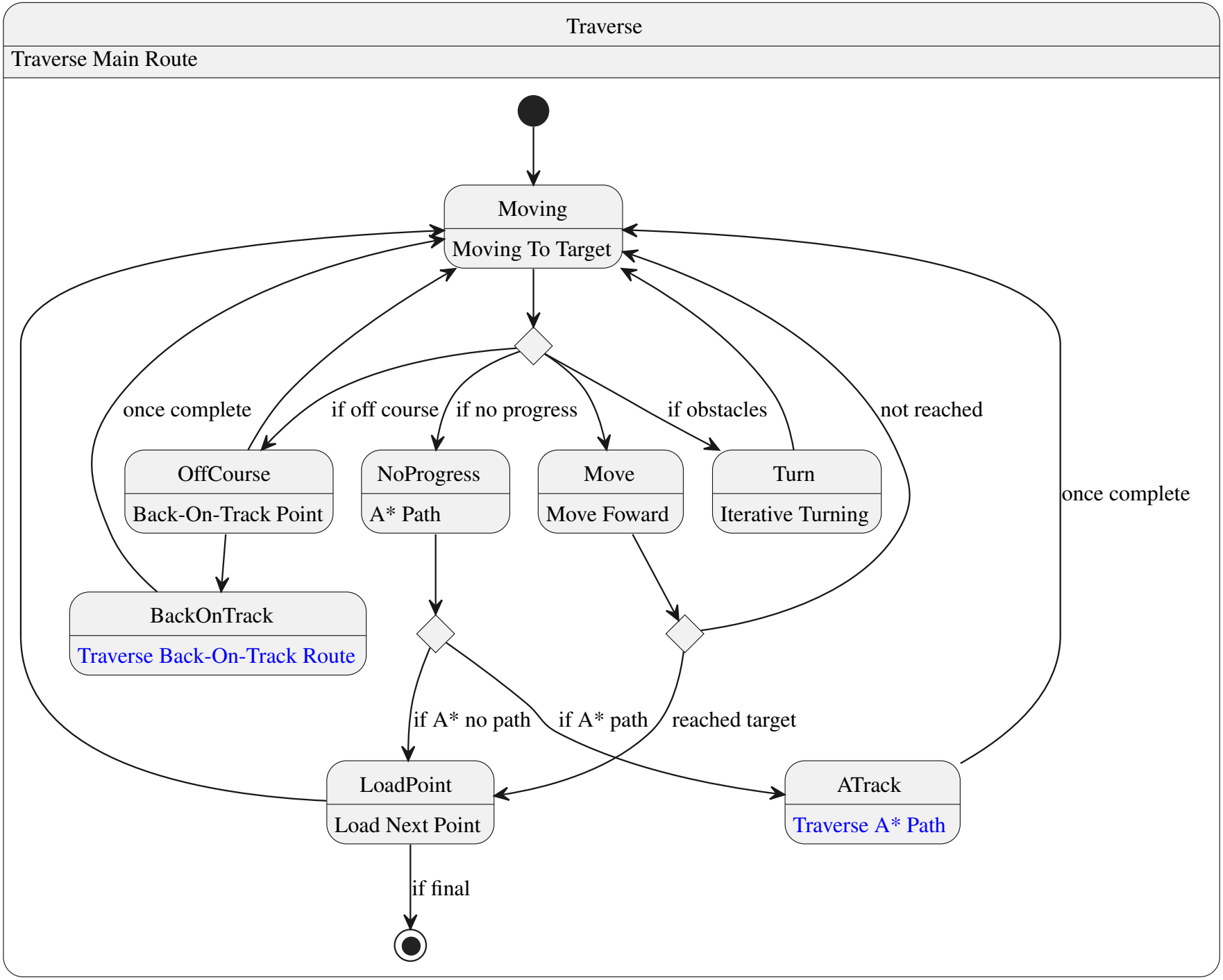


Figure 15: Navigation Flow Diagram

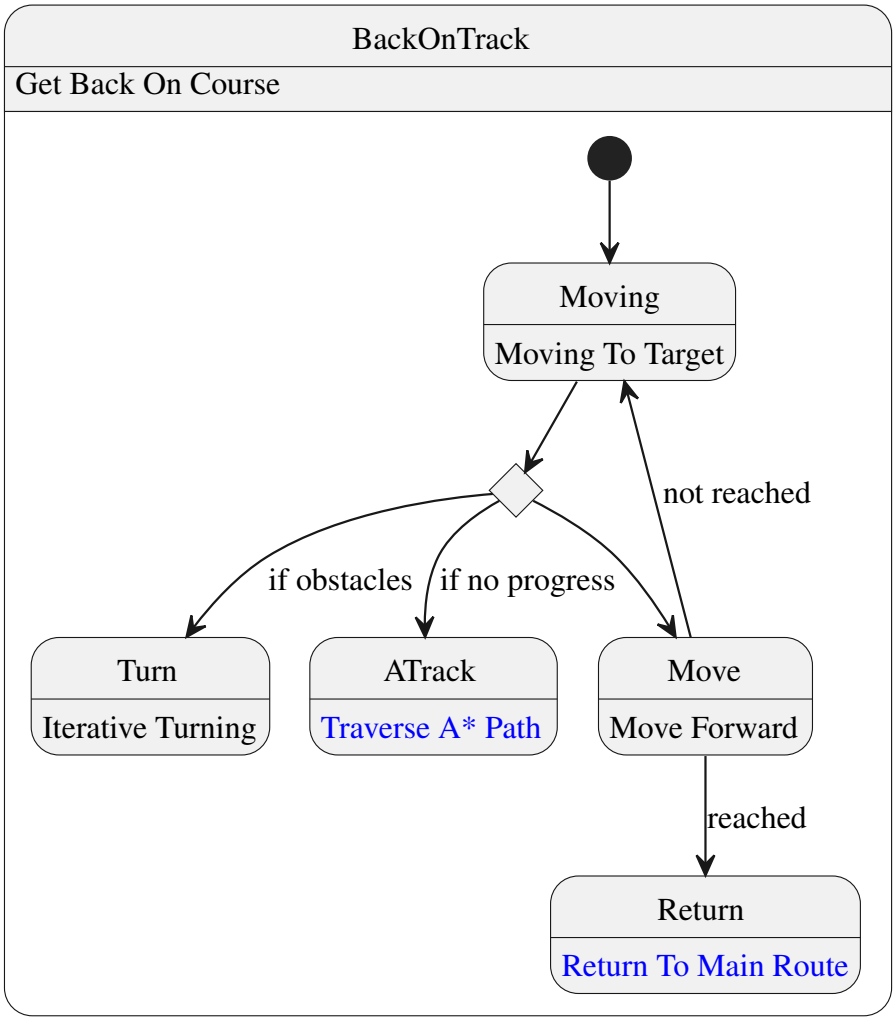


Figure 16: Back On Track Flow Diagram

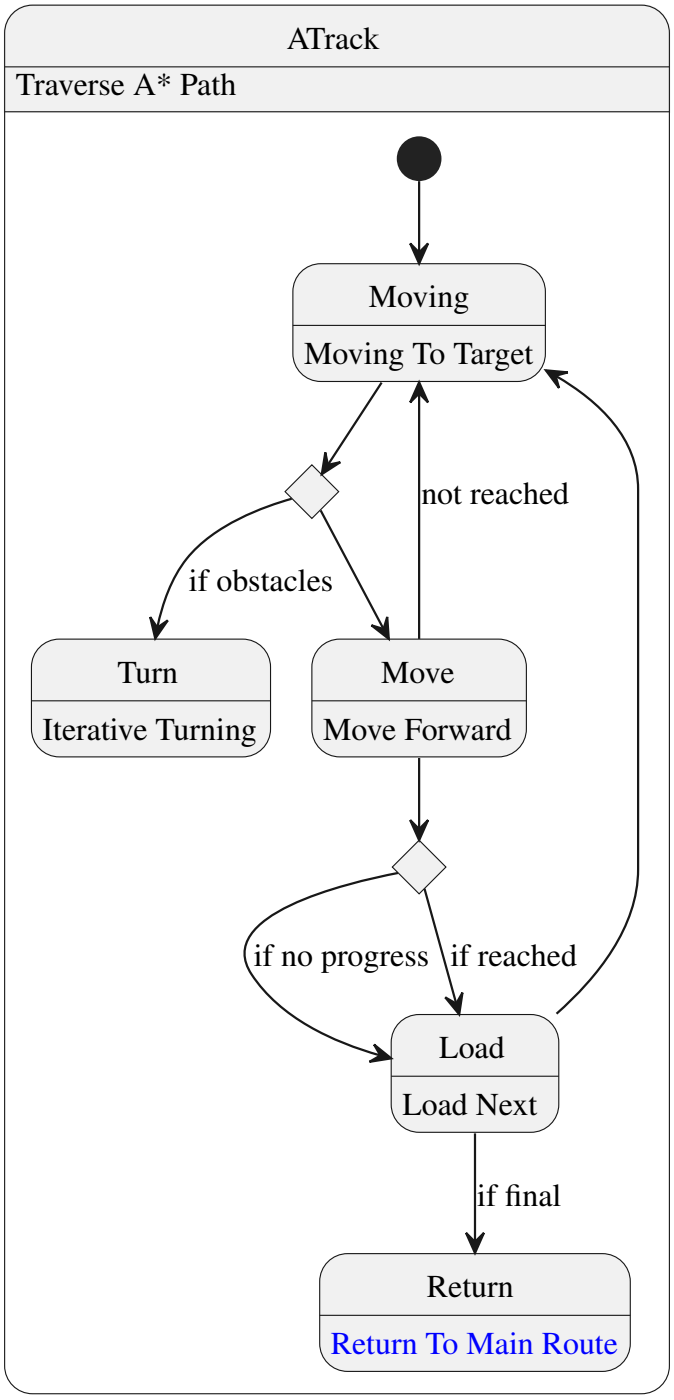


Figure 17: A* On Line Path Flow Diagram

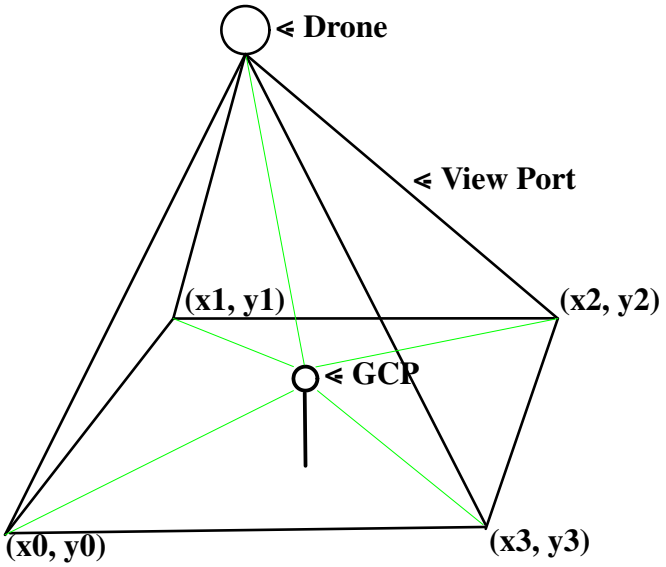


Figure 18: Aerial Imaging Using a GCP for Higher Accuracy Geo-Referencing

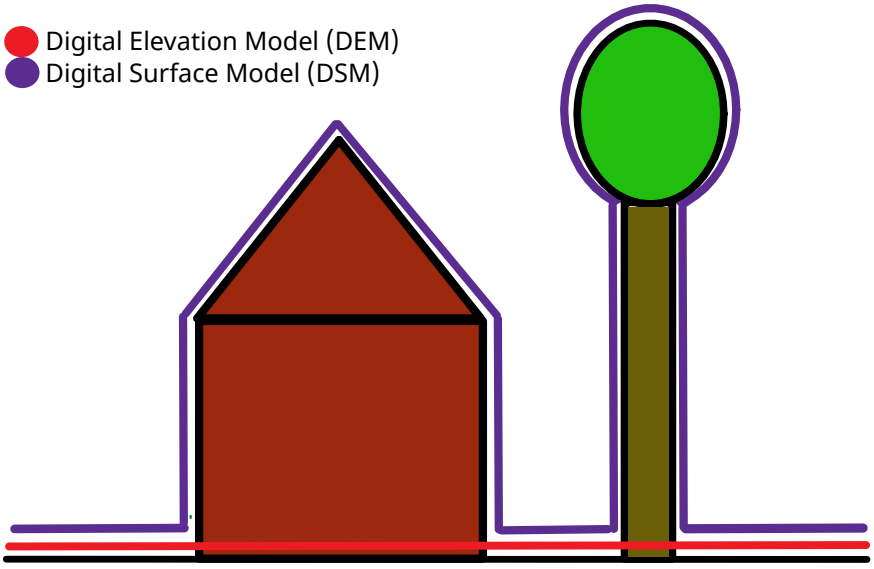


Figure 19: Digital Elevation Model Compared with Digital Surface Model

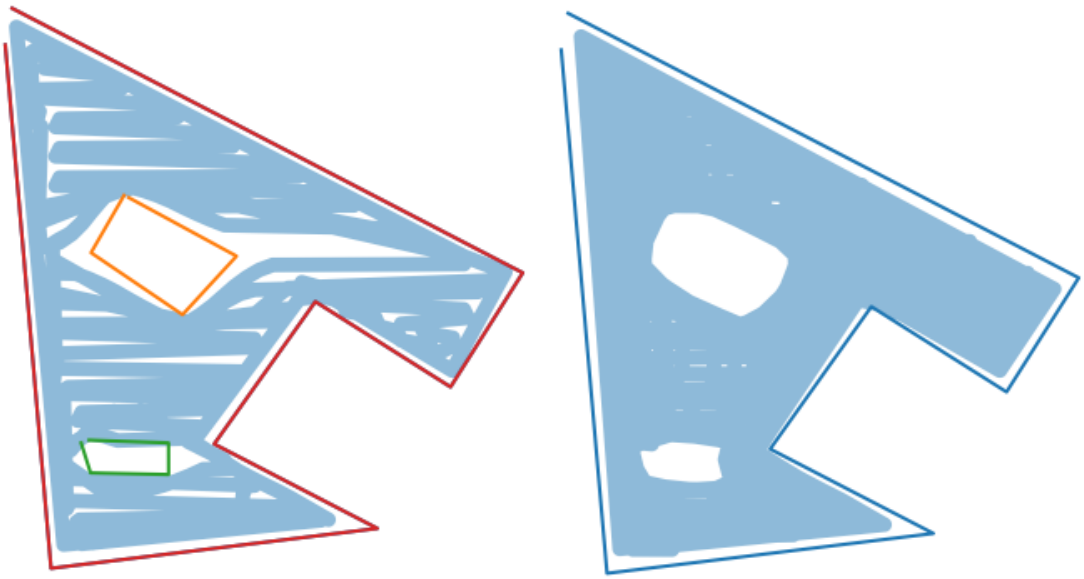


Figure 20: Simulation Result: One Route (Left), Two Crossing Routes (Right)

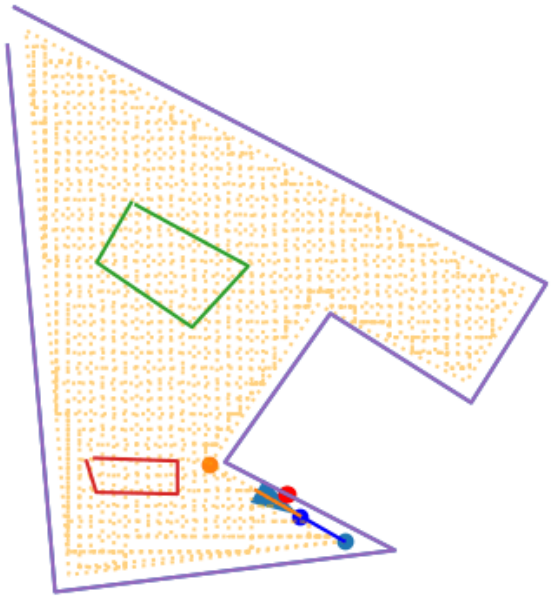


Figure 21: Combined Navigation Simulation (This animated figure may not work in all file viewers. This, and others, can be found in this GitHub Repo)

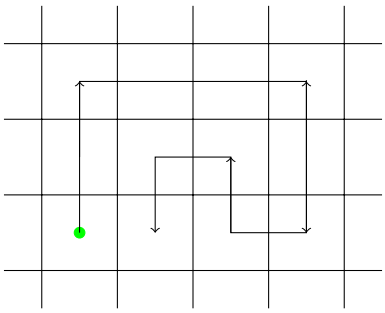


Figure 22: Grid-Based Pathing

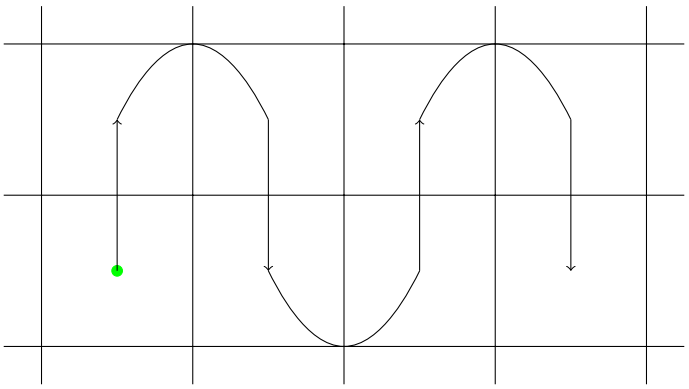


Figure 23: Headlands and Swathes Based Pathing

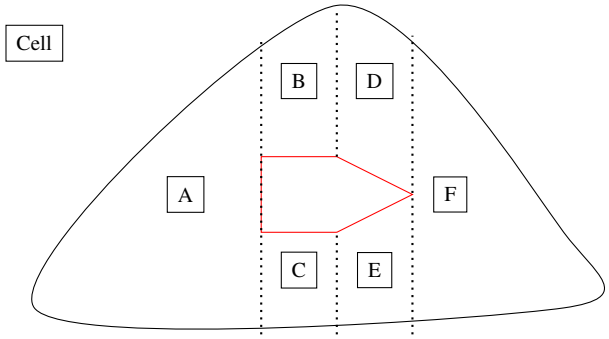


Figure 24: Cell Decomposition

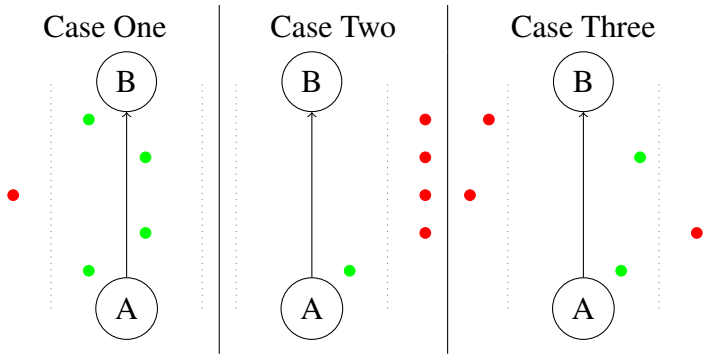


Figure 25: Coverage Path with Buffer

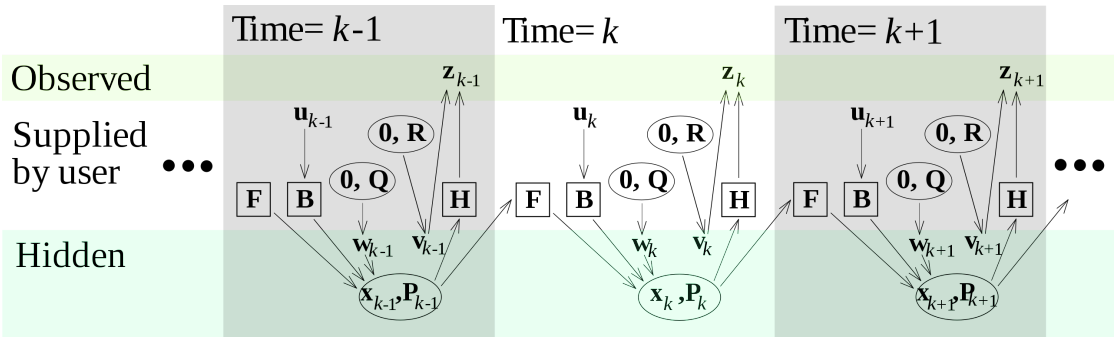


Figure 26: Hidden Markov Chain underlying the Kalman filter. Image under public domain

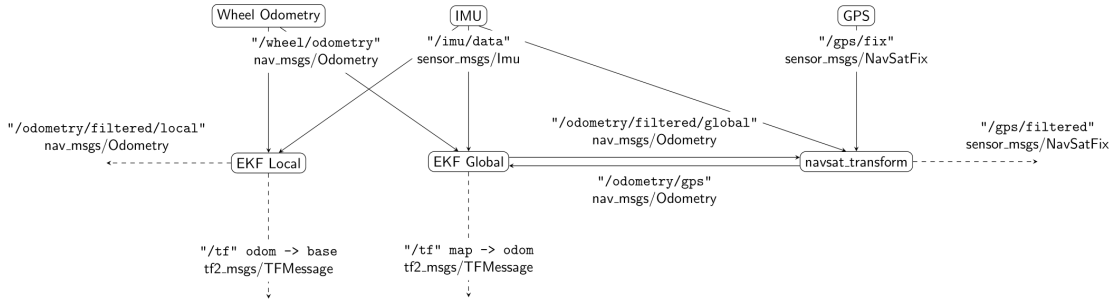


Figure 27: Moore and Stouch (2016) localization workflow

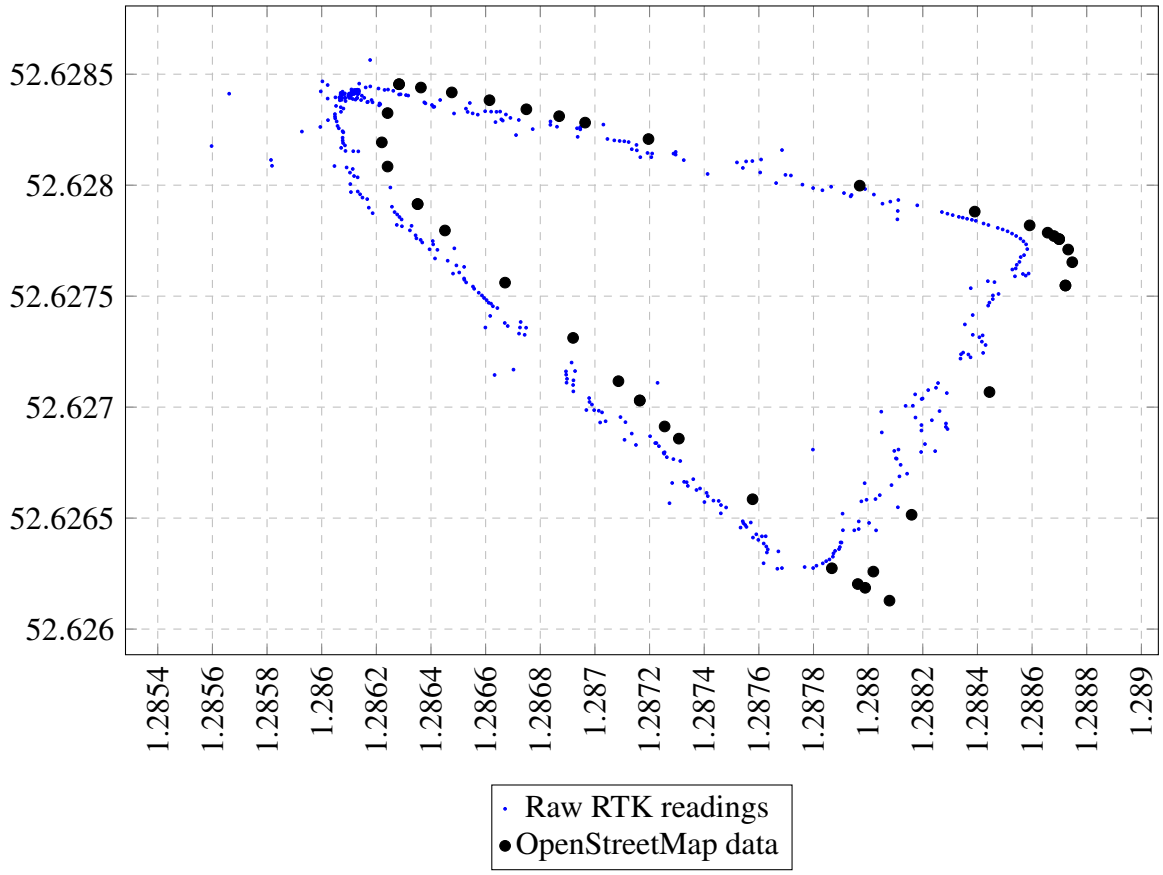
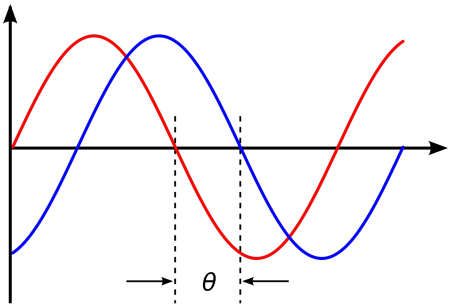
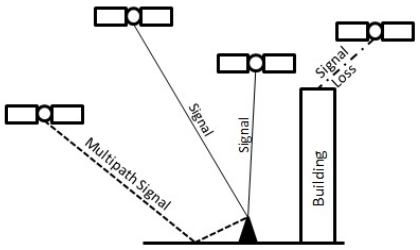


Figure 28: Plot showing raw RTK readings, in addition to OpenStreetMap data from a test. OSM data is licensed under Open Data Commons Open Database License (ODbL) *https://www.openstreetmap.org/copyright*. The conclusions and setup for this test are described in Sec. 11.

Figure 29: RTK sensor setup for preliminary testing. The system was walked to be a similar speed that the robot would move at.

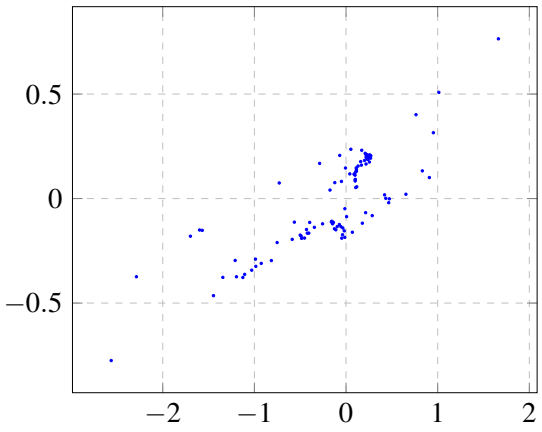


(a) Received waves that travel along different paths will arrive at the receiver phase shifted, thus contradicting each other. Image licensed under CC BY-SA 3.0 (*https://creativecommons.org/licenses/by-sa/3.0*) by user `Peppergrower`.

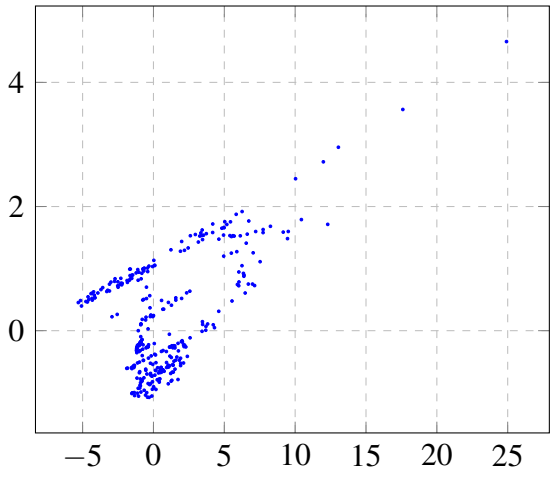


(b) Example of multipath signal propagation. Signals could additionally bounce off the building before arriving at the antenna to produce additional erroneous values. Image from Jyothsna and Raju (2021).

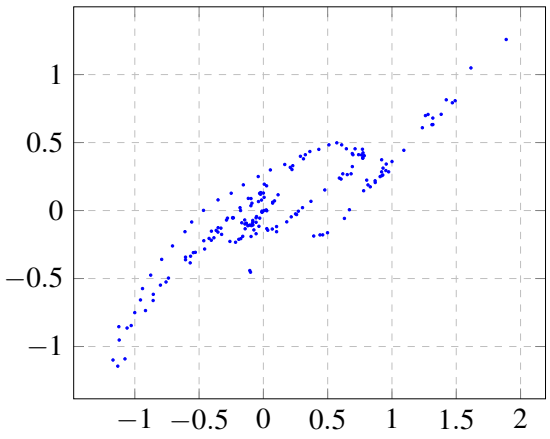
Figure 30: Figures showing how multipath error is propagated.



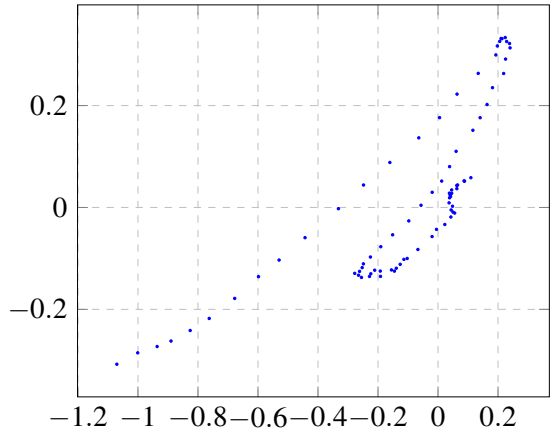
(a) Static RTK deviation tested near the sainsbury's center at the University of East Anglia



(b) Static RTK deviation tested near the sainsbury's center at the University of East Anglia

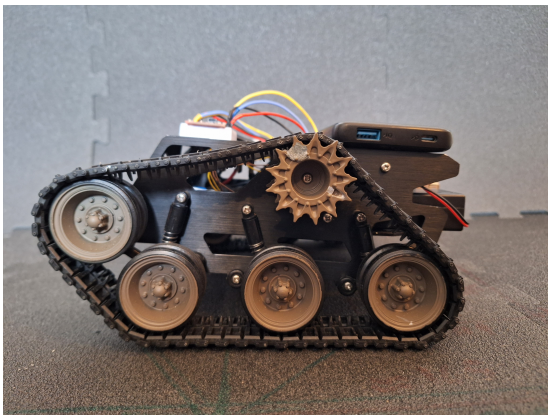


(c) Static RTK deviation tested at Chapelfield Gardens in central Norwich

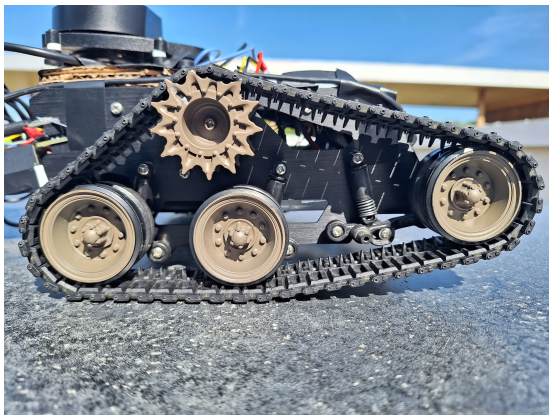


(d) Static RTK deviation tested at near the Castle Mall shopping center, within line of sight to the base station

Figure 31: The variation of RTK readings given with a static antenna in various locations around the base station. The standard deviation in readings when the robot is static can be used as a metric for RTK inaccuracies. Geometric median calculated with an implementation of Vardi and Zhang (2000). Distance is in UTM Meters.



(a) The track setup on the robot before having to alter the tension and surface contact with the ground



(b) The track setup on the robot after altering the tension and the amount of surface contact the robot has to the ground

Figure 32: A before and after view of the track setup on the robot after altering the tension and the amount of surface contact the robot has to the ground to help solve the issue of the robot's tracks getting stuck on uneven surfaces when rotating on the spot