Multi-Agent SLAM: Exploration and Mapping in Simulated testing Environments

Charlie Anthony [CandNo: 246537]
Supervisor: Dr Chris Johnson

Dissertation
Computer Science and Artificial Intelligence BSc

Department of Informatics and Engineering
University of Sussex
May 2024

Contents

1 Introduction

Swarms exist everywhere in life. Nearly all organisms exhibit some form of swarming behaviours within their communities. Starlings display impressive organisational behaviour, positioning themselves with respect to the movement of their neighbours. Humans show swarm behaviours when moving in crowds, for example, moving around sports venues or exiting buildings in emergencies. No matter how hard you look, regardless if the context, swarms are typically present.
These behaviours can also be artificially created in robotics. Within the realm of computing, parallelising processes is breaking barrier after barrier - swarm robotics brings the same benefits. Being able to divide and conquer a problem has the ability to massively increase the rate of work by employing multiple robots. Therefore, it would be wasteful not to properly dedicate the time which this discipline deserves.
For my project, I am going to try and reproduce some of these behaviours artificially. I will start by simulating robotic agents in a 2D environment. The agents will be placed within close proximity inside a simulated environment and then allowed to explore and combine their findings; ultimately creating a visualization map of its environment. The agents will need to both navigate the environment and avoid collisions, whilst creating an internal representation of its surroundings. The best-case scenario for the agents within the swarm is to be fully independent; creating a decentralized system.
I will initially explore this problem by creating SLAM simulations, and then attempting to apply similar techniques to a centralised system. These initial simulations will employ techniques such as graph-based SLAM, random walks and other elements of swarm behaviours in order to create a base-line representation of the environment. This project will also have the flexibility to potentially implement physical robots, given time permits.

2 Literature Review

2.1 Introduction to SLAM

SLAM (Simultaneous localization and Mapping) is a technique used in robotics to create a map of an unknown environment (1). It is an important area of research in robotics as it is heavily used in autonomous vehicles, drones and vacuum cleaners; allowing agents to understand and navigate their environment effectively. Figure 1 shows an example of a robot scanning its environment, with the sensors visually added to the image.
SLAM can be broken down into two sub-problems: localization and mapping. Localization is the process of determining the location of a robot in its environment, whilst mapping is the process of constructing a map of the environment. The maps are constructed using data collected from sensors, such as cameras and laser scanners; Figure 2 shows this. A lot of existing work in SLAM is based on single robot applications, however, there is a growing interest in multi-agent SLAM.
Code implementations of SLAM usually divide the problem into the Front-end and the Back-end (2). The Front-end is where the agent interprets the environment; it will receive data, which could be in the form of images, LiDAR scans or other sensor data. The Front-end will then process this data, extracting features and identifying landmarks. Furthermore, the Front-end will also perform data association, which will compare new features/landmarks to data that has been previously collected. The Back-end is where

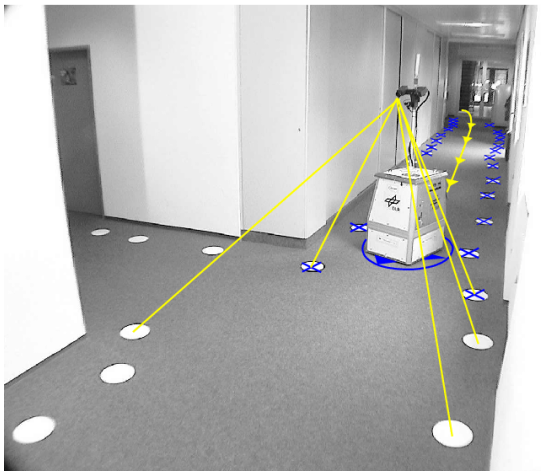

Figure 1: A visual representation of a robot scanning its environment (reproduced from (1))

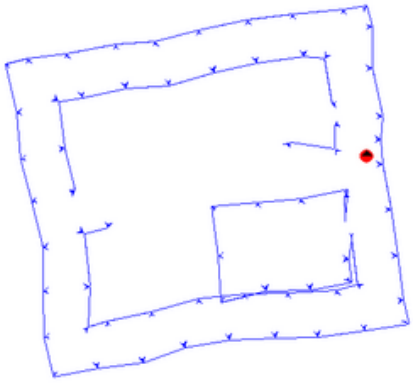

Figure 2: The map (after loop closure) produced by the robot's SLAM algorithm of its environment (reproduced from (1))

the agent will use the data collected from the Front-end to create a map of the environment and localise itself. There are a number of paradigms which may be used during this process, such as Graph-based SLAM, Particle filtering and Extended Kalman Filters. By combining the Front-end and the Back-end, the agent will be able to create an internal representation of its environment and its position in that environment.

2.2 Background and Evolution of SLAM

In the mid 1980s, SLAM concepts were first introduced by Smith and Cheeseman (3). They initially laid out the foundations of the problem, which was to be able to reason about the position of an object with potentially inaccurate information about the environment. Their paper demonstrates a lot of themes which we now associate with SLAM, such as taking frames (poses) which have an associated positional uncertainty, taking measurements of the environment at each frame and then using these measurements to identify objects in the environment. Smith and Cheeseman used the Kalman Filter equations for static-state estimations, and then merged these state estimations to create a representation of the environment.

The next major developments in SLAM came shortly after, in the early 90s, from the work of Leonard and Durrant-Whyte. In their paper, they identify further challenges in SLAM, such as the data association problem and environment dynamics (4). They also outlined one of the traditional problems with SLAM, which is the "chicken and egg" problem. This arises as to build a useful map, the robot needs to know where it is, and to know where it is, it needs a useful map. To tackle this, they proposed the use of an Extended Kalman Filter (EKF), which builds on the traditional Kalman Filter equations by allowing for non-linear state estimations. This was a significant development, as it allowed for the creation of more accurate maps; however they also concluded that the EKF was not suitable for large-scale environments, as data association becomes increasingly difficult.

Come the 2000s, SLAM had become a well-established area of research, with a number of different paradigms being used to approach the problem. One of which was FastSLAM,

which was developed by Michael Montemerlo and Sebastian Thrun (5). FastSLAM was an attempt to solve one of the fundamental flaws of EKF SLAM, which was scalability. FastSLAM works by using a particle filter to estimate the robot's pose, and then constructs a map of the environment using these poses. A particle filter is a probabalistic technique used to estimate the robots position, which works by having a set of particles randomly distributed across the environment, which represent possible positions the robot could be in. As the robot moves, the particles are adjusted based on measurements from the robots sensors; the position of the robot is then estimated by taking the average of the particles. FastSLAM main strengths are its ability to handle large environments and improve the computational complexity of EFK SLAM, however it's weaknesses are that it is not as accurate as EKF SLAM and it struggles with dynamic environments.

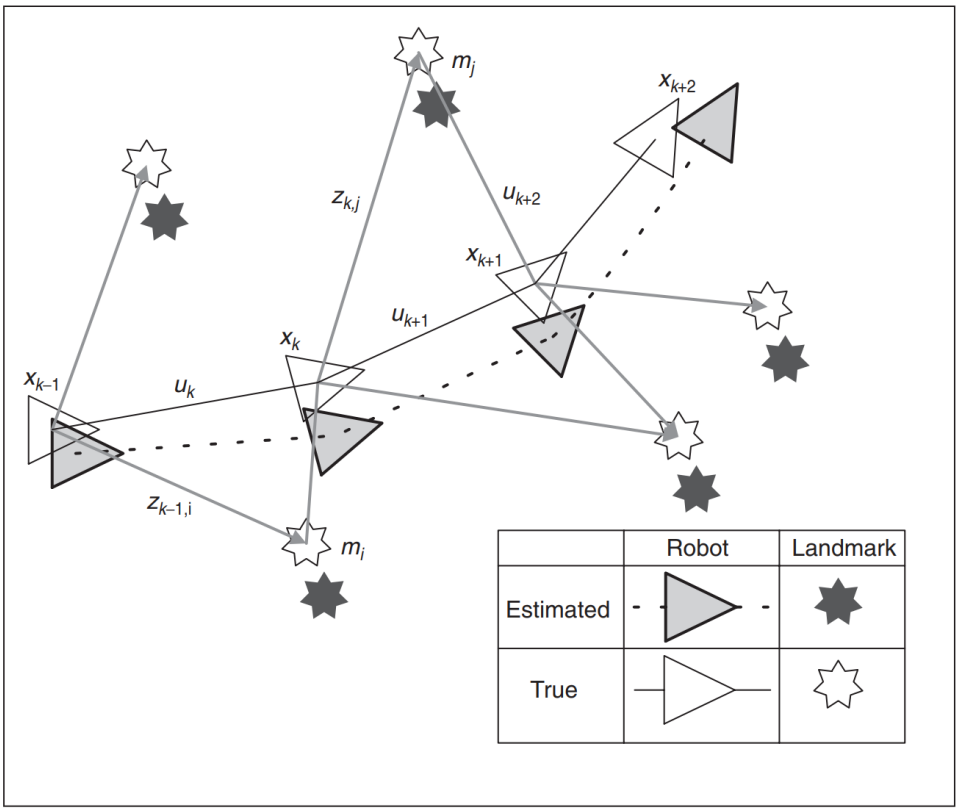

Figure 3: Formal representation of the SLAM problem (reproduced from (6))

Finally, in 2006 Durrant-Whyte and Bailey wrote a significant paper outlining the future of SLAM (6). They started by discussing existing approaches to the problem and then formally outlining the SLAM problem, shown in Figure 3. They then went into further detail on how both EFK SLAM and FastSLAM work, and then discussed the limitations of each approach. They also wrote a second paper, discussing the future of SLAM, including multi-agent SLAM, 3D SLAM and Dynamic environments (7).

2.3 Technical Frameworks and Models

2.3.1 Mapping techniques

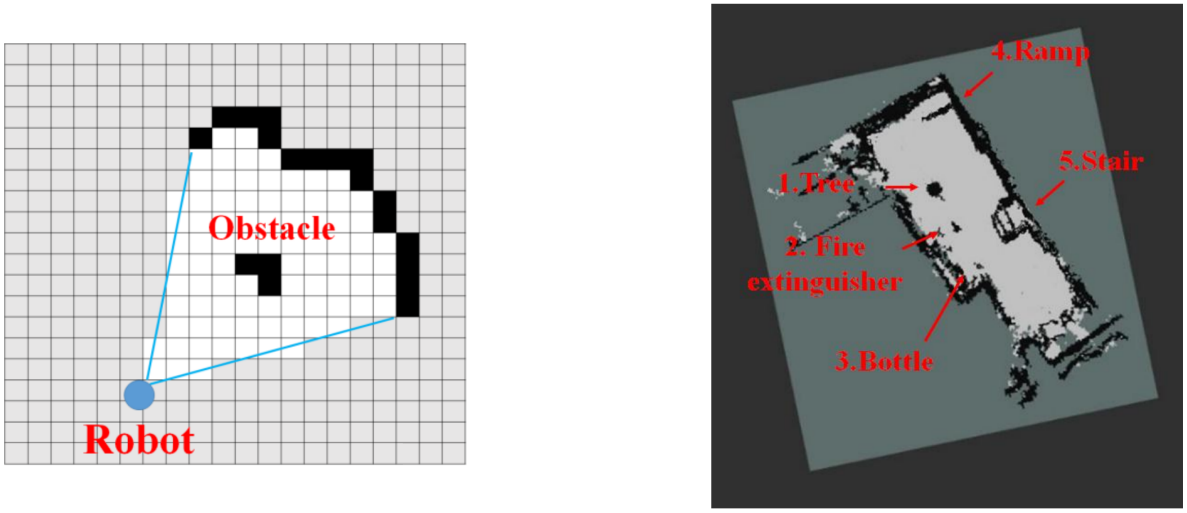

Figure 4: Example of simple (left) and complex (right) occupancy grids (reproduced from (8))

Mapping techniques are used by the agent to create a representation of its environment. There are a number of different mapping techniques commonly used in SLAM, such as grid maps, feature-based maps and semantic maps. Grid maps, or occupancy maps, are the most common type of map used in SLAM, as they are easy to implement and computationally efficient. They are typically represented as a 2D array, which simply stores a binary value for each cell, which represents whether the cell is occupied or not. Figure 2.3.1 shows a simple example of an occupancy grid, where the black cells represent occupied space and the white cells represent free space. Figure 2.3.1 demonstrates how an occupancy grid can be used to represent a more complex environment, such as the experiment environment used by Nam et al (8). Occupancy grids are typically implemented in conjunction with distance sensors, such as LiDAR - this allows the agent to measure the distances to multiple points in the environment, which can then be used to update the occupancy grid.

Feature-based maps are another common type of map used in SLAM. They work by having the agent detect features in the environment, such as walls, corners and edges. These features are then used to define landmarks, which are then used to create a map of the environment. Landmarks are distinct features in the environment which can be used to localise the agent; a common metaphor used to describe landmarks is a distinctive skyscraper in a city, like the Shard in London, which you could use to figure out your location. Feature-based maps are also implemented with distance sensors, as the agent can use points detected by the sensor to detect features.

Finally, semantic maps are a more recent addition to SLAM, which work by allowing the agent to detect and classify objects in the environment. This is achieved using computer vision techniques, such as object detection and image segmentation. Figure 5 shows an example of a semantic map; you can see the image segmentation has identified various objects, such as the keyboard and the books. Semantic maps are used more frequently in research whilst less often in industry, as they are much more computationally expensive

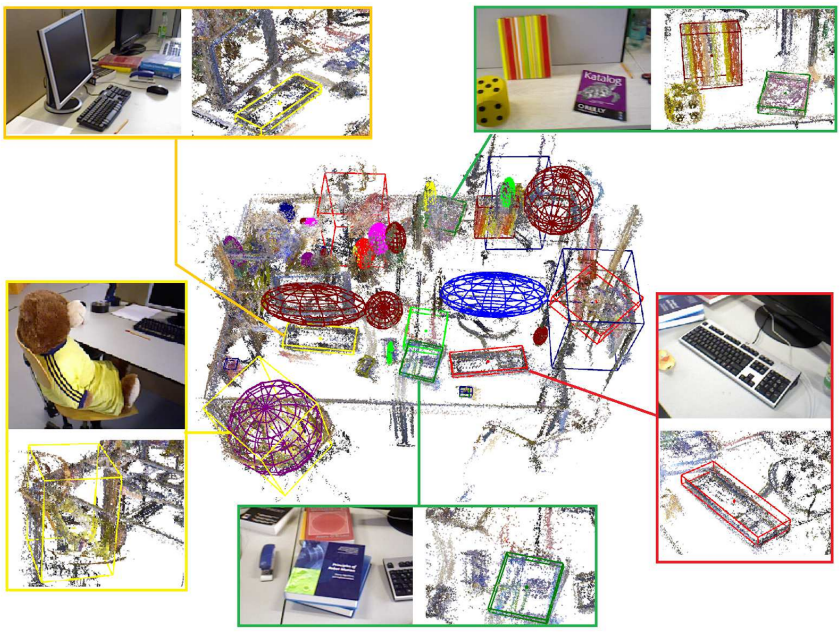

Figure 5: Example of a semantic map (reproduced from (9))

and require more complex sensors, such as cameras.

2.3.2 Localization techniques

Localization techniques are used to help the agent determine its position in a given environment. There are a number of popular techniques commonly used in SLAM, such as Kalman Filters, Particle Filters and Visual Odometry. Kalman Filters are a set of mathematical equations which are used to estimate the position of the agent given a set of noisy sensor readings. The equations are split into two main functions, the prediction function and the update function (10). The prediction function is used to predict the agents position based on the previous state, which incorporates uncertainty from the agents movement. The update function is then used to update the agents position based on new sensor readings and update the uncertainty. The prediction functions are defined as:

- State Prediction Equation:

$$\hat{x}_{k+1} = f(\hat{x}_k, u_k, 0) \tag{1}$$

- Covariance Prediction Equation:

$$P_{k+1} = A_k P_k A_k^T + W_k Q_k W_k^T \tag{2}$$

The state prediction equation predicts the next state $\hat{x}_{k+1}$ based on the previous state $\hat{x}_k$, the control input (the agents desired movement) u_k and the process noise (which is assumed to be zero here). The covariance prediction equation then updates the uncertainty of the agents position based on the state transition matrix A_k, the previous position uncertainty P_k, the control input matrix W_k and the process noise covariance matrix Q_k. The update functions are defined as:

- Kalman Gain Equation:

$$K_k = P_k H_k^T (H_k P_k H_k^T + V_k R_k V_k^T)^{-1} \tag{3}$$

- State Update Equation:

$$\hat{x}_k = \hat{x}_k + K(z_k - h(\hat{x}_k, 0)) \tag{4}$$

- Covariance Update Equation:

$$P_k = (I - K_k H_k) P_k \tag{5}$$

The Kalman Gain equation determines how impactful the new sensor reading z_k should be, based on the previous position uncertainty P_k. H_k is the Jacobian of the measurement function $h(\hat{x}_k, 0)$ with respect to the state, and V_k is the Jacobian of the measurement function with respect to the measurement noise. The state update equation then updates the agents position based on the Kalman Gain. Finally, the covariance update equation updates the uncertainty of the agents position based on the Kalman Gain. One of the limitations of the Kalman Filter equations is that they are only valid for linear functions, which makes them less useful in practice. To adapt the Kalman Filter to non-linear functions, the Extended Kalman Filter (EKF) is used. The EKF works by linearising the state transition and measurement functions, which allows the Kalman Filter equations to be used. The specifics of the EKF equations will be discussed in the methods section, as I plan on using the EKF in my project.

Particle Filters, or Monte Carlo Localization, is ano ther popular technique used in

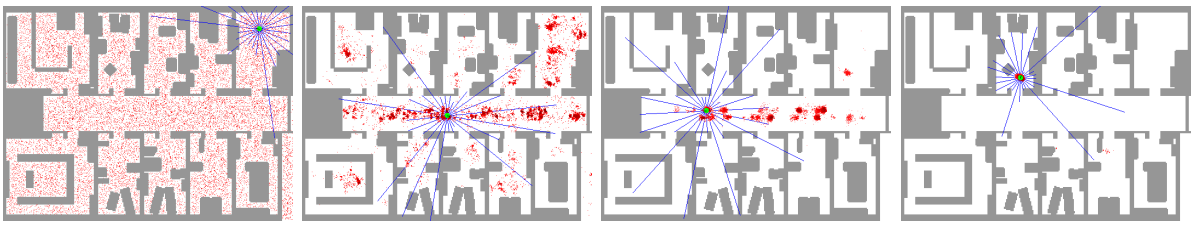

Figure 6: Example of Particle Filter in a 2D environment (reproduced from (11))

SLAM, which work by having a set of particles randomly positioned across the environment which represent the possible positions of the agent (posterior). Each particle is assigned a weight, which represents the probability that the particle's position is the true position of the agent. As the agent moves, the particles are updated based on sensor readings, which are used to adjust the weights of the particles. The problem can be represented as:

$$w_i \propto p(z_t | x_i^t) \tag{6}$$

Where w_i is the weight of the particle, z_t is the observed measurement at time t and x_i^t is the state of the particle at time t. As the agent discovers more about its environment, the particles will converge towards the true position of the agent - as shown in Figure 6. Dellaert et al demonstrated the strengths of particle filters in their paper (12), where they showed that particle filters are able to localise the agent in complex environments, even when the environment is dynamic.

Finally, Visual Odometry works by having the agent estimate its position based on visual data. This is achieved using the previously mentioned semantic maps, which are used to detect and classify objects in the environment. As the name suggests, visual odometry only uses visual data to estimate the agents position; benefits of this approach are that cameras are cheap and lightweight, however they are also computationally expensive. Furthermore, visual odometry is less effective in environments with poor lighting and textureless surfaces, making it less reliable than other techniques.

2.3.3 Sensor technologies

There are many different types of sensor technologies used throughout SLAM research, such as LiDAR, cameras and Inertial Measurement Units (IMUs). LiDAR (Light Detection and Ranging) is a distance sensor which works by shooting out a laser and measuring the time it takes for the laser to bounce off an object (up to a given distance) and return to the sensor. Typically, the sensor will rotate 360 degrees and take measurements at regular intervals, which is particularly useful in SLAM as it allows the agent to sense its complete surroundings.
Cameras are another common sensor used in SLAM, which work by taking images of the environment and then using computer vision techniques to extract features and landmarks. One of the advantages of cameras is that they can capture a lot of information about the environment, such as objects and textures; which make data association easier. However, cameras are also a lot more computationally expensive in comparison to LiDAR, as feature extraction and landmark detection have a lot more complexity.
IMUs are used to measure the orientation, velocity and gravitational forces of the agent; this makes them useful in SLAM as they help record the agents movement. They are normally used alongside other sensors to help improve the accuracy of the localisation algorithm.

2.4 Algorithms and Implementations

2.4.1 Algorithms

Graph-based SLAM is a technique used to create a map of an environment, by using a graph to represent the environment. It works by having a robot move around its environment, whilst taking measurements of its surroundings. These measurements are usually received by a sensor, such as a camera or laser scanner. The robot then uses these measurements to create a plot of where objects may be, by combining the measurements from sensors with its memory of the route it has taken. This technique is used in many applications, both in research and in the real world.
One limitation of graph-based SLAM is that it is computationally expensive, as it requires a lot of memory to store the sensor readings. Also, it is not very scalable, as the more sensors that are added, the more memory is required. Another limitation is that it cannot always be reliable, as the algorithm depends heavily on detecting loop closures, which when not detected, can lead to a lot of errors in the map. This, combined with even the slightest inaccuracy from sensors/motors makes it difficult to apply to the real world.
Oriented FAST and Rotated BRIEF (ORB) SLAM is a real-time, visual SLAM algorithm

which works by using a camera to detect features in the environment. A real-time SLAM algorithm is one that can process data as it is received - this is significant for ORB SLAM as the camera input is very computationally expensive to analyse. ORB SLAM works by detecting features in the environment, such as corners and edges, and then using these features to create a map of the environment; typically done using the previously mentioned semantic maps. Originally, ORB SLAM was a development of bundle adjustment, which is a technique used to get accurate estimations of the agents positions, but isn't real-time (13). A popular use case of ORB SLAM is in autonomous vehicles, as modern vehicles are equipped with cameras and powerful computers, which can be used to run complex algorithms.
Hector SLAM is a more recent development in SLAM, which uses LiDAR to create a map of the environment. It is significant because it differs from other previously mentioned techniques by relying less on wheel odometry and other forms of motion estimation. Instead, Hector SLAM employs a scan matching algorithm, which works by comparing the current LiDAR scan to previous scans, and using the differences to estimate the agents position. Hess et al implemented Hector SLAM using a branch and bound algorithm, which allowed them to find the best match between scans (14); other implementations have used gradient descent to achieve similar results. Overall, this technique is particularly useful in environments where wheel information is less reliable, such as on rough terrain or on slippery surfaces.

2.4.2 Software

There are a number of existing tools and packages that can be used to implement and simulate SLAM algorithms; including Robot Operating System (ROS), Mobile Robot Programming Toolkit (MRPT), Webots, Enki, Gazebo and CoppeliaSim (formally V-REP). The mentioned resources can be split into two distinct groups: algorithm implementations and simulation environments. Packages like ROS and MRPT fall under the former, providing tried and tested SLAM algorithms. ROS is an open-source package which was created using C++ and Python, and is now widely used in both research and industry. More recently, development of ROS2 has picked up pace, which is a more modern version of ROS, aiming to be more appropriate for modern applications. MRPT offers a number of SLAM algorithms which can be used in both 2D and 3D environments, and as the name suggests, is particularly useful for mobile robots.
Enki, Gazebo and Webots are all robot simulator packages, which contain a number of premade robots and environments. They are particularly useful for testing SLAM algorithms, as it allows the user to create custom environments and test the algorithms in a controlled and realistic setting. Enki is written in C++ and is open-source, which makes it easy to modify and extend. Gazebo can be used to create 3D environments, which can be used to test visual SLAM algorithms. Webots also allows for modelling, programming and simulating robots in 3D environments; the package is widely used in research and education. All of these packages are typically used in conjunction with packages that implement SLAM algorithms, like ROS.

2.5 Real-world Applications and Case Studies

SLAM has a number of real-world applications, such as autonomous vehicles, drones, vacuum cleaners and search and rescue. In the context of drones, UAV (Unmanned Aerial Vehicles) SLAM is used to help drones navigate their environment. This is particularly challenging as drones are often used in dynamic environments. Furthermore, drones often operate at fast speeds, which means the SLAM algorithm needs to be able to process data quickly. UAV SLAM can be implemented using similar techniques to ground-based SLAM, using LiDAR and cameras for feature detection; Caballero et al demonstrated how visual SLAM can be used in this context (16).
Another application of SLAM is in Virtual Reality (VR) and Augmented Reality (AR) technologies. SLAM is used in VR and AR to help localise the headset in the environment, which is important for creating realistic and consistent movement. Kuo et al demonstrated this well in their paper, where they used ORB SLAM 2 to localise a VR headset in a 3D (15). Their implementation involved using a VR headset to control a robot; they were able to demonstrate that the error rate of the SLAM algorithm was at least 0.5cm, which is accurate enough for most VR applications.

3 Requirements Analysis

Table 1 shows the requirements for my project, along with their justification. Initially, I will create a simulation interface, where the user can see the agents and the environment. After, I will work towards implementing core parts of the SLAM algorithm, such as feature extraction and landmark detection. I will then attempt to implement a Kalman Filter, which the agent will use to localise itself in the environment. Finally, I will work towards adding data association, which will hopefully complete the SLAM algorithm. The latter stages of the project will be spent on evaluating the performance of the algorithm and documenting my findings. I have also added a couple optional requirements, which can be carried out should time permit but are not critical towards the success of the project.
When creating the simulation interface, I will be using the Python programming language, along with the PyGame library. This will abstract away a lot of the complexity of creating a graphical user interface, allowing me to focus on the core functionality. I have chosen to use Python in my project as it is a language I am familiar with and is widely documented, which will be useful should I run into issues. I will also use various other scientific python libraries throughout my project, such as NumPy, SciPy and Matplotlib.
I have chosen to create my own simulation environment, instead of using an existing package, as I believe this will give me more control over the project. I enjoy knowing how each component of my project works, and by creating my own environment, I will be able to create a more tailored solution.

ID	Requirement Type	Requirement	Justification
1	Funcitonal	Simulation Interface	A graphical user interface will be critical for demonstrating the agents behaviours, and will also aid de-bugging. Pygame will make it easy to watch the algorithm in real-time and visualise the mapping process.
2	Funcitonal	Implement the Extended Kalman Filter (EFK) algorithm	EFK will be essential for estimating the state of the agent and the uncertainty related to it's position, and the position of other agents.
3	Funcitonal	Simulation Environment Creation	In order for SLAM to work, a simulated environment will be needed. This will help me demonstrate the SLAM algorithm and test its performance.
4	Non-Funcitonal	Program Performance	The program will need to be efficient and the UI should be smooth. This is necessary to help provide a realistic demonstration of the SLAM algorithm.
5	Non-Funcitonal	Modularity	The program should be modular, allowing for easy changes to both the agents behaviour and the environment. This is important as it will allow me to easily test different scenarios and variations of the SLAM algorithm.
6	Non-Funcitonal	Documentation and Code Comments	The code should be well documented and commented, as this will help make the codebase more maintainable and easier to understand.

Table 1: Table of requirements and their justification

Along with this table of requirements, there will also be a number of opportunities for optional extensions, should time permit. These include:

- Implementing physical robots.
- User interface enhancements, such as adding being able to view each individual agents internal map.
- Implementing a more complex SLAM algorithm, such as FastSLAM.

4 Project Plan

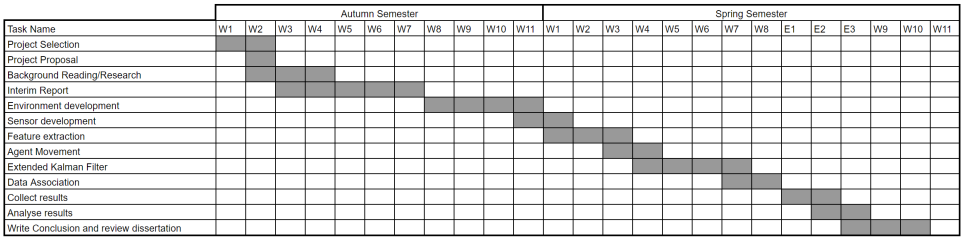

Figure 7: Gantt chart showing the project plan

The execution of my project will be split into various phases, where each phase will focus on an area of development. In order to remain focused, I will be using an agile-like approach, where I will be working in short sprints to complete certain tasks, then writing and reflecting on them in my report. Figure 7 shows the project plan, where the grey bars represent the time spent at each phase. Should my project overrun, I will have contingency time built into both the Christmas break and prior to the due date, which is currently unaccounted for in the project plan. I have also tried to be realistic with my time estimations, allowing for a reasonable amount of time to complete each phase.

4.1 Phase 1 - Research and Planning

The first phase of my project involves researching and planning. During this period I will create a project proposal, research single and multi-agent SLAM algorithms, and write my interim report. This phase will be completed by week 7. It is important to carry out this phase as it provides structure for the whole project, which will help ensure that the project is completed on time.

4.2 Phase 2 - Project Setup

The second phase of my project will be to develop the simulation environment and sensors for the agent. This will involve creating a graphical user interface, using the PyGame library, where the user can see the agent, the environment and a representation of the agents internal map. I will then create a sensor which can detect the presence of walls and objects in the environment. I plan on implementing a LiDAR sensor, as this will provide my feature detection system with the information it should require. This phase will be completed by week 11 - putting me in a good place to work on implementing SLAM algorithm after the Christmas break.

4.3 Phase 3 - Data Preprocessing

The third phase of my project will be to establish the fundamenals of the SLAM algorithm. I will start by implementing a feature extraction system, which will be used to draw lines and edges through points detected by the LiDAR sensor. I will then work on identifying landmarks, which will be used to create a map of the environment. After

implementing these, I will spend some time writing up my methods in my report, ensuring that I have a clear understanding of the algorithms I have implemented. This phase should be completed by week 4 of the Spring semester.

4.4 Phase 4 - Algorithm Implementation

The fourth phase of my project will be to implement the core parts of the SLAM algorithm. I will start by implementing a Extended Kalman Filter, which will be used to estimate the agents position and the position of landmarks in the environment. I will then implement a data association algorithm, which will be used to compare new features to existing landmarks. After completing this work, I should have fully implemented the SLAM algorithm. I expect this stage of the implementation to take the longest, so I have allocated a significant amount of time to complete it. This phase should be completed by week 8 of the Spring semester.

4.5 Phase 5 - Analysis and Conclusion

I will start the final phase of my project during the Easter break, where I will collect data which will be used to analyse the performance of the algorithm. I will then use this data to create a series of graphs and charts, which will be used to visualise performance and draw conclusions. After, a lot of time will be spent writing up my findings and analysing the performance of the algorithms. Finally, I will finish my project by writing a conclusion, which will summarise my findings and discuss potential future work. Prior to submitting my report, I will also spend time proofreading and editing my work.

5 Professional and Ethical Considerations

My project maintains compliance towards all ethical considerations, as there is minimal external involvement from humans. The majority of my project will be carried out in simulation, therefore no ethical approval is required. Should my project progress to physically implementing agents, considerations such as safety around the robots, will be considered. All tests will be carried out in an environment where people cannot be hit, therefore mitigating any trip hazards.

To ensure all elements of the BCS code of conduct are met, I have summarised each section and how I meet certain criteria.

5.1 Public Interest

As mentioned previously, my project has due regard for public health, as there is minimal external involvement from humans. Furthermore, no major privacy, security or wellbeing considerations are required due to the nature of this project. Third parties will be respected throughout the project with consistent citations and there will be no discrimination against anybody involved. Finally, I will promote equal access to the benefits of IT by open-sourcing my research once I have graduated. This will be accessible on my Github profile: https://github.com/CharlieAnthony/

5.2 Professional Competence and Integrity

My project is within my professional competence, as it significantly relies upon knowledge obtained from modules such as "Acquired Intelligence and Adaptive Behaviour" and "Fundamentals of Machine Learning." Furthermore, I will develop my professional knowledge, skills and competence through communicating with my supervisor and ensuring all relevant gaps in knowledge are explored through reading extensively. As part of my background reading, I have made myself familiar with the BCS code of conduct and surrounding legistlation. I will comply with this throughout when carrying out my professional responsibilities. There will also be no unethical inducements offered or accepted throughout the project.

5.3 Duty to Relevant Authority

As the relevant authority will be the University of Sussex, I will comply with all relevant codes of conduct and legislation. I will exercise my professional judgement at all times, including avoidance of any situation that may give rise to a conflict of interest between myself and the university. I will also make it my responsibility to ensure all colleagues work is properly referenced in a bibliography at the end of my dissertation.

5.4 Duty to the Profession

Finally, I accept my duty to uphold the reputation of the profession. I will work to the best of my ability to ensure my project is complete to the highest possible standard. As mentioned previously, I will seek to improve professional standards through communication with my supervisor. This dissertation will be written with integrity and respect towards all members of BCS and colleagues of the profession.

6 Methods

6.1 Environment Development

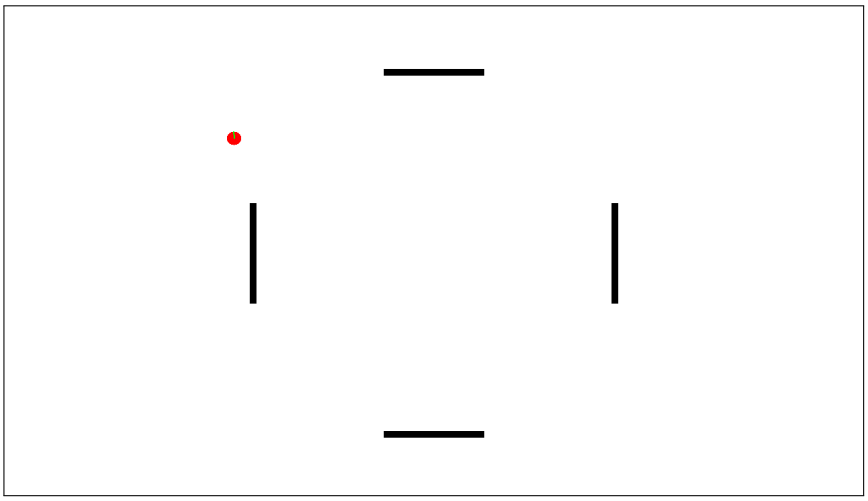

Figure 8: Screenshot of the simulation environment

When constructing the graphical user interface (GUI), simplicity has always been the focus. To maintain this, the GUI clearly displays the agent (and its bearing) and it's environment in an intuitive manner; as seen in Figure 8. Upon initialisation, the environment accepts an image file,which it then uses to create an array representation of the environment. I chose to use images as input as it allows for quick and easy creation of environments, which can be easily modified. Arrays are used to represent the environment as they are efficient and easy to manipulate.

6.2 Sensor Development

A realistic sensor is crucial for the agent to be able to detect its environment. To achieve this, I have implemented a LiDAR sensor, which in a similar way to real-world LiDAR sensors; it rotates 360 degrees and takes measurements at regular intervals. My sensor implementation mimics real-world LiDAR sensors by expanding a ray from the agent into the environment, which then returns a distance measurement if it hits an object. Below is the pseudocode for the sensor implementation.

Algorithm 1 LiDAR Sensor Algorithm

```
function LIDAR_SCAN(environment, agent_position, num_rays, max_distance)
    data ← []
    for i ← 0 to num_rays do
        angle ← i * 360/num_rays
        for j ← 0 to max_distance do
            new_x ← agent_position.x + j * cos(angle)
            new_y ← agent_position.y + j * sin(angle)
            if environment.is_object(new_x,new_y) then
                data.append((angle , j/max_distance))
                break
            end if
        end for
    end for
    return data
end function
```

The procedure *lidar_scan* takes four parameters: the environment, the agent's position, the number of rays to cast and the maximum distance (range) of the sensor. The algorithm will execute in $O(d * r)$ time, where d is the maximum distance and r is the number of rays. The algorithm's memory complexity is $O(r)$, as it is bounded by the number of rays.

6.3 Feature Extraction

Feature extraction is the process of detecting and extracting features from sensor data. In the context of SLAM, features are points of interest in the environment, such as corners, edges and lines. These features can then be used to create a map of the environment. There are many different techniques for feature extraction, such as the Harris corner detector and the split-and-merge algorithm. My implementation uses Seeded region growing, which was proposed by Gao et al. (17).
Seed segment detection works by observing a set of points from a single sweep. The algorithm starts by fitting a line through the given points. Then, in order for a seed segment to be further considered, it must satisfy the following two conditions:

- The distance between the point and the line must be less than a given threshold.
- The distance between the point and it's predicted position must be less than a given threshold.

Seed segment detection uses orthogonal line fitting to propose a feature through a set of points, as it is more effective than using traditional methods, such as standard least square fitting; this is due to the nature of least square fitting which only takes into account the vertical distances of each point.
After, the algorithm applies region growing, which helps create line segments; which will then become our features. The region growing works by examining neighbouring points and adding them to the line segment if they satisfy the same conditions. This process is

repeated until no more points can be added to the line segment.

Algorithm 2 Seed-Segment Detection Algorithm (reproduced from (17))

```
function SEED_SEGMENT_DETECTION(N_p, ε, δ, S_num, P_min)
    flag ← true
    for i ← 1 to (N_p - P_min) do
        j ← i+ S_num
        fit_line(i, j)
        for k ← i to j do
            predicted_point ← predict_point(k)
            distance ← euclidean_distance(k, predicted_point)
            if d_1 > δ then
                flag ← false
                break
            end if
            distance ← euclidean_distance(k, line)
            if distance > ε then
                flag ← false
                break
            end if
        end for
        if flag == true then
            return fit_line(i, j)
        end if
    end for
end function
```

Algorithm 2 is the pseudocode for the seed segment detection algorithm. The function *seed_segment_detection* takes five parameters: the number of points, the thresholds ϵ and δ, the number of points in a segment S_{num} and the minimum number of points P_{min} required to form a segment. The algorithm works by iterating over groups of points and attempting to fit a line through them. It then checks if the points conform to the line by calculating the distance between the point and it's predicted points position, ensuring it is less than δ. If this condition is met, algorithm then checks the distance between the point and the line, ensuring it is less than ϵ. If both conditions are met, the algorithm returns the line segment. If no line segment is found, the algorithm will return null. The algorithm will execute in $O(n)$ time, where n is the number of points. The algorithm's memory complexity is $O(1)$, as it is bounded by the number of points.

Algorithm 3 Seed-Segment Growing Algorithm (reproduced from (17))

function SEED_SEGMENT_GROWING($line(i, j)$, N_p, P_{min}, L_{min}, ϵ)
  $P_f \leftarrow j + 1$
  $P_b \leftarrow i - 1$
  while euclidean_distance(P_f, $line(i, j)$) $< \epsilon$ **do**
    if $P_f > N_p$ **then**
      break
    else
      $line \leftarrow fit\_line(P_b, P_f)$
    end if
    $P_f \leftarrow P_f + 1$
  end while
  $P_f \leftarrow P_f - 1$
  while euclidean_distance(P_b, $line(i, j)$) $< \epsilon$ **do**
    if $P_b < 1$ **then**
      break
    else
      $line \leftarrow fit\_line(P_b, P_f)$
    end if
    $P_b \leftarrow P_b - 1$
  end while
  $P_b \leftarrow P_b + 1$
  $L_l \leftarrow euclidean\_distance(P_b, P_f)$
  $P_l \leftarrow P_f - P_b$
  if $L_l \geq L_{min}$ and $P_l \geq P_{min}$ **then**
    return $line(P_b, P_f)$ with **Parameters**
  end if
end function

The seed segment (line) produced by Algorithm 2 is then passed into Algorithm 3, which is used to grow the line segment. The function *seed_segment_growing* takes five parameters: the seed segment, the number of points, the minimum number of points required to form a segment, the minimum length of a segment and the threshold ϵ. Generally speaking, the algorithm expands the line segment by adding points to either side of the existing line segment, essentially 'growing' the line. Seed segment growing starts by attempting to expand the line forwards, by adding points to the end of the line and fitting a new line through the points. It then checks if the new line is within the threshold ϵ of the original line by calculating the distance between the points and the line. If it is, it will continue to expand the line. It repeats this until either all the points have been used or the threshold is exceeded. After, the same series of steps is repeated in the opposite direction, attempting to grow the region backwards. Finally, the algorithm checks if the grown line segment is of sufficient length and contains enough points to be considered a feature, then returns the line segment. The algorithm will execute in $O(n)$ time, where n is the number of points. The algorithm's memory complexity is $O(1)$, as it is bounded by the number of points.

6.4 Agent Movement

When implementing the agent's movement, I wanted it to accurately mimic a real-world agent moving in an environment. To achieve this, I have implemented a basic kinematic model, which uses the agent's current position and velocity to calculate the next position. For each time step, the agent passes the model it's control inputs, consisting of the agent's velocity and angular rate of change. The model then calculates the agent's new position and orientation. The model is important as it allows the agent to move around the environment in a predictable manner, which will help make the implementation of my project easier. Below is the equations used in my kinematic model.

$$\begin{aligned} p_x &= v\cos(\theta) \\ p_y &= v\sin(\theta) \\ \theta &= \omega \end{aligned} \tag{7}$$

In the equations above, p_x and p_y are the agent's x and y positions, θ is the agent's bearing. v is the agent's velocity and ω is the agent's angular rate of change. In my implementation, I have capped the agent's velocity and angular rate of change to ensure the agent moves in a realistic manner.

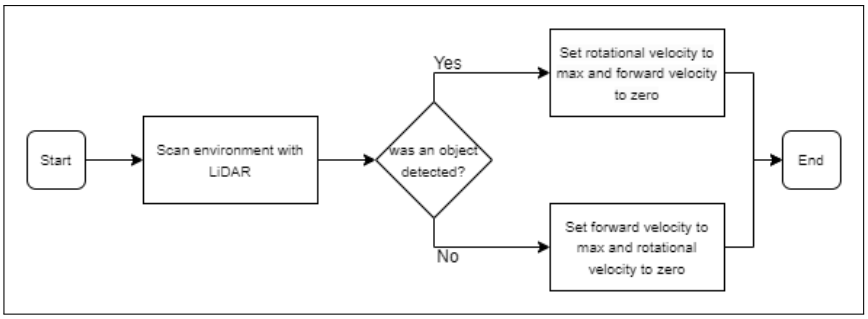

Figure 9: Flowchart of the wall tracking exploration strategy

To be able to create a map of the environment, the agent must first explore the environment. One of the simplest exploration strategies is wall tracking, where the agent moves in a single direction until it hits an obstacle, then it until no longer facing the obstacle and continues. This is a common strategy used in SLAM as it is easy to implement and computationally efficient; however it is by no means the best strategy for exploration. Figure 9 is a simple flowchart of my implementation of the wall tracking.

6.5 Extended Kalman Filter

As mentioned in the literature review, the Extended Kalman Filter (EKF) is a popular choice for SLAM algorithms which builds upon the original Kalman Filter equations. The EKF works by estimating the state of the agent and the landmarks in the environment by using a combination of sensor measurements and control inputs. The EKF is particularly useful in SLAM as it can handle non-linearities in the system, which are common in real-world environments. In this context, linearity is referring to directly proportional variables, such as the agent driving in a straight line at a constant speed.
Algorithm **??** shows the EKF algorithm, which is used to estimate the state of the agent

and the landmarks in the environment. This algorithm is taken from the Probabilistic Robotics textbook by Thrun et al. (18) - which is a widely used resource in the field of robotics. I have adapted the algorithm to work with my implementation of SLAM.

Algorithm 4 EKF SLAM Algorithm (reproduced from (18)

1: $\mathbf{F}_x = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}$

2: $\bar{\mu}_t = \mu_{t-1} + \mathbf{F}_x^\top \begin{bmatrix} -\frac{v_t}{w_t}\sin(\mu_{t-1,\theta}) + \frac{v_t}{w_t}\sin(\mu_{t-1,\theta} + w_t\Delta t) \\ \frac{v_t}{w_t}\cos(\mu_{t-1,\theta}) - \frac{v_t}{w_t}\cos(\mu_{t-1,\theta} + w_t\Delta t) \\ w_t\Delta t \end{bmatrix}$

3: $\mathbf{G}_t = I + \mathbf{F}_x^\top \begin{bmatrix} 0 & 0 & \frac{v_t}{w_t}\cos(\mu_{t-1,\theta}) - \frac{v_t}{w_t}\cos(\mu_{t-1,\theta} + w_t\Delta t) \\ 0 & 0 & \frac{v_t}{w_t}\sin(\mu_{t-1,\theta}) - \frac{v_t}{w_t}\sin(\mu_{t-1,\theta} + w_t\Delta t) \\ 0 & 0 & 0 \end{bmatrix} F_x$

4: $\bar{\Sigma_t} = \mathbf{G}_t\Sigma_{t-1}\mathbf{G}_t^\top + \mathbf{F}_x^\top \mathbf{R}_t \mathbf{F}_x$

5: $\mathbf{Q}_t = \begin{bmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{bmatrix}$

6: **for all** observed features $z_t^i = (r_t^i \phi_t^i s_t^i)^\top$ **do**

7: $j = c_t^i$

8: **if** landmark j never seen before **then**

9: $\begin{bmatrix} \mu_{j,x} \\ \mu_{j,y} \\ \mu_{j,s} \end{bmatrix} = \begin{bmatrix} \bar{\mu_{t,x}} \\ \bar{\mu_{t,y}} \\ s_t^i \end{bmatrix} + r_t^i \begin{bmatrix} \cos(\phi_t^i) + \bar{\mu_{t,\theta}}) \\ \sin(\phi_t^i + \bar{\mu_{t,\theta}}) \\ 0 \end{bmatrix}$

10: **end if**

11: $\delta = \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} = \begin{bmatrix} \bar{\mu_{j,x}} - \bar{\mu_{t,x}} \\ \bar{\mu_{j,y}} - \bar{\mu_{t,y}} \end{bmatrix}$

12: $q = \delta^\top \delta$

13: $\hat{z_t^i} = \begin{bmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \bar{\mu_{t,\theta}} \\ \bar{\mu_{j,s}} \end{bmatrix}$

14: $\mathbf{F}_{x,j} = \begin{bmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ & & & & & & & \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 1 & 0 \cdots 0 \end{bmatrix}$

15: $\mathbf{H}_t^i = \frac{1}{q} \begin{bmatrix} \sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & -\sqrt{q}\delta_x & \sqrt{q}\delta_y & 0 \\ \delta_y & \delta_x & -1 & -\delta_y & -\delta_x & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{F}_{x,j}$

16: $K_t^i = \bar{\Sigma_t}\, {\mathbf{H}_t^i}^\top \left(H_t^i \bar{\Sigma_t}\, {\mathbf{H}_t^i}^\top + Q_t \right)^{-1}$

17: **end for**

18: $\mu_t = \bar{\mu}_t + \Sigma_i K_t^i (z_t^i - \hat{z}_t^i)$

19: $\Sigma_t = (I - \Sigma_i K_t^i \mathbf{H}_t^i)\, \bar{\Sigma_t}$

20: **return** μ_t, Σ_t

6.6 Data Association

I decided to create a simple algorithm for data association, which works by comparing the endpoints of detected features with known landmarks. The algorithm calculates the distances between each endpoint of the detected feature and each landmark's set of endpoints, then determines whether the feature is a new landmark or an existing one using a pre-determined threshold. The algorithm is simple but effective; it's computational efficiency is bounded by the number of landmarks, which makes it appropriate for my project.

7 Appendices

7.1 Supervisor Meetings

7.1.1 Meeting 1 - 11/10/2023

Discussed on the project idea and potential directions to take. Discussed the possibility of implementing physical agents, challenges that may occur and potential ways of implementing swarm algorithms. Need to focus on researching SLAM and swarm and looking into existing resources.

7.1.2 Meeting 2 - 27/10/2023

Discussed potential algorithms, such as particle filters and graph-based SLAM. We also discussed the logistics of the project, ensuring that it remains both realistic and achievable. We also discussed the possibility of implementing physical agents, and where relevant resources could be found.

7.1.3 Meeting 3 - 14/11/2023

Started with feedback on the interim report - discussing the structure and content. After we discussed how the project will move forward and the next steps to take. Given the interim report is now complete, we can focus mainly on development, following the project plan. As I have already developed a basic simulation interface, I can now move onto implementing my first SLAM algorithm.

7.1.4 Meeting 4 - 08/12/2023

In this meeting, we turned to ironing out the specifics of the implementation - including looking at existing resources, like Enki, and concepts that need to be considered, such as Differential Turning. The goal of this meeting was to guide me into starting to create my environment and first SLAM algorithm, which will be implemented over the christmas break.

7.1.5 Meeting 5 - 02/02/2024

We firstly caught up on progress made over the christmas break. After, we started to look forward to the next steps of the project, discussing the projects overall direction and the next steps to take. One notable suggestion was the move away from swarm algorithms and perhaps the move towards multi-agent SLAM, as this would be more achievable in

the time frame. Finally, we discussed how I should manage my time towards the end of the project and how I could start working on my dissertation.

7.1.6 Meeting 6 - 09/02/2024

Started with me demonstrating my current progress, with my environment working, LIDAR sensor appropriately implemented and my work-in-progress feature detection. We discussed then how I could approach landmark detection and how I planned on implementing it. We ended the meeting with clearing up questions regarding the project presentation, poster competition and submission.

7.1.7 Meeting 7 - 16/02/2024

We discussed how my project was going; talking about feature extraction and landmark detection. As I had been having troubles with bugs in the previous week, Chris suggested spending more time writing test cases. We then discussed how I should approach writing the final report; considering structure and content. We agreed that there are parts of the report that I could start now, such as my literature review and methodology used in feature detection and landmark detection.

7.1.8 Meeting 8 - 23/02/2024

Started with showing my progress on landmark detection and randon walk exploration. We then discussed different exploration algorithms that could be used, as random walk exploration isn't efficient. We discussed creating some form of wall-avoidance navigation, which would be far more efficient. We then reviewed my plan and reflected on how progress was going. We both agreed that progress isn't as fast as we would like, but we are still on track to complete the project on time.

7.1.9 Meeting 9 - 29/02/2024

We had a quick online meeting this week; discussing where the project is and the next steps. We started by talking about my implementations of exploration strategies and how I could improve them. We then discussed evaluation metrics and how I could gather and present data. Finally, we talked about how I could approach multi-agent SLAM, as it's an area I am concerned about due to it's complexity.

7.1.10 Meeting 10 - 15/03/2024

This meeting was a clear turning point in the project. As progress hadn't proceeded as expected, we discussed the scope of the project and how it could be adjusted to ensure that I could complete it on time. We agreed that I should focus on the single-agent components of SLAM, as they are the most important parts of my project. We also discussed how I could approach the final report - including changes that would need to be made to adhere to the new project scope. Finally, we discussed in further detail how I could gather data for my analysis and how I could present it.

7.1.11 Meeting 11 - 22/03/2024

As the final decisions had been made, a lot of the previous week was spent working on my paper; therefore there wasn't much to show. We met on zoom and discussed the rate of progress, how the report is coming along and suggestions about my referencing. We also discussed the presentation some more, as it has been an area troubling me. Chris talked me through the process - stating that it should just be a summary of my project, really focusing on the key points and results. This was the last meeting before the easter break, so progress should increase after this point.

7.1.12 Meeting 12 - 03/04/2024

We met during the easter break on zoom to catch up about on progress. Chris provided some general feedback on the earlier sections of my paper, suggesting how I could adjust it's content to create a more balanced report. We then talked about how the code was coming along; I talked through some of the troubles I've had with the EKF implementation and Chris made some suggestions about how I could approach it differently. We finished by talking through my plan for the final weeks of the project - I discussed a rough plan I had constructed and Chris confirmed that it seemed appropriate.

7.1.13 Meeting 13 - 12/04/2024

We met in person this week, as we both happened to be on campus. I showed Chris my progress on my EKF implementation and we discussed final changes I could make to the code. After we talked about the final report, and how I could send Chris my draft in parts for a more structured feedback process. After, we got slightly side-tracked - discussing life after uni and potential career paths, concluding the meeting.

7.2 Project Proposal

Swarm Robotics: Exploration and Mapping in Simulated testing Environments

Charlie Anthony [candNo: 246537]
Supervisor: Dr Chris Johnson

Project Proposal
Computer Science and Artificial Intelligence BSc

Department of Informatics and Engineering
University of Sussex
October 2023

Contents

1 Aims and Objectives

Aim:
To understand and showcase the principles of swarm robotics in the realm of navigation and mapping. This project is driven by a fascination with swarm robotics and its potential in autonomously navigating and mapping unknown environments.

Primary Objectives:

- Design and develop a basic simulation environment representing an unknown environment.
- Implement swarm intelligence principles to allow a group of agents to collaboratively navigate and map the environment.
- Evaluate and refine the agent behaviours for effective navigation and territory mapping.

Extensions (if time allows):

- Optimise agent behaviour for efficiency in discovering the quickest route to a goal point within a maze-like environment
- Investigate the challenges associated with transitioning from simulation to real-world application (the sim-to-real gap)

2 Relevance

This project integrates principles of artificial intelligence, robotics and simulation, making it highly relevant to my degree in Computer Science and Artificial Intelligence. The exploration of swarm robotics in navigation can provide insights into optimizing algorithms for real-world challenges.

3 Resources Required

This project will require the use of lab computers, and should the extensions be carried out, the occasional booking of seminar rooms/study rooms for carrying out physical experiments. Should it be required, the project will also be aided by a small degree of funding to allow purchase of physical components which may be required for constructing agents. While the purchase of such components may not be essential, it would allow a more in-depth and thorough review of the swarm behaviours implemented.

4 Timetable

Here is a simplified version of my timetable:

	Mon	Tue	Wed	Thur	Fri
9:00		project/cw	Lecture		project/cw
10:00		project/cw	Lecture	Lecture	project/cw
11:00		project/cw	Lecture	Lecture	project/cw
12:00					
13:00		Lecture			Lab
14:00		Lecture			Lab
15:00		Lab			
16:00					
17:00					

8 References

References

[1] Frese, U., Wagner, R. and Röfer, T. (2010). A SLAM Overview from a User's Perspective. *KI - Künstliche Intelligenz*, 24(3), pp.191–198. http://dx.doi.org/10.1007/s13218-010-0040-4

[2] Alsadik, B. and Karam, S. (2021) The Simultaneous Localization and Mapping (SLAM)-An Overview. *Journal of Applied Science and Technology Trends, 2(02), pp. 147 - 158.* https://doi.org/10.38094/jastt204117

[3] Smith, R.C.; Cheeseman, P. (1986). "On the Representation and Estimation of Spatial Uncertainty". *The International Journal of Robotics Research. 5 (4): 56–68.* https://doi.org/10.1177/027836498600500404

[4] J. J. Leonard and H. F. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," *Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91, Osaka, Japan, 1991, pp. 1442-1447 vol.3,* https://doi.org/10.1109/IROS.1991.174711

[5] Montemerlo, M., Thrun, S., Koller, D. and Wegbreit, B., 2002. "FastSLAM: A factored solution to the simultaneous localization and mapping problem," *Proceedings of the AAAI National Conference on Artificial Intelligence. pp. 593-598.*

[6] Durrant-Whyte, H. and Bailey, T. (2006). "Simultaneous localization and mapping: part I", *IEEE Robotics Automation Magazine, [online] 13(2), pp.99–110.* https://doi.org/10.1109/mra.2006.1638022

[7] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): part II," *IEEE Robotics Automation Magazine, vol. 13, no. 3, pp. 108-117, Sept. 2006.* https://doi.org/10.1109/MRA.2006.1678144

[8] Nam, Tae Shim, Jae Cho, Young. (2017). "A 2.5D Map-Based Mobile Robot Localization via Cooperation of Aerial and Ground Robots", *Sensors 17(12): 2730.* http://dx.doi.org/10.3390/s17122730

[9] Y. Wu, Y. Zhang, D. Zhu, Y. Feng, S. Coleman and D. Kerr, (2020). "EAO-SLAM: Monocular Semi-Dense Object SLAM Based on Ensemble Data Association," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4966-4973*, https://doi.org/10.1109/IROS45743.2020.9341757

[10] Welch, G. and Bishop, G. (1995). An Introduction to the Kalman Filter. *online* Available at: https://perso.crans.org/club-krobot/doc/kalman.pdf (Accessed: 13 February 2024)

[11] Kim, S. (2023). "FastSLAM: Algorithm for Simultaneous Localization and Mapping," *[online] MLPurdue.* Available at: https://medium.com/ml-purdue/fastslam-algorithm-for-simultaneous-localization-and-mapping-6fb5bd141f30. (Accessed: 20 March 2024)

[12] F. Dellaert, D. Fox, W. Burgard and S. Thrun, (1999). "Monte Carlo localization for mobile robots," *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), pp. 1322-1328 vol.2,* https://doi.org/10.1109/ROBOT.1999.772544

[13] R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, (2015). "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147-1163,* https://doi.org/10.1109/TRO.2015.2463671

[14] W. Hess, D. Kohler, H. Rapp and D. Andor (2016). "Real-time loop closure in 2D LIDAR SLAM," *2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 1271-1278,* https://doi.org/10.1109/ICRA.2016.7487258

[15] Chen-Yu Kuo, Chun-Chi Huang, Chih-Hsuan Tsai, Yun-Shuo Shi, Shana Smith, (2021). "Development of an immersive SLAM-based VR system for teleoperation of a mobile manipulator in an unknown environment," *2021 Computers in Industry, Volume 132, 103502, ISSN 0166-3615,* https://doi.org/10.1016/j.compind.2021.103502

[16] Caballero, F., Merino, L., Ferruz, J. et al. "Vision-Based Odometry and SLAM for Medium and High Altitude Flying UAVs," *J Intell Robot Syst 54, 137–161 (2009),* https://doi.org/10.1007/s10846-008-9257-y

[17] Gao, H., Zhang, X., Fang, Y. and Yuan, J. (2018). "A line segment extraction algorithm using laser data based on seeded region growing," *International Journal of Advanced Robotic Systems*, p.172988141875524. https://doi.org/10.1177/1729881418755245

[18] Thrun, S., Burgard, W. and Fox, D. (2005). "Probabilistic Robotics". *MIT Press, Cambridge, MA, USA.*