Final Project I

Collective Transport using Decentralised Swarm Robotics

Submitted to the

Project Committee appointed by the
International School of Engineering (ISE)
Faculty of Engineering, Chulalongkorn University

Project Adviser

Asst.Prof.Paulo Fernando Rocha Garcia, Ph.D.

Submitted By

6438067021 Nattadon Tangsasom
6438075021 Ting-Yi Lin
6438079621 Tinapat Limsila
6438118421 Noppawan Srikhirin
6438187721 Mehul Sharma

1/2024: 2147416 Final Project I
Robotics and Artificial Intelligence Engineering (International Programme)
International School of Engineering (ISE) Faculty of Engineering, Chulalongkorn University

Contents

Abstract

The rapid advancements in robotics have created significant opportunities to address complex tasks through collaborative systems. This report explores the project **"Collective Transport using Decentralised Swarm Robotics"**. The core problem addressed is the difficulty of achieving effective collaboration between decentralised robots, particularly in tasks such as collectively transporting objects in unstructured environments. Unlike centralised systems, which are vulnerable to single-point failures, decentralised swarm systems offer robust, scalable, and efficient solutions.

This document provides an in-depth analysis of key components, including communication protocols, object detection, localisation, SLAM, and hardware integration. It also includes a comprehensive literature review and a solid theoretical framework underpinning the project's objectives. The report is organised into nine chapters, beginning with an introduction and a comprehensive literature review, followed by an overview of the simulation. Subsequent chapters delve into the primary aspects of the project: communication, object detection, localisation, SLAM, and robot formation.

This report summarises the progress achieved during this semester and provides a comprehensive overview of our work on the project.

Chapter 1

Introduction and Overview

The project, "Collective Transport using Decentralised Swarm Robotics," aims to pioneer a new approach to robotic collaboration by enabling decentralised control for tasks such as object transportation in unstructured environments. Traditional centralised systems, like those in RoboCup Soccer, have dominated the field for decades but face limitations such as single points of failure and reliance on powerful centralised servers. This project seeks to address these challenges by leveraging decentralised swarm robotics, which ensures robust, scalable, and efficient operations without dependency on a single control unit.

A core objective of the project is to develop a system where three robots can simultaneously map their environment, communicate with one another, and collaborate to transport an object from point A to point B in a synchronised manner. This involves the robots autonomously detecting an object, localising themselves in the environment, and coordinating their movements to ensure seamless and efficient transport. This goal demonstrates the potential of decentralised swarm robotics for highly coordinated tasks that require precise communication and execution. See Figure 1.2 for an simulation of the project's goal using the simulation. See Figure 1.1 for an illustration of the project's goal using a drawing

In real-world applications, this approach holds transformative potential, particularly in warehouse logistics and search-and-rescue operations. The decentralised model offers resilience and adaptability, making it suitable for scenarios where centralised systems may falter. Moreover, this technology aligns with Thailand's growing need for innovative solutions in robotics, showcasing a shift towards decentralised systems that can operate independently without a centralised power hub. This represents a vision for the future of robotics, where hardware and computation costs decrease, enabling accessible and scalable robotic systems.

The project development is divided into two phases: 1. Phase 1: Exploration of foundational modules, including communication protocols, object detection via computer vision, and localisation. While the original plan included SLAM, it was deprioritised due to its limited necessity in the simulation phase. However, the team actively studied SLAM in preparation for its integration in Phase 2. A significant focus was placed on ensuring the robustness of these individual modules, validated through simulations in Webots, which provided a cost-efficient, risk-free environment. 2. Phase 2: Integration of these foundational modules into a working prototype of robots navigating their environment. This phase also incorporates SLAM, advanced robot formations, and initial hardware configurations. The ultimate goal is to transition from

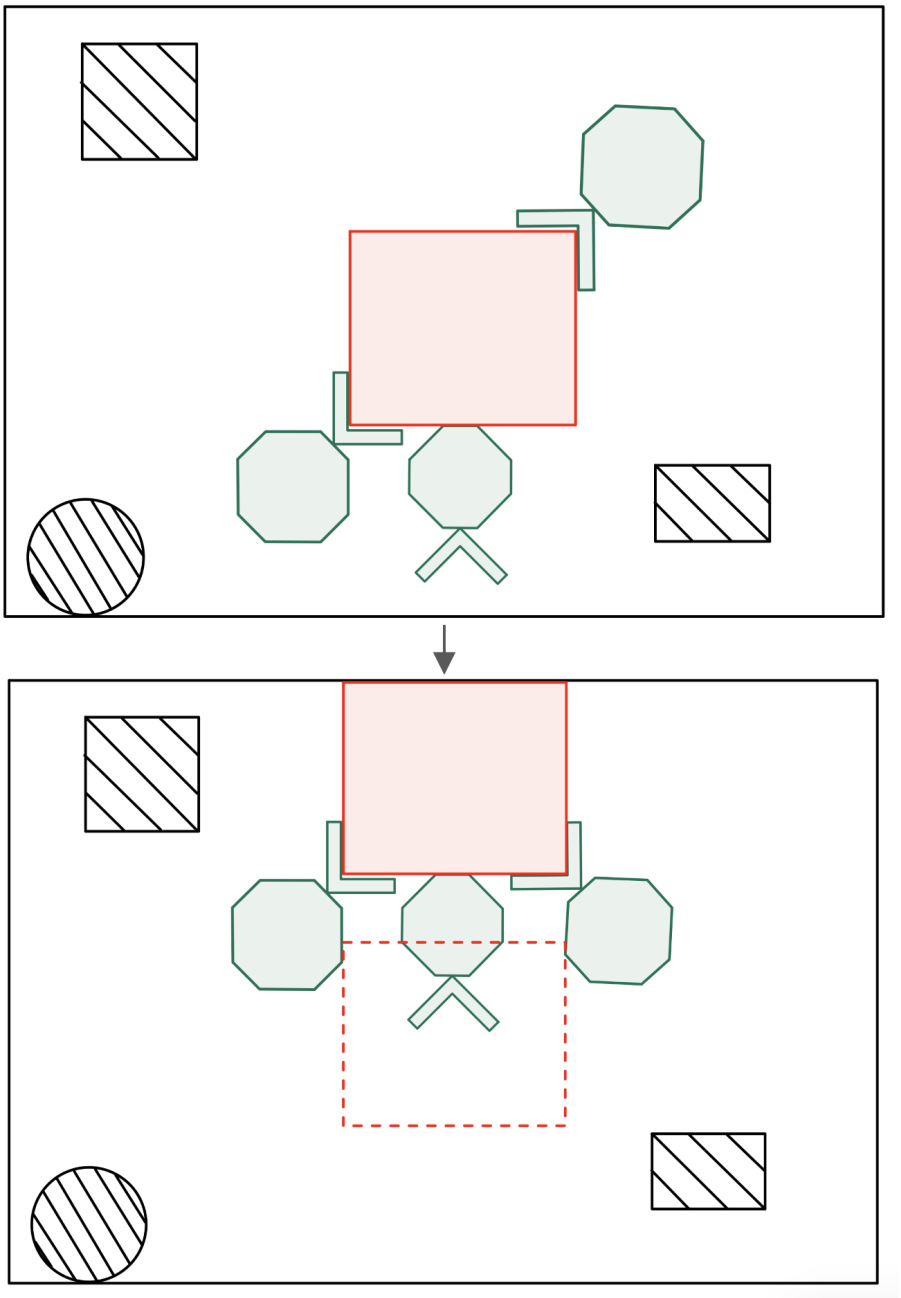

Figure 1.1: Illustration of the project's goal: three robots transporting an object using a drawing

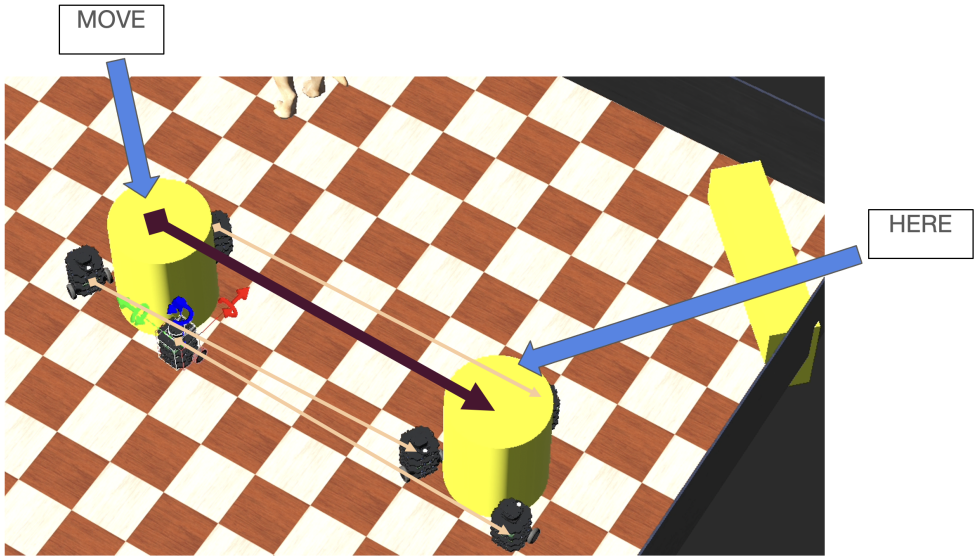

Figure 1.2: Illustration of the project's goal: three robots transporting an object using simulation

simulation to real-world applications, with the simulation results guiding the hardware design.

The success of the project is measured not only by the accuracy and control of individual robots but also by the robustness of their communication and coordination. Metrics include the ability to resolve edge conditions, such as simultaneous object detection or potential collision paths, and the effectiveness of decentralised decision-making. Overhead cameras verify the robots' positional accuracy, ensuring that simulated and real-world performance align

This report highlights the project's accomplishments, including completed tasks like communication, object detection, and localisation, alongside the progress of module integration. It also underscores the importance of simulation as a foundation for hardware development and presents a roadmap for achieving decentralised, collaborative transport systems. Through this effort, the project aims to establish a framework that not only advances robotics technology but also addresses challenges specific to Thailand, paving the way for future innovations in swarm robotics.

This midpoint report aims to highlight the progress of our project throughout the whole semester, with the evaluating criteria being the team's contributions alongside the pace in comparison to the ideal schedule. The ideal project timeline is presented with the Gantt Chart below. (Figure 1.3)

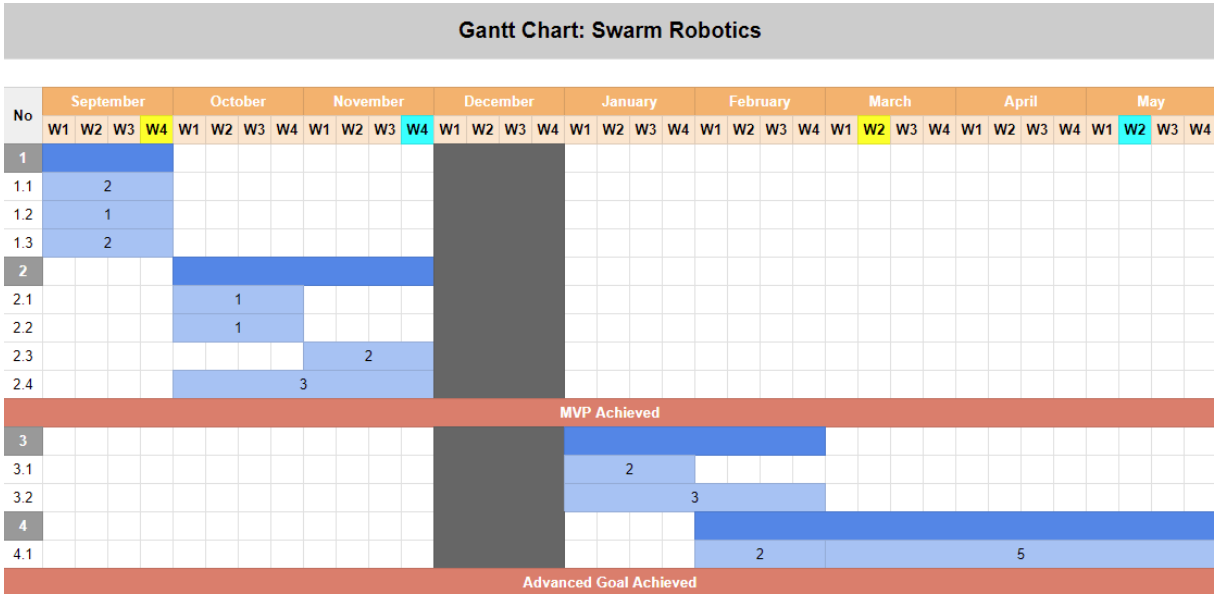

Figure 1.3: Project Gantt Chart

Current Gantt Chart and Progress:

Phase 1: Simulation and Fundamental Modules

1.1. Communication – *Completed* Achieved seamless coordination between robots using an infrared-based communication system, with a consensus algorithm to resolve conflicts during simultaneous detections or task assignments.

1.2. Object Detection using Computer Vision – *Completed* Developed a 2D object detection system with OpenCV for identifying the target (a yellow cylinder), aligning bounding boxes, and calculating distances accurately.

1.3. Localisation – *Completed* Implemented wheel odometry with encoders and differential drive kinematics to update robot positions reliably.

2.1. Simple Robot Formation – *Completed* Enabled basic formation control, allowing robots to form equilateral shapes and navigate in coordination.

2.4. Combination of Individual Modules in Simulation – *Completed* Integrated communication, object detection, and localisation modules into a working prototype in the simulation.

2.2. Simultaneous Localisation and Mapping (SLAM) – *In progress* Actively working on GraphSLAM, focusing on creating initial graphs, loop closure, and error function refinement to improve mapping accuracy.

Phase 2: Transition to Hardware and Advanced Features

2.3. Hardware – *In progress* Transitioning simulation modules to hardware, including: - Jetson installation with optimised configurations. - Testing closed-loop control for BLDC motors. - Preparing hardware communication systems for decentralised operations.

3.1. Coordinated Gripping – *To do* Develop an end-effector for gripping and securing objects during collaborative transport.

3.2. Advanced Robot Formation – *To do* Implement dynamic formation control to adapt to environmental changes and enhance navigation.

4.1. Hardware – *To do* Transition the full system to physical robots, focusing on advanced motor control, sensor calibration, and hardware-based object detection.

Chapter 2

Literature Survey and Review

2.1 Coordination

Swarm robotics solution is a design architecture that obtains inspirations from biological interactions; thus, they should operate autonomously to solve problems rather than relying on a central authority[1]. This decentralised approach is also crucial to achieve increased resilience and flexibility given a complex environment of deployment[2].

Communication protocols can be categorised into two principal types depending on the nature of information transmission: Direct communication and Indirect communication[2]. Direct communication refers to robotic agents that are able to coordinate via networked communication. Direct communication use cases are highlighted in many studies.

Ibrahim et al. [3] studied direct communication to determine the optimal distance for robotic agents to achieve consensus. Three strategies were tested within a 50 cm range: Close-neighbour, Far-neighbour, and Rand-neighbour. Close-neighbour excels in stable environments but performs poorly in complex ones. Both Close-neighbour and Far-neighbour introduce bias, reducing accuracy. Rand-neighbour, which randomly selects swarm members for communication, proved superior due to efficient information flow and minimal bias. This strategy can be one of our key designs for swarm communication in this project.

Ayari and Bouamama[4] and Perera et al.[5], S et al.[6] and Z. Wang et al.[7] conducted studies to promote robots' actions based on their sensory observations and objectives. S et al.[6] proposed a Multi-Agent Deep Deterministic Policy Gradient (MADDPG), an observation-based decision making flow with an architectural twist, where there is a central swarm manager that evaluates decentralized agents' reinforcement learning for optimal rewards. Z. Wang et al.[7] proposed increasing sensory inputs from both visual data and communication for redundancy, which highly aligns with the proposed swarm system.

Yasser et al.[8] expanded more on Clustered Dynamic Task Allocation (CDTA), an approach to dynamically assign tasks based on swarm state and the environment [9], for the purpose of increasing the velocity of swarm communication. They proposed

CDTA-CL (Centralised Loop) and CDTA-DL (Dual Loop). CDTA-CL sends information to the leader for computation, while CDTA-DL compares information before sending it to the leader. CDTA-DL outperformed CDTA-CL, increasing speed by 75.976% compared to 54.4%. CDTA-DL will be considered as a design pillar for this project.

In addition to direct communication improvements, indirect communication, known as Stigmergy, also plays a significant role in swarm intelligence. This concept involves individual robot actions modifying the environment, impacting the decision-making of other robots. For example, construction robots can leave blocks and materials to signal ongoing work[2]. This is an intriguing notion to consider in the interaction of the swarm system.

For effective communication, especially in interdependent tasks, robots need to be aware of each other and task requirements. Semantic communication, where contextually relevant data is prioritised, is necessary[10]. The balance between communication and context must align with task demands. High sensory data tasks require reliable, real-time communication, while tasks with lower communication demands can prioritise contextually relevant information[11].

2.2 Object Detection

Object Detection plays a crucial role in computer vision by identifying and locating objects within images or video feeds. Its primary goal is to identify objects as well as determining its precise location through bounding boxes. In order for the robots to interact with the environment and eventually complete their task, it is crucial that they have to understand the dynamic and complex surroundings with their sensors, followed by recognising all targets around them as well as targeting them. Another key requirement is that each robot should distinguish other team members from objects in the environment, known as "Kin detection", and know their relative position to others with data from their sensors

In various real-world applications, such as autonomous vehicles [12] and multi-robot coordination in warehousing [13], 2D object detection is effective for basic identification and localization tasks. However, detecting objects in complex environments presents challenges, particularly when object occlusion occurs [14]. The limitations of 2D object detection, which relies on single-plane images, make it difficult to accurately identify objects that are distant or situated in cluttered settings [15].

Three-dimensional data can be acquired using various devices such as RADAR, RGB-D cameras, and stereo cameras [16]. Each of these devices has distinct advantages and disadvantages:

RADAR employs radio signals to measure distance and estimate the shape and sise of objects but does not detect colour and has low spatial resolution [16]. This limitation makes it challenging to distinguish between closely spaced or thin objects [17]. However, RADAR is particularly effective in poor visibility conditions, such as for collision avoidance and adaptive cruise control.

RGB-D cameras, like the Intel RealSense D455, capture both colour and depth information, which allows for detailed spatial analysis [18]. These cameras use techniques such as time-of-flight and structured light to measure depth [18]. The data from RGB images and depth maps can be processed to create 3D point clouds or models, facilitating object detection and scene analysis. These cameras are adept at perceiving attributes like colour and shape in real-time, making them suitable for dynamic environments. Nonetheless, their performance can be impacted by lighting conditions, which affects depth accuracy, and they have a more limited sensing range compared to RADAR.

Stereo cameras operate with two lenses, capturing left and right images to mimic human vision [19]. The data collected typically consists of a 3D point cloud that shows the spatial distribution of objects in the environment. Creating this 3D point cloud involves processing stereo images to identify matching points and calculate their depth. While stereo cameras can detect distant and small objects due to their dense pseudo point cloud[20], they may struggle with object detection in dynamic environments because of noise in the point cloud data during rapid changes[21].

Considering stereo camera and RGB-D camera performance, as it can perceive colour which is the main characteristic that is required in object detection and classification, in 3D object detection. The RGB-D camera produces 0.82% average percentage error, while the stereo camera produces 2.61% average percentage error measuring under the same condition and doing the same tasks in multi-object detection[22].

Considering the data format obtained from the selected device which is the RGB-D camera will help to identify the suitable candidate of the 3D object detection approach. The input RGB-D data contain two formats, one with normal colour image which is red,green, and blue, another one uses black and white to represent the distance information in a single channel, The darker colour represents an object that is further from the camera than the ones with lighter colour.

Given the variety of tools and algorithms used in 3D object detection research with RGB-D cameras, a proper quantitative comparison has not been consistently conducted. Only a few approaches can be categorised into one of the four main groups of 3D object detection. Therefore, this section aims to provide an overview of the key techniques used. Each technique will include a discussion of its core concepts, along with its strengths and weaknesses. A quantitative comparison will be presented at the end of this section.

There are four major categories for 3D object detection using RGB-D cameras. Firstly, **2.5D processing methods**. This method utilises the 3D convolution kernel The concept of this method is to use depth images obtained from the RGB-D camera as 2D images directly. Additionally, HHA encoding is used for enhancing the geometric representation including object's height, surface angle compared to the ground etc.[23]. The combination of RGB and depth data in this approach enhances detection accuracy, especially in complex environments[23][24]. However, it can potentially perform not so well in highly dynamic or unstructured scenes especially when depth data is less accurate[23].

Secondly, **2D Driven 3D Methods** combine 2D object detection techniques with 3D processing to increase efficiency of 3D object detection. It basically narrows down the space with 2D detection followed by refining and defining 3D object boundaries using depth data. While RGB data remain unchanged, the depth data are transformed into point clouds[23]. A more advanced model called " The series of Frustum PointNet" which utilises RGB-D data for enhanced 3D object detection by integrating 2D image information (RGB data) with 3D point cloud data. 3D fustrums are transformed from 2D object proposals using camera projection metrics allowing effective feature extraction and object localisation using several advanced neural networks e.g. PointNet and LDG CNN. The integration of RGB and depth data boost the detection performance especially in small object detection like pedestrian as well as optimise computational resources while maintaining high detection rates[25]. However, the complexity of the system and processing time can be increased from relying on both point cloud and RGB data[26].

Geometric Descriptor-based Methods utilises depth information from RGB-D camera for enhancing object recognition capabilities by using geometric features extracted from RGB image and depth data. This category comprises three main methods; Centre of Gravity (COG) utilises the geometric centre of the detected objects' features to improve the accuracy of the objects' localisation. The boundaries of the object are refined and the detection reliability is enhanced by analysing spatial distribution of points in 3D space. Latent Surface Shape (LSS) identifies the pattern of the indoor settings by focusing on the surface that an object rests on. This approach helps explain the differences in object shapes. Although this approach allows small object detection to be done easier, it can struggle with objects with unclear supporting surfaces[27]. Lastly, H3DNet[28] is a neural network that uses geometric shapes to detect objects. A detailed description of points is created for predicting the object's shapes. This is then used to make and improve proposals of object detections. The network refines object detection by adjusting the bounded box or other geometrical shape followed by classifying the object and eventually adding labels[23].

Lastly, **3D convolution based models** which highlight the importance of texture data that is obtained from the 3D object detection. There are several methods utilising both image and point clouds for enhancing overall performance with fusion schemes implementation.

The series of sliding windows including The Sliding Shapes method and Deep Sliding Shapes (DSS). The Sliding Shape approach performs the object classification using 3D sliding windows which are able to handle occlusions and changes in viewpoint effectively. However, the processing can be slowed down due to the hand-crafted features which this method relies on extensively. For the Deep Sliding Shapes, a 3d convolutional neural network (CNN) is used to improve the efficiency upon the Sliding Shapes method. This significantly solved two main challenges; speeding up the target detection and eliminating the need of CAD design that is done manually. Although the performance is enhanced, the demand for higher computation which is a result of 3D convolutions complexity.

3D object detection performance comparisons on 10 classes on SUN RGB-D datasets.

Methods	Bathtub	Bed	Bookshelf	Chair	Desk	Dresser	Nightstand	Sofa	Table	Toilet	mAP-10	Running Time
DSS [45]	44.2	78.8	11.9	61.2	20.5	6.4	15.4	53.5	50.3	78.9	42.1	19.55 s
2D Driven [56]	43.5	64.5	31.4	48.3	27.9	25.92	41.9	40.39	37.0	80.4	45.1	4.2 s
PointFusion [66]	37.26	68.57	37.69	55.09	17.16	23.95	32.33	53.83	31.03	83.80	45.4	1.3 s
COG [61]	58.3	63.7	31.8	62.2	45.2	15.5	27.4	51.0	51.3	70.1	47.6	10–30 m
3D-SSD [57]	57.1	76.2	29.4	9.2	**56.8**	12.3	21.9	1.7	32.5	38	50.7	0.22 s
LSS [62]	76.2	73.2	32.9	60.5	34.5	13.5	30.4	60.4	55.4	73.7	51	
F-VoxNet [60]	43.3	81.1	33.3	64.2	24.7	32.0	58.1	61.1	51.1	90.0	53.89	0.12 s
F-PointNet [58]	43.3	81.1	33.3	64.2	24.7	32.0	58.1	61.1	51.1	90.9	54.0	0.12 s
VoteNet [50]	74.4	83.0	28.8	75.3	22.0	29.8	62.2	64.0	47.3	90.1	57.7	
MLCVNet [68]	**79.2**	85.8	31.9	75.8	26.5	31.3	61.5	66.3	50.4	89.1	59.8	
H3DNet [63]	73.8	85.6	31.0	76.7	29.6	33.4	65.5	66.5	50.8	88.2	60.1	
HGNet [69]	78.0	84.5	35.7	75.2	34.3	37.6	61.7	65.7	51.6	**91.1**	61.6	
Group-Free [49]	77.5	87.0	35.7	**78.1**	32.0	37.2	66.7	70.2	**52.2**	91.3	62.8	
IMVoteNet [106]	75.9	**87.6**	**41.3**	76.7	28.7	**41.4**	**69.9**	**70.7**	51.1	90.5	**63.4**	

Figure 2.1: 3D object detection performance comparisons on 10 classes on SUN RGB-D datasets. Table reused from *Recent advances in 3D object detection based on RGB-D: A survey*[23]

VoteNet employs the voting mechanism when locating objects as well as generating high quality proposals which both help solve the sparse point cloud issue, but this can be limited by noisy or missing data points. Additionally, imVoteNet is built on the VoteNet by incorporating 2D image data together with texture and semantic information added. This enhances the object detection performance especially when there is a sparse point cloud data. Although the 2D and 3D data fusion increases the accuracy, the complexity is raised as well.

The VoteNet's object detection is enhanced by BRNet which traces representative points back to their original vote centres and re-examining point clusters, allowing a more detailed understanding of object structures which also comes with higher cost of computational requirement due to higher complexity. MLCVNet improves VoteNet's performance by incorporating multi-level contextual information which enables the model to better grasp the relationships between objects and their surroundings, leading to more accurate detections. However, the increased complexity of simultaneously modelling both global and local context causes higher computational costs.

The Hierarchical Graph Network (HGNet) is the VoteNet method that adopts a graph-based approach for improving object detection performance. This features a shape-attentive feature extractor as well as integrated global scene contect, making the prediction even more accurate. However, the improved detection precision of this approach leads to higher computational cost as well.

The author performs comparison of the 3D object detection performance of each approach by using the SUN RGB-D dataset which is a single-view RGB-D dataset containing 47 distinct indoor scenes shown in Figure 3.1.

Although 3D convolution-based models tend to outperform other approaches, it's important to consider additional factors when selecting a method for our project, such as computational requirements, cost, deployment, integration with SLAM, and hardware compatibility.

As some 3D object detection approaches build upon 2D object detection methods, such as convolutional networks, it's equally important to evaluate the various avail-

able 2D object detection approaches in terms of their concepts, functionality, and performance.

There are two major stages in 2D object detection development; the traditional stage and the new stage with deep learning. The concept of traditional object or target detection uses sliding window methods to generate boxed on target images or videos followed by manual feature extraction. Lastly, classifiers like Support Vector Machine (SVM) and Logistic regression will classify the extracted features and there will be a box bounded around the target position. This can be seen in some models like SIFT and Cascades and some typical algorithms are HOG[29], Viola Jones[30], etc. which have limitations causing low detection speed and precision.

In the second stage, Convolutional Neural Network (CNN) helps increase average detection accuracy by approximately 30%[31]. There are two major algorithms in target detection with deep learning, one stage target detection and two-step target detection.

One-stage object detection algorithms predict both the coordinates and class probabilities of objects in a single step using a single neural network[32]. Two prominent examples of these algorithms are Single Shot MultiBox Detector (SSD) and You Only Look Once (YOLO), both of which perform differently in various applications. SSD employs a VGG16 backbone for feature extraction and generates predictions at multiple layers, enabling it to handle objects of varying scales without the need for a region proposal network[33]. On the other hand, YOLOv8, the latest iteration of the YOLO family, uses a more advanced Dark-53 backbone, alongside improved data augmentation techniques and anchor box clustering, which collectively boost its accuracy and efficiency[31]. To compare the performance of SSD and YOLOv8 in multi-object detection tasks, factors such as accuracy, adaptability, and speed are analysed. This comparison is based on the findings from the study "Performance Analysis of YOLOv8, RCNN, and SSD Object Detection Models for Precision Poultry Farming Management," where SSD, YOLOv8, and Faster R-CNN were tested for multi-object detection, as shown in Table 3.1.

Additionally, Two-stage Target Detection involves two steps: the first stage detects potential object locations, while the second stage refines these locations using a deep neural network to extract features from the proposed regions[32]. The addition of a Region Proposal Network (RPN) in the second stage introduces extra computational demands, resulting in higher processing time and hardware requirements compared to one-stage detectors[34]. The main detectors in this category include; R-CNN detector uses selective search to generate region proposals before applying CNN to each region to classify the target and bound each target with boxes. However, this approach is computationally expensive as thousands of regions are processed by CNN independently[14].

Fast R-CNN improves R-CNN by integrating region proposal generation and classification into one network for faster processing. It uses a CNN to process the image and classify ROIs from a shared feature map but still relies on external region proposals, limiting its real-time performance[35].

Faster-CNN eliminates the need of selective search as it features an internal Regional Proposal Network (RPN) offering a strong balance between speed and accuracy which makes it suitable for real-time task[36].

In addition to the aforementioned characteristics of various one-stage and two-stage target detection algorithms, three selected models; YOLOv8, SSD, and Faster-CNN are compared focusing on their speed, precision, and adaptability. The result is measured from the multi-object detection in precision poultry farming management.

Algorithms	Recall Value	Mean Average Precision (mAP@0.5)	Precision
YOLOv8	1.00	98.7%	96.77%
SSD	0.65	77%	89%
Faster R-CNN	0.67	59%	77%

Table 2.1: A comparative analysis of YOLOv8, SSD, and Faster R-CNN based on key performance metrics, including Recall Value, Mean Average Precision (mAP@0.5), and Precision. Table reused from *Performance analysis of YOLOv8, RCNN, and SSD object detection models for precision poultry farming management*[37]

Following this comparison, it is evident that YOLOv8 excels in multi-object detection, combining high accuracy with balanced speed and efficiency. This makes YOLOv8 particularly well-suited for real-time applications. Furthermore, YOLOv8's lower hardware demands make it a more practical choice compared to the more resource-intensive Faster R-CNN and other two-stage detectors[38].

2.3 SLAM

SLAM, or Simultaneous Localization and Mapping, is a widely spread algorithm for navigation in the field of mobile robotics because of the exponential improvement in computer processing speed and the accessibility of sensors such as cameras and LiDAR [39]. Using SLAM, a mobile robot can construct an internal environment map while simultaneously using the map to estimate its location without needing predefined knowledge of area [40].

Environment mapping is one of the vital techniques in SLAM. The algorithm consists of building a mathematical model for the spatial information of an actual environment, which encapsulates the necessary information for navigation and interaction. However, as for the SLAM technique, additional requirements are needed; the mathematical model must be able to represent the robot's state and the position of landmarks relative to the robot's location [40]. Hence, the challenge with the requirements is that the robot must perform the localization and the mapping simultaneously.

Given these complexities, the backbone of all principal SLAM methods is the utilization of these SLAM frameworks consisting of odometry, landmark prediction, landmark prediction, landmark extraction, data association, and matching, pose estimation, and map update [41].

Building on this, situational awareness becomes an extreme component of SLAM, the precision and accuracy of the robot's perception play a huge role in defining the characteristics of other variations of the SLAM implementation. Therefore, a thorough understanding of advantages and disadvantages of each common perception device is an imperative concept not just for the robot's components but also the structure of the SLAM's backend algorithm.

Firstly, acoustic sensors are widely used across the preliminary stage of SLAM implementations to minimise the pose drift with time, with most of the sensors being SONAR, or Sound Navigation and Ranging [42]. These sensors are well operated in dark environments, as well as dusty and humid, due to their insensitivity towards illumination and opaqueness [43].

Secondly, LiDAR, or Light Detection and Ranging Sensor, is relatively similar to an ultrasonic sensor in terms of functionality [42]. However, LiDAR uses electromagnetic waves as a radiation reference instead of acoustic waves. A LiDAR renders a 3-dimensional representation of its surroundings known as the Point Cloud [44]. The strength of LiDAR is that the sensor can provide 360 degrees of perception with high precision [45].

Thirdly, depth cameras' mechanism works based on the illumination of the site with infrared light and measures the time-of-flight [46]. Comparing the range of measurement and accuracy, a depth camera performs poorer than a 3D LiDAR scanner because the depth camera can only acquire data within a limited range of field of view; moreover, environmental factors may affect the accuracy of the depth camera; for example, the depth camera's output is susceptible to certain materials of surfaces, such as reflective or transparent materials [47]. However, a depth camera is still a popular option for SLAM as it's a relatively economical device compared to its relatives, 3-D LiDAR, for instance.

Ultimately, event-based cameras present the local bitmap-level motion alterations to an event that took place, which is different from conventional framing-based cameras [42]. The new technique has gained popularity more recently in the field of SLAM as an event-based camera yields more efficient computational performance and better overall accuracy [48].

After reviewing the different sensors and addressing technological advancements available in today's world, it is crucial to understand how researchers have implemented those ideas to different variations of SLAM. This understanding helps in overcoming challenges and limitations that their predecessors had faced and set new standards for new research frontiers. Moreover, it becomes essential to effectively classify those SLAM variations under different criteria.

Li et al.[20] perfectly encapsulated how SLAM techniques can be classified:

Simultaneous Localization and Mapping (SLAM) techniques can be categorised by using different factors. Firstly, they can be divided into categories based on the type

of sensors employed. They may include vision-based SLAM using cameras, LIDAR-based SLAM using LIDAR sensors, and RGB-D SLAM, which combines RGB cameras with depth sensors. Secondly, feature-based SLAM, which tracks distinguishing characteristics and direct SLAM, which executes mapping intensity or depth directly can be considered as different categories. Thirdly, the estimated approach, such as filter-based SLAM, which uses filters such as Particle Filter and graph-based SLAM, which is formulated as a graph optimization problem, provides another classification criterion. Finally, SLAM can be categorised based on time synchronization, with offline SLAM processing data in batches after collection and online SLAM estimating pose and map incrementally in real-time.

Among the various SLAM methods, one of the most well-known is Visual Slam, or V-SLAM, a variation of SLAM that uses images from cameras, ranging from a conventional camera to an RBD-D camera (depth and ToF camera). V-SLAM itself can be divided into two sub-categories [49]. Firstly, a feature-based SLAM system that matches camera data using sparse methods; one example of this robust and popular algorithm is the ORB-SLAM [50]. Secondly, the direct dense methods that evaluate based on the general luminance of the pixels in images, one of the famous algorithms is Direct Sparse Odometry [51].

In addition to V-SLAM, another key variation is LiDAR-based SLAM, which utilises LiDAR sensors to localise itself by collecting data from its surroundings while building the map of the data representation. Registration algorithms, for instance, iterative closest point (ICP), are used to estimate the relative transformation of the point clouds during the operation [52]. On the other hand, feature-based algorithms, such as LiDAR Odometry and Mapping, are used to represent 2D or 3D point cloud maps as grid maps [53].

Furthermore, by combining the strengths of different sensors and overcoming each's limitations, the multi-sensor SLAM utilises multiple sensors, such as cameras, Inertial Measurement Units (IMUs), Global Positioning System, LiDAR, etc. FAST-LIO, or a Fast LiDAR-Inertial Odometry, is a decent example of an algorithm that combines LiDAR feature points with IMU data [54].

As SLAM continues to evolve, one of the emerging trends is collaborative SLAM, especially in a distributed framework. This includes multi-robot SLAM that leverages stability of the system by minimizing global error accumulation, risk concentration [55].

In this context, Slide-SLAM presents a real-time decentralised metric-semantic SLAM method that creates an object-based representation to append autonomous exploration functionality to a robot team. This is achieved by attaching a communication module to each robot, and then a unified map is obtained from each robot's observation [56].

Similarly, C-SLAM, an open-source system of Swarm-SLAM that has main purposes to be a scalable, decentralised, and sparse system for multi-robot, especially swarm-like robot teams, to perform navigation operations in unknown environments. C-SLAM is designed to support IMU, LiDAR, stereo, and RGB-D sensing [57].

2.4 Collective Movement

In the domain of swarm robotics, collective movement coordination and dynamic role assignment are crucial for enabling robots to work together efficiently. Research on coordinated motion in swarms often emphasises the need for algorithms that allow robots to adapt their roles and behaviours in real-time. For example, the study on "Efficient Strategies for Coordinated Motion and Tracking in Swarm Robotics" is a comprehensive overview of various coordination algorithms, contrasting different techniques for multi-robot collaboration.

The first coordination algorithm mentioned is the leader-follower model. This algorithm is rather straightforward in the sense that one or more robots are designated to guide the swarm while the other robots adjust their positions. The leader can be pre-programmed or autonomously chosen depending on the path while the followers maintain a set distance and set angle. This model as mentioned before is simple to apply while also being centralised providing clear direction for the followers. Additionally, the followers do not need the full knowledge of the environment meaning that this model can be scalable. However, this swarm being centralised means that it is prone to a single point of failure and having reduced flexibility [58]. This model would only work well for a simple structured environment with predefined paths which unfortunately does not match with our objectives.

Another coordination algorithm is the potential fields algorithm. This algorithm is based on virtual forces with each robot in the swarm being treated as a particle that is influenced by virtual forces exerted by other robots, obstacles and targets. These forces can attract or repel each other. The object is for the robot to be “pulled” towards the goal while avoiding collisions. This model has a couple advantages; namely: Decentralised control as each robot moves autonomously based on the forces acting on it, and smooth movements. However a couple of challenges come with it as well. One challenge is the possibility of the robot being stuck in a local minimum where virtual forces cancel each other out. Secondly, proper fine tuning of the force parameters is required to prevent the robot from oscillating [59]. Overall this approach could be useful for environments with many obstacles where smooth and continuous navigation can be important. The third algorithm offered is the virtual force algorithm which is similar to the potential fields algorithm but with more constraints thereby being ineffectual to our project [42].

When comparing these three algorithms above, potential field algorithm and virtual force algorithm are decentralised while the leader-follower model is centralised. While the leader-follower model offers a simple, scalable nature, it introduces a single point of failure. In contrast, the potential fields algorithm offers a more decentralised approach, which is better suited for dynamic environments but requires careful turning to avoid local minima.

In swarm robotics, dynamic role assignment plays a crucial role in enabling robots to adapt their behaviours and tasks in real-time. One common method is insect-inspired behaviour, which mimics the role distribution seen in social insect colonies [60]. In this approach, robots assume roles based on simple, local rules, such as task demand

or proximity to a target, without centralised control. This method offers high scalability and robustness, as robots can seamlessly take on different tasks as needed, making it suitable for large swarms. However, its reliance on local information can sometimes lead to suboptimal task assignments, particularly in complex environments where global awareness might be needed.

On the other hand, market-based approaches [61] use a more structured mechanism where robots bid for tasks based on their capabilities and availability. This ensures that tasks are allocated to the most suitable robots, leading to more efficient task execution. However, the bidding process requires communication between robots, which may introduce delays and increase system complexity. While market-based approaches tend to be more optimal for task allocation, they may not scale as easily as insect-inspired methods, especially in large or dynamic environments where constant communication is challenging.

Decentralised control in swarm robotics offers several key advantages, particularly in terms of scalability, robustness, and adaptability [62]. In decentralised systems, each robot operates autonomously, relying on local information and interactions with neighbouring robots, which eliminates the need for a central controller. This allows the swarm to scale more easily, as adding more robots does not increase the computational or communication burden on a single entity. Additionally, decentralised systems are more robust, as the failure of one or more robots does not compromise the entire system; each robot can continue functioning independently. This is particularly advantageous in dynamic or unpredictable environments, where flexibility and fault tolerance are critical.

In contrast, centralised control systems rely on a single controller to manage all robots, which creates a bottleneck as the number of robots increases. Centralised systems can suffer from single points of failure—if the controller fails, the entire system may halt. Moreover, communication delays and computational limits can hinder real-time performance in larger systems. While centralised control offers more efficient coordination in smaller, simpler environments, decentralised control is better suited for real-world applications where scalability and resilience are essential for handling complex and dynamic tasks.

Despite significant advancements in swarm robotics and coordination algorithms, there remain notable gaps in applying these methods to practical, real-world environments such as cleaning tasks. Most research on algorithms like potential fields, leader-follower models, and market-based role assignment has focused on simulations or controlled environments, which often lack the complexity and unpredictability found in real-world scenarios. For instance, limited work has been done on integrating these algorithms with sensor-rich, dynamic settings where robots must navigate cluttered spaces, identify and manipulate diverse objects, and coordinate in real-time without centralised control. Additionally, the scalability of these systems is often not tested in practical, large-scale environments, such as a commercial building cleaning system, where communication constraints, battery life, and real-time decision-making are crucial factors.

Chapter 3

Simulation Overview

For our simulation, we are working with three Turtlebots in the Webots simulation within a square arena. Initially, each robot faces parallel to the x-axis, resulting in a starting orientation angle (theta) of 0. This setup acts as a calibration step to ensure that every robot's odometry is correctly aligned with a theta of 0. The orgin x, y coordinates of each robots are given by the simulation, afterwards the location of the robots is updated based on the encoder odometry data.

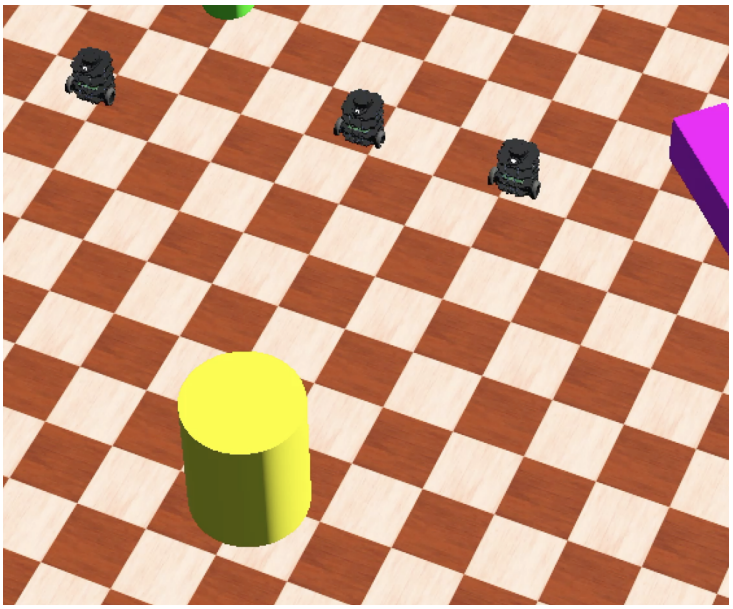

Figure 3.1: Starting of the simulation

Once the simulation starts, the robots move based on a predefined polynomial path rather than random motion. Typically, this movement involves slight left turns. The simulation is designed so that two of the robots will detect a yellow object. If any of the robots detects the yellow object, it will stop and send a signal to the other two robots. The robots simultaneously share their location data, forming a mesh communication system.

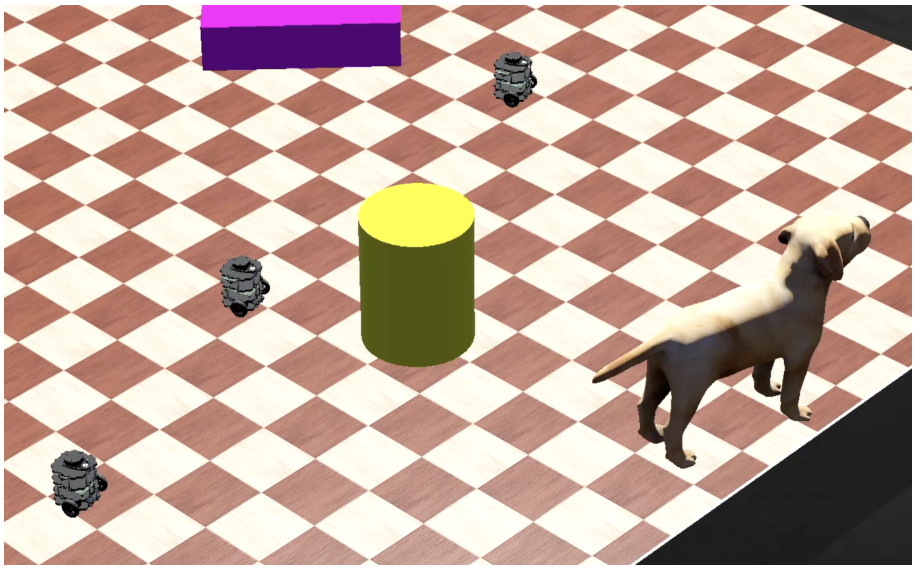

Figure 3.2: 3 robots following a path before an object is detected

To handle potential multi-detections, a consensus mechanism based on an additional priority queue ensures that even if two robots detect the object at the same time, the system processes the detection correctly. When the first robot detects the yellow object, it captures the object coordinates by inferring from a known database and calibrated camera. Since this robot already knows the locations of the other two, it calculates the paths for them to reach the object and grasp it.

The host robot — the one that detected the object — sends these paths to the other two robots. Once the message is received, the two robots acknowledge with a confirmation message. All three robots then follow the calculated paths to the destination, avoiding any obstacles and collisions along the paths. For path-following, a PID controller is used. The PID controller calculates the error in distance to each waypoint and adjusts the motor velocity or force accordingly. When the robots reach the detected object, they stop.

Figure 3.3: 3 robots converged to the detected object

The main challenge of this simulation was ensuring reliable communication among the three robots and accurately exchanging data.

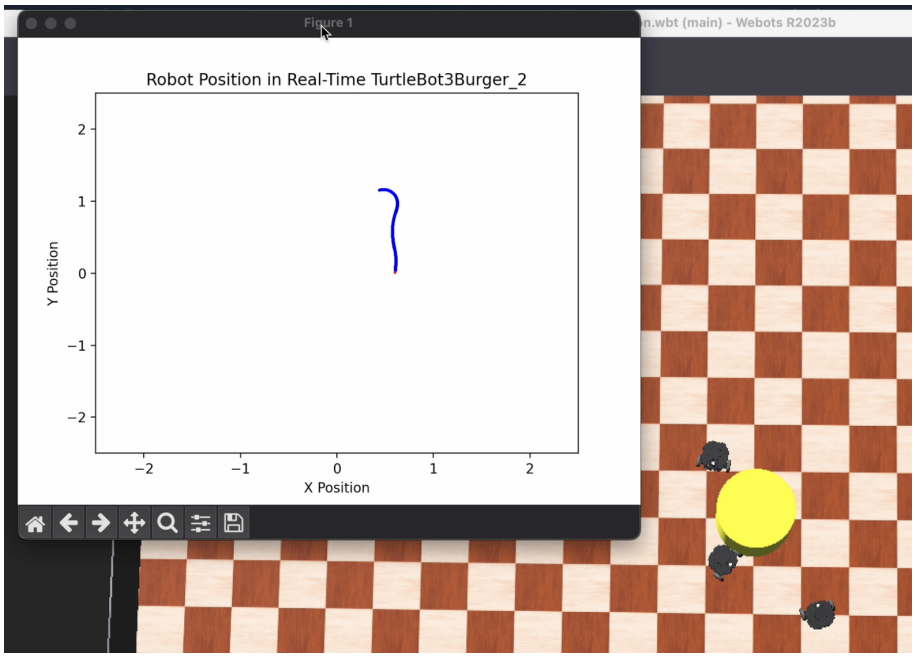

Figure 3.4: 3 robots converging to the pre-determined position.

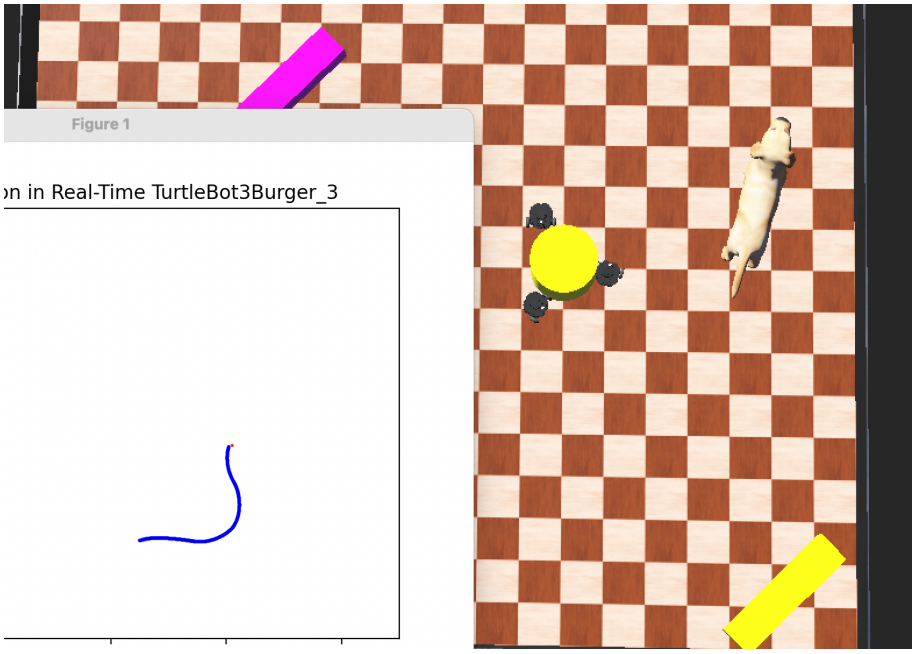

Figure 3.5: Path of the TurtleBotBurger3 after receiving path to follow from the main controller.

Chapter 4

Object Detection using Computer Vision

In the current simulation on Webots, the object detection system utilises OpenCV with a masking method to identify and interact with a yellow object in the arena. The task involves detecting and moving a yellow cylindrical object, randomly placed within the arena, to a specified location.

The detection process commences with the robot utilising an RGB camera mounted on the front-most part of the TurtleBot3 to capture its surrounding environment. To address potential issues related to the robot's physics and balance, the RGB camera is approximated as a point source, as illustrated in Figure 4.1. The captured image is subsequently converted into the HSV colour space to facilitate the segmentation of the yellow object based on a predefined colour range. A mask is then applied to the regions containing the yellow colour, enabling the generation of a contour around the detected object. This contour functions as a visual boundary, allowing the robot to perceive and localise the object with precision.

Figure 4.1: The location of the RGB camera mounted on the TurtleBot3

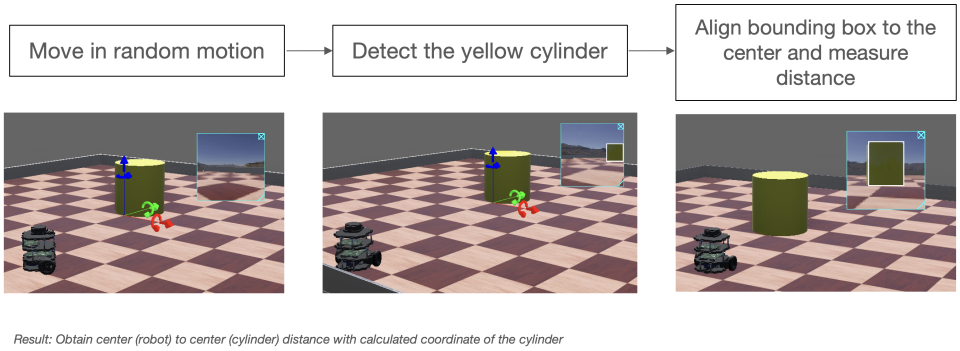

Figure 4.2: Object Detection Flowchart

Once the object is detected, the robot moves until the contour is centred within the camera frame, ensuring that the robot faces the object directly. This alignment is achieved by comparing the distances between the left and right edges of the bounding box surrounding the contour and the corresponding edges of the camera frame. Specifically, the distance between the left edge of the frame and the left edge of the bounding box is compared with the distance between the right edge of the frame and the right edge of the bounding box. The robot stops when the difference between these two distances is less than or equal to 1, as the camera calculates distances as whole numbers. Using a threshold of ≤ 1 is more reliable than requiring exact equality, as slight variations in measurement may occur due to the camera's resolution or other factors. See Figure 4.2

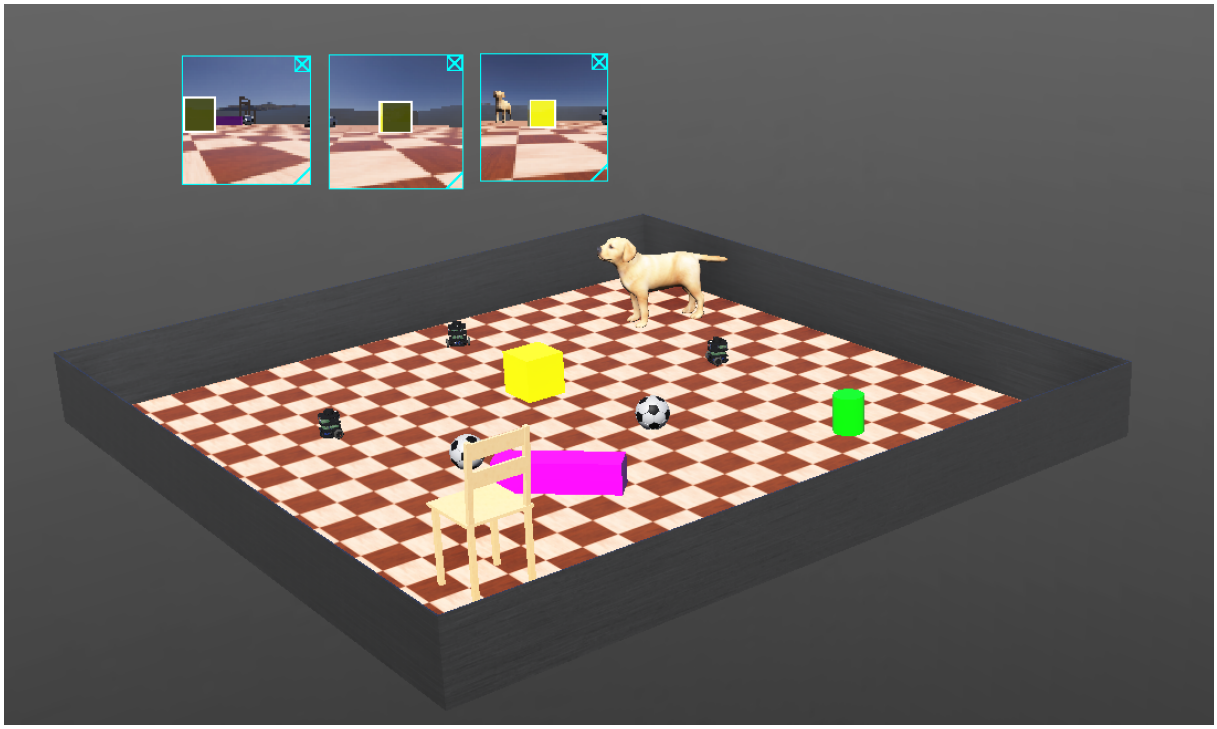

Figure 4.3: The location of the RGB camera mounted on the TurtleBot3

After aligning with the object, as illustrated in Figure 4.3, the robot determines the real-world distance between its current position and the object's surface. By utilising

the known distance and real-world dimensions of the object within the simulation, the system computes the real-time distance from the robot's camera to the object's surface. This information guides the robot in determining the distance it must travel to reach the yellow cylindrical object. The calculation is performed using a pre-determined focal length of the camera, derived through the following formulas:

$$\mathrm{f} = \frac{H_{\text{real}} \cdot D_{\text{known}}}{H_{\text{img}}} \tag{4.1}$$

$$\mathrm{d}_{\text{real}} = \frac{H_{\text{real}} \cdot f}{H_{\text{img}}} \tag{4.2}$$

Where:

- f represents the focal length of the camera (in pixels),
- H_{img} is the known width of the object in the image (in pixels),
- D_{known} denotes the known distance to the object (in metres),
- H_{real} is the real-world width of the object (in metres),
- f is the focal length of the camera (in pixels).

To calculate the real-world distance, the process begins by determining the focal length of the camera. The positions of both the robot and the cylinder are defined within the simulation. The distance perpendicular to the outermost surface of the cylinder and the RGB camera mounted on the robot is denoted as D_{known}. The cylinder's height in the real world, defined in the simulation, is represented as H_{real}, while H_{img} corresponds to the detected height in pixels, obtained via the OpenCV model. Using Equation 4.1, the focal length f is calculated. This focal length is then applied in Equation 4.2, along with H_{real}, H_{img}, and f, to compute d_{real}, the real-world distance. Once the real-world distance is obtained, we will add the radius of the target and the distance between the centre of the TurtleBot and the RGB camera so that we can get the distance from the centre of the robot to the centre of the target, which will be utilised further for determining the coordinate of the target on the map.

The location of the target can be determined using data from the robot's odometry, including its position (x, y) and orientation (yaw) relative to the origin (0, 0) of the map, as shown in Figure 4.4.

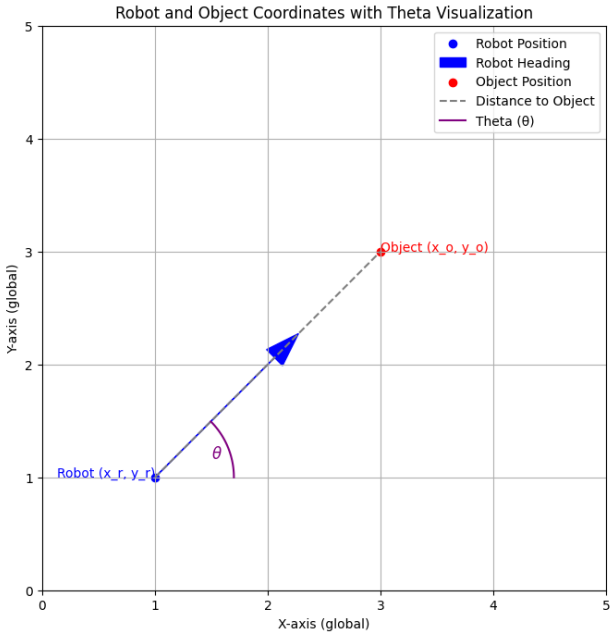

Figure 4.4: The location of the RGB camera mounted on the TurtleBot3

By applying trigonometry, the coordinates of the target (x_o, y_o) can be calculated as shown in Equations 4.3 and 4.4.

Given the robot's position (x_r, y_r), heading θ (measured counter-clockwise from the global x-axis), and the distance d to the object, the target's coordinates (x_o, y_o) can be computed using the following equations:

$$x_o = x_r + d \cdot \cos(\theta) \tag{4.3}$$

$$y_o = y_r + d \cdot \sin(\theta) \tag{4.4}$$

Where:

- x_r, y_r are the robot's coordinates,
- d is the distance between the robot and the target,
- θ is the robot's heading (positive angle is counter-clockwise from the global x-axis).

This equation works for both positive and negative values of θ, as trigonometric functions inherently account for the direction.

Several adjustments were made to improve the detection system. Increasing the simulation's luminosity setting from 1 to 1.2 enhanced the brightness, ensuring the environment's colours closely resembled real-world conditions. Consequently, the HSV colour range for detecting yellow was refined to $(70, 100, 0)$ to $(90, 255, 255)$, making the

detection process more reliable and accurate. These modifications addressed earlier issues with colour recognition and bridged the gap between simulated and real-world object detection.

Regarding the evaluation of whether the project met its expectations, the object detection performance is somewhat below the level anticipated in the project proposal. The original plan included advanced detection capabilities, such as distinguishing objects of different colours and geometries. However, this goal was deprioritised to concentrate on establishing the functionality of other essential components, such as communication and localisation. The advanced object detection features will be addressed during the transition from simulation to the physical robot to minimise inefficiencies, such as retraining the model specifically for real-world conditions.

One of the primary challenges encountered during the development of the object detection simulation was related to the programme configuration. Specifically, it was necessary to readjust the luminosity settings in the Webots environment to achieve optimal detection performance. Additionally, there was a limitation with the device used for displaying real-time images from the camera. Instead of using the camera overlay, which could not render the bounding box generated by the OpenCV model, the display device had to be utilised to perform this task effectively.

The current system reliably detects and interacts with a single yellow object. Moving forward, the next step is to transition the object detection simulation to a real-world setting, beginning with cylindrical objects and other simple geometric shapes. The ultimate objective is to enable robots to detect and interact with objects of diverse geometries and colours. This will include addressing challenges such as object pose estimation, thereby enhancing the system's adaptability and versatility in real-world applications.

In summary, the object detection system combines effective visual processing techniques with a robust control mechanism, allowing robots to detect, align with, and calculate the distance to a target object. These advancements mark a significant step towards achieving the project's broader objectives.

Chapter 5

Communication in the swarm

This chapter gives the end results for the test from the Webots simulation on the sub-task **Communication in the swarm**. This sub-task resolves around the assumption that the *swarm fleet is capable of object detection*, which is the key for task parallelization in our system. The main components of the communication system consist of the protocol, the robot's modes (states), consensus, and system integration.

In general, the main robot we use is the Burger model of the third version of the TurtleBot robot, which has two functioning motors and a pre-built LiDAR system. Additionally, three more basic nodes are added to the TurtleBot object, which are the following: Firstly, the GPS node (Figure 2.1), this module is installed to obtain the robot's position in the simulated world. This direct approach is used because the localization and mapping process is not available yet. Secondly, the Emitter node (Figure 2.2), this module is added to the robot to model a broadcast behavior. Thirdly, the receiver node (Figure 2.3), this module is connected as an accompaniment to the Emitter Class because the emitter node does not support both emitting and receiving functionalities. By combining all the devices, the basic simulation of swarm robotics communication is completed.

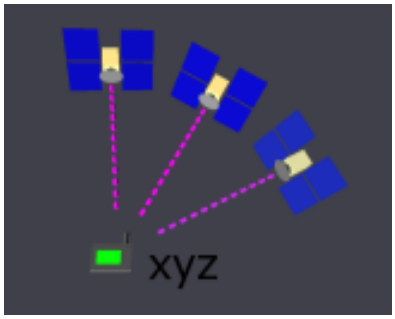

Figure 5.1: GPS node

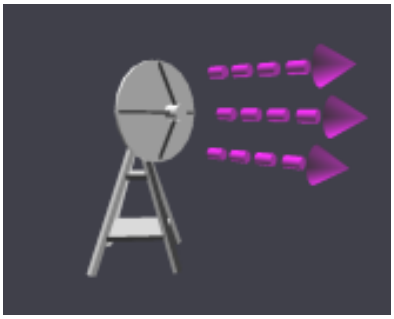

Figure 5.2: Emitter node

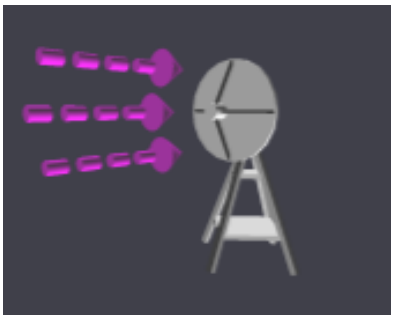

Figure 5.3: Receiver node

This approach allows simple communication in the swarm system and awareness of other robot members in the swarm system. We then further build the system's protocol to ensure useful and safe information is transmitted across a reliable platform. These messages are processed through states combining structured communication with dynamic state transitions.

The protocol is built to support the incoming series of structured messages in a queue (Table 2.1) and robot states (Table 2.2).

Messages such as [probe] is used for updating position data. When an object is detected, a [task] message is broadcast, designating the detecting robot as the task master and informing the object's location to others. If conflicts occur, with two or more robots detecting the object at the same time, the robots will send a [task_conflict] message, notifying other robots, and resolve them through a priority-based reassignment. Once tasks are completed, the [task_successful] message signals the robot to transition to the next stage.

probe	Broadcasts real-time position data for efficient task coordination. **Sender**: Sends robot's position data. **Receiver**: Updates information the robot_entries dictionary. **Format**: {[probe], robot_id, message_id, (robot_position["x"], robot_position["y"], robot_position["theta"])}
object_detected	Broadcasts the encounter of the target object. **Sender**: Sends object's position data. **Receiver**: Changes its state to *idle* and waits for further action. **Format**: {[object_detected], robot_id, found cylinder @ {cylinder_position}}
task	Broadcasts the assigning of a new taskmaster for the path planning. **Sender**: Sends an initiative to assign itself as the taskmaster. **Receiver**: Changes its state to *task* and report conflicting status. **Format**: {[task], robot_id, message_id, cylinder_position}
task_conflict	Broadcasts the conflict occurs in task master assignment. **Sender**: Sends conflict information to other robots. **Receiver**: Change its state to *reassign*. **Format**: {[task_conflict], robot_id, message_id, priority_queue}
task_successful	Broadcasts a success operation in task master assignment. **Sender**: Confirms the success of the operation and the task master. **Receiver**: Ensures all data and information complies to itself. **Format**: {[task_successful], robot_id, message_id, task_master}
path_following	Broadcasts a notification for robots to follow the given paths. **Sender**: Sends planned path to each robots. **Receiver**: Changes its state to *path_following*, and updates its path. **Format**: {[path_following], robot_id, message_id, paths_json}

Table 5.1: Message Types

Each message may trigger a state transition; this is done by the communicator handling the message from the queue. Robots may start in the *idle* or *NaN* state, finding the object. If one of the robots detects an object, it will change to *consensus* state, where the task master is selected and agreed upon in this phase. The taskmaster then transitions to *path_finding* to generate navigation routes for both itself and other robots. After that, when all robots receive their own routes to the destination, it will switch to the *path_following* stage.

idle (Default)	State where the robot halt and wait for instructions
consensus	State where an agreement of the task master is decided.
path_finding	State where the task master generates navigation paths for all robots.
path_following	State where the robot follows it's path.
task	States that handles task management, particularly conflicts.
reassign	States that reassigns the task master when conflicts arise.

Table 5.2: Robot States

The consensus mechanism used in this project is inspired by the Paxos Algorithm but is more lightweight. We are making a protocol that allows a group of robot agents to agree on a single value, even if some of the agents fail to be responsive. In our case, the major concern that we need to tackle is a scenario of multiple robots detecting the object at the same time, and the allocation of taskmasters needs to be agreed upon by all parties. To do that, we added an intermediate state after an object is detected. And if the detected robot receives *object_detection* from other robots before receiving confirmation messages, it will raise the error and can send *task_conflict* to other robots.

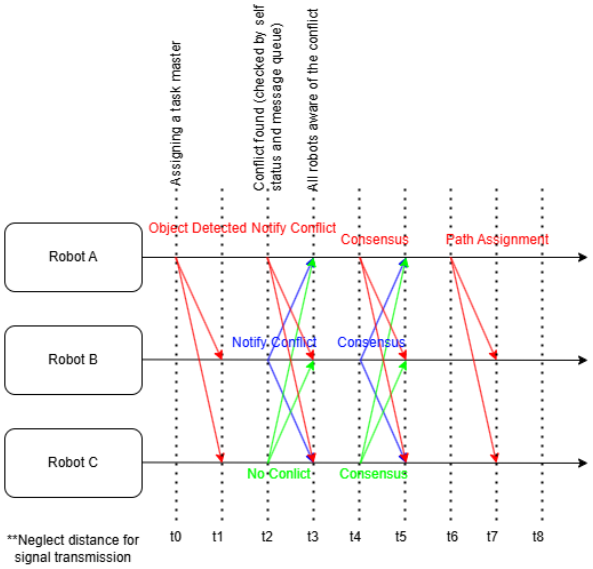

Figure 5.4: Conflict State

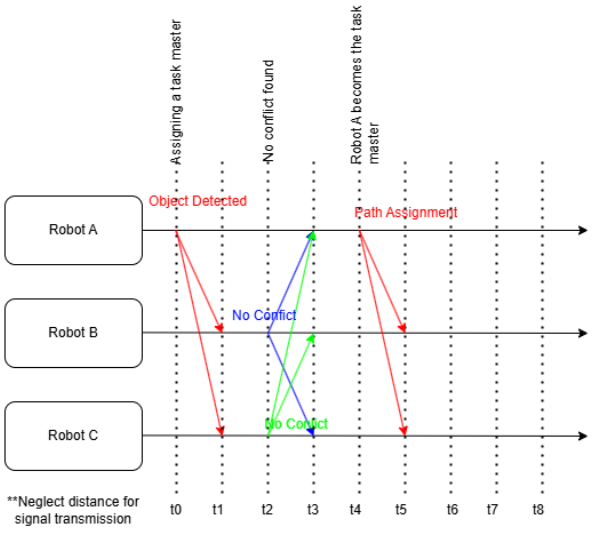

Figure 5.5: Normal State

Chapter 6

Localisation

This chapter focuses on robot localisation and simultaneous localisation and mapping (SLAM) in a simulated environment. Localisation is achieved through odometry computations to determine the robot's position and orientation, while Graph SLAM builds on this foundation to simultaneously construct a map and localise the robot within it.

6.1 Localisation Using Odometry

Robot localisation involves computing its position and orientation using wheel encoder readings, a method known as odometry. The robot calculates the distance traveled by each wheel, determining the forward movement as:

$$\Delta_{\text{center}} = \frac{\Delta_{\text{left}} + \Delta_{\text{right}}}{2}$$

and the change in orientation as:

$$\Delta_{\theta} = \frac{\Delta_{\text{right}} - \Delta_{\text{left}}}{\text{wheel base}}$$

Using these values, the robot updates its global position and orientation at each step of the simulation. The updated coordinates are given by:

$$x_{\text{new}} = x_{\text{old}} + \Delta_{\text{center}} \cdot \cos(\theta_{\text{old}})$$

$$y_{\text{new}} = y_{\text{old}} + \Delta_{\text{center}} \cdot \sin(\theta_{\text{old}})$$

$$\theta_{\text{new}} = \theta_{\text{old}} + \Delta_{\theta}$$

The system continuously monitors the encoder values during each simulation step, ensuring real-time refinement of the robot's position and orientation. By leveraging these calculations, the robot maintains an accurate understanding of its location within the environment, which is essential for precise movement and navigation.

In this project, we initially set out to implement SLAM for the TurtleBot3 in the Webots simulation environment but realised our implementation was limited to localisation on a predetermined map rather than full SLAM. The TurtleBot3 was configured with motors for continuous movement, enabling exploration and data collection, and equipped with a LIDAR sensor for 360 degree obstacle detection. Encoders tracked

wheel rotations, and odometry calculations used differential drive kinematics to update the robot's position and orientation. Localisation was achieved by updating the robot's position based on encoder data and interpreting LIDAR data relative to a predefined rectangular map with fixed dimensions. This predefined knowledge of the environment meant that the implementation did not fully achieve SLAM, which requires both simultaneous mapping and localisation in an unknown environment. The robot updated its position and visualised its progress on the predetermined map in real-time, with periodic visualisations and debugging ensuring accuracy. Challenges, such as maintaining proper updates and sensor configuration, were addressed systematically.

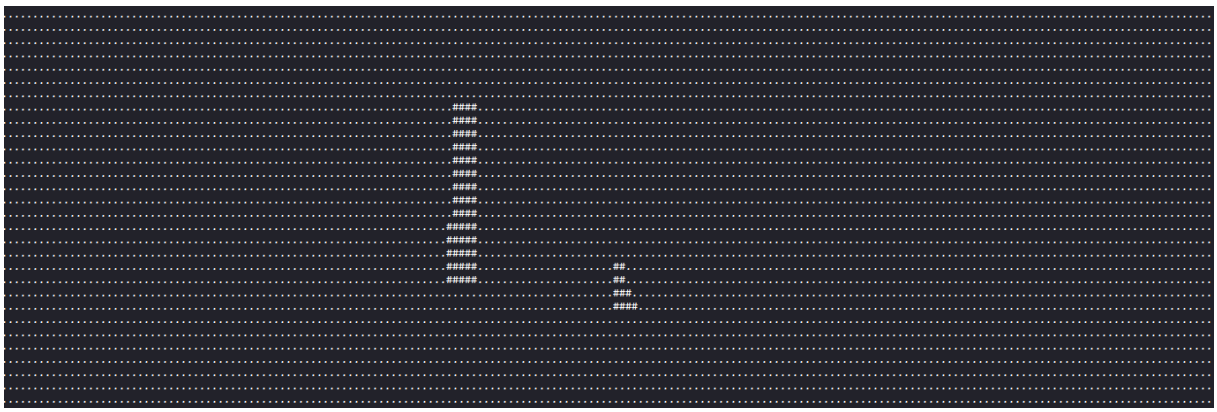

Figure 6.1: A 2-D map depicting the location of the robot in its environment

6.2 Graph SLAM

Graph SLAM builds upon localisation by simultaneously constructing a map and localising the robot within it. The steps involved are outlined below:

6.2.1 Create the Initial Graph Using Odometry

The robot's pose is updated using odometry as:

$$x_t = x_i + \Delta s \cos(\theta_i),$$

$$y_t = y_i + \Delta s \sin(\theta_i),$$

$$\theta_t = \theta_i + \Delta\theta$$

where Δs is the distance traveled.

The edge e_{ij} is represented as:

$$e_{ij} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix}.$$

6.2.2 Loop Closure

When the robot recognises a previously visited landmark, an additional edge is added between non-consecutive poses x_i and x_j where $j \neq i+1$. This edge is based on sensor

measurements, not odometry:

$$e_{ij} = \begin{bmatrix} \Delta x_{ij} \\ \Delta y_{ij} \\ \Delta \theta_{ij} \end{bmatrix}.$$

6.2.3 Defining the Error Function/Residual

$$e_{ij} = \begin{bmatrix} x_j - (x_i + \Delta x_{ij}) \\ y_j - (y_i + \Delta y_{ij}) \\ \theta_j - (\theta_i + \Delta \theta_{ij}) \end{bmatrix}.$$

Here, x_j, y_j, θ_j are the current noisy estimates, and $x_i + \Delta x_{ij}, y_i + \Delta y_{ij}, \theta_i + \Delta \theta_{ij}$ are the expected values for pose j.

6.2.4 Objective Function/Sum of Squared Errors

$$f(x) = \sum_{(i,j)\in \text{edges}} e_{ij}(x_i, x_j)^T \Omega_{ij} e_{ij}(x_i, x_j),$$

where Ω_{ij} is the information matrix. It measures the confidence in measurements, with examples like:

$$\Omega_{ij} = \begin{bmatrix} \omega_x & 0 & 0 \\ 0 & \omega_y & 0 \\ 0 & 0 & \omega_\theta \end{bmatrix},$$

where $\omega_x, \omega_y, \omega_\theta$ are weighted confidences.

6.2.5 Gauss-Newton Optimization

1. Linearise using Taylor expansion:

$$e_{ij}(x_i, x_j) \approx e_{ij}(x_{i0}, x_{j0}) + J_{ij}(x_i - x_{i0}),$$

   where $J_{ij} = \frac{\partial e_{ij}}{\partial x}\Big|_{x=x_0}$.

2. Substitute into the objective function and expand:

$$f(x) \approx \sum_{(i,j)\in \text{edges}} \begin{pmatrix} e_{ij}(x_{i0}, x_{j0})^T \Omega_{ij} e_{ij}(x_{i0}, x_{j0}) \\ + 2e_{ij}(x_{i0}, x_{j0})^T \Omega_{ij} J_{ij}(x - x_0) \\ + (x - x_0)^T J_{ij}^T \Omega_{ij} J_{ij}(x - x_0) \end{pmatrix}.$$

3. Solve the normal equations:

$$H \Delta x = -g,$$

   where

$$H = \sum_{(i,j)\in \text{edges}} J_{ij}^T \Omega_{ij} J_{ij}, \quad g = \sum_{(i,j)\in \text{edges}} J_{ij}^T \Omega_{ij} e_{ij}(x_{i0}, x_{j0}).$$

4. Use Cholesky decomposition for efficiency:

$$H = LL^T,$$

   solve $Ly = -g$, then $L^T \Delta x = y$.

5. Update:

$$x \leftarrow x + \Delta x.$$

   Repeat until convergence.

Chapter 7

Simple Robot Formation

This chapter records the exploratory endeavors into the realm of swarm robotics path planning in an attempt to build a formation around an "object". A brief recapitulation of the reason we require an object is because our end goal is to perform collective transportation using swarm robotics.

In the process of building a swarm formation, computations for the desired coordinates where the swarm units should be moving to (later referenced as **"target coordinates"**), and the desired path the members can take in order to move towards the goal are essential. The key parameters for this calculation include: the **radius** where the swarm should place itself around the object, swarm fleet **member count**, **current coordinates** for the swarm, and target **object coordinates**.

The member count is a vital initial point since it will be used to compute the angle θ between each set of target coordinates using the following formula:

$$\theta = (2\pi/n) * i$$

where:

θ = angle between each set of target coordinates (in radians)

n = swarm system member count

i = member identifier (e.g. 0 to 4 for a swarm fleet of 5 members)

Currently, the formation is determined by utilizing a given radius for the swarm robotic members to attempt to surround. In accordance with the member count providing the angle, the individual target coordinates for each robot can be calculated using an adaptation from the Pythagorean Theorem.

$$x' = x + rcos\theta$$
$$y' = y + rsin\theta$$

where:

(x, y) = robot's present coordinates

(x', y') = target (x, y) coordinates

r = radius

The target coordinates at this stage would often result in floats; therefore, it is necessary to round the numbers and record the margins of error to snap the coordinates to a grid system. With the resulting calculation of the target coordinates, it is sufficient to proceed to the next stage, which is **path planning**. The path planning algorithm follows a simple **"Avoid paths that cause conflicts, but if all paths cause conflicts, pause before continuing"** rule. Hence, recording the paths and the timestep the movement will happen is crucial. The algorithm is presented in Algorithm 1.

For instance, when the intended movement for robot2 and robot3, there is a conflicting movement in *timestep 2* at the *coordinates (3, 3)*. Therefore, robot3 halts for a single timestep before continuing. This is designed so that robots will not take unnecessary detours and create environmental variabilities.

Moreover, The current implementation of the path planning algorithm also takes into account the presence of obstacles presented by the localization and object detection modules as input. The cases for obstacle appearance and avoidance can be divided into two main categories: obstacles appearing at the **turning** location while heading to the destination, and obstacles appearing while heading **straight** towards a destination.

Obstacles appearance during turning can also be considered as the them appearing on a different axis compared to the ongoing movement (robot moving along the x axis, obstacles appear while turning onto the y axis, vice versa). Obstacles encountered this way can be tackled by continuing movement by a single step in the same direction as its ongoing movement, then computing the path as per the usual method. For visualization purposes, this case has been portrayed in Figure 7.1.

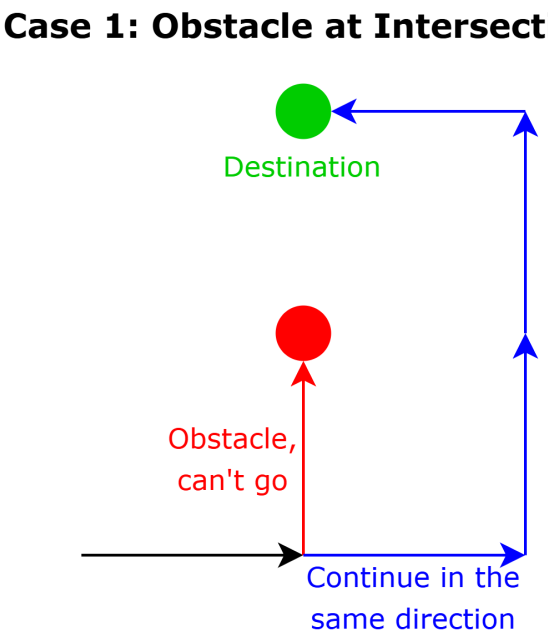

Figure 7.1: Obstacle Avoidance Case 1: Obstacle during Turning

The other case for obstacle appearance can be considered as one where it coincides with the line of movement the robot is currently ongoing (robot moving along the x axis, obstacles appear along that x path). This case can be handled by "going around" the obstacles, appending movements that go around the obstacles both in the positive and negative directions, and continuing the existing path planning algorithm. Figure 7.2 illustrates the case and the proposed solution.

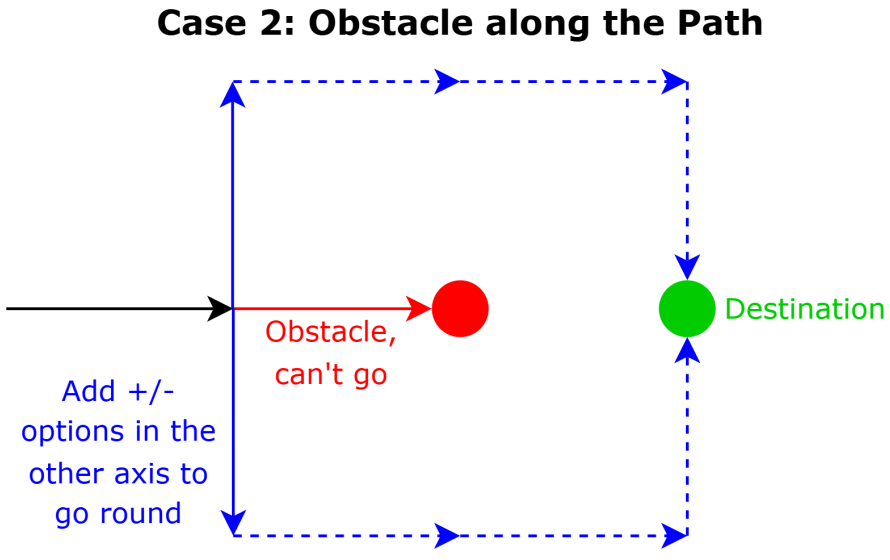

Figure 7.2: Obstacle Avoidance Case 2: Obstacle along the Current Path

Algorithm 1 Path Planning Algorithm

$\Delta x = (x_t - x_s)$
$\Delta y = (y_t - y_s)$
{where subscript *t* stands for target and subscript *s* stands for start}
$x_c =$ **false**
$y_c =$ **false**
{where subscript *c* represents correct positions}
while $x_c \neq$ **true and** $y_c \neq$ **true do**
  if $\Delta x > 0$ **then**
    $x' = x + 1$
  else if $\Delta x < 0$ **then**
    $x' = x - 1$
  else
    $x_c \leftarrow$ **true**
  end if
  $movementOptions \leftarrow (x', y)$ {new x, original y}
  {movementOptions is a collection of possible movements for the robot}
  if $\Delta y > 0$ **then**
    $y' = y + 1$
  else if $\Delta y < 0$ **then**
    $y' = y - 1$
  else
    $y_c \leftarrow$ **true**
  end if
  $movementOptions \leftarrow (x, y')$ {original x, new y}
  for all $movementOption \in conflictMap$ **do**
    $movementOptions - movementOption$
  end for
  {conflictMap is a record of steps and paths from other robots to avoid collisions}
  for all $movementOption \in obstacles$ **do**
    if $movementOptionDir \neq prevMovementDir$ **then**
      $movementOptions \leftarrow (x, y) + prevDir$
      {continue moving in same direction}
    else if $movementOptionDir = prevMovementDir$ **then**
      $movementOptions \leftarrow (x, y) + !prevDir$
      $movementOptions \leftarrow (x, y) - !prevDir$
      {go around the obstacle}
    end if{movement in different/same axis}
  end for
  {obstacles are from localization and object detection}
  if $len(movementOptions) = 0$ **then**
    $chosenMovement \leftarrow (x, y)$ {stay in place}
  end if
  $chosenMovement \leftarrow movementOptions[0]$ {choose first movement}
end while

Chapter 8

Hardware

Due to the predicted cost of hardware, we decided to ask a local Chulalongkorn University robotics club for one of their swarm robots, particularly the robot used in a previous RoboCup soccer competition. While the electronics in these robots are outdated, the motors remain exceptional, as they are Maxon motors, known for their reliability, efficiency, and speed. Hiveground, one of our project supporters, has also agreed to provide us with another identical robot, as one of the founders is an alumnus of the EIC. At the time of writing, we currently have two identical robots in our possession.

The Maxon motors in both robots are arranged in an X layout, paired with high-quality omnidirectional wheels. However, due to a lack of use over the past decade, the rubber on the wheels has decayed. Given the custom nature of these wheels, we will 3D print new rubber O-rings to replace the old ones. Omnidirectional wheels are ideal for this case due to their ability to provide smooth and precise multi-directional movement, which is crucial for achieving accurate and agile manoeuvres in swarm robotics applications, allowing the robots to navigate and collaborate effectively in complex environments.

The next step involves testing the motors, including their encoders, to evaluate the accuracy, power, and torque of the current setup. Our goal is to determine whether

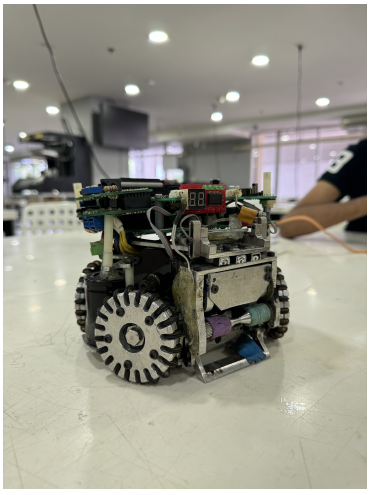

Figure 8.1: One of the robots obtained from EIC Chula, RoboCup

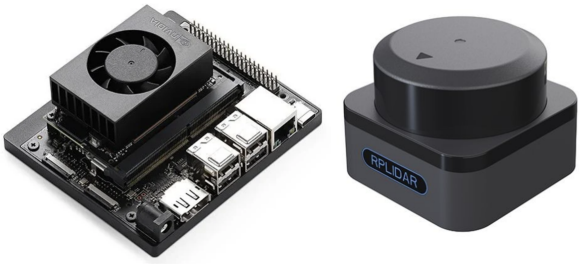

Figure 8.2: Image of the Jetson Nano Orin(left) and RPLiDAR S3 360(right)

these motors can perform adequately as servomotors for precise control and calculations. For this purpose, we will use an Arduino to control the motors during testing.

In terms of schedule, we are currently ahead of the planned timeline for the hardware development stage, giving us some flexibility for further testing and refinement.

We have encountered issues with reusing old motors and robots from RoboCup Soccer. The Hall effect sensor and the optical sensor have worn out. To address this, we designed and 3D-printed a bracket to mount the Maxon EC 45 Flat motor along with the AS5600 magnetic encoder. We are currently testing the motor to achieve closed-loop control. The motors in the robot are custom-made, and we do not know the number of pole-pairs, making calibration more challenging.

Initially, we used the SimpleFOCmini V1.0 drive board, leveraging the SimpleFOC library for testing. However, after further evaluation, we have decided to transition to the XDrive by Makerbase with ODrive firmware. This decision was made due to the superior support provided by the ODrive community and its extensive documentation, which offers better reliability and ease of integration into our project. The XDrive with ODrive firmware will replace the SimpleFOC solution and be used for achieving precise closed-loop control of the motors.

The next step, after achieving closed-loop control using XDrive with ODrive firmware, is to purchase additional motor drivers and control all four wheels. This will allow us to properly drive the X-Drive omnidirectional system using the Jetson Nano (see 8.6)), which will then be abstracted to operate under ROS.

Our hardware in Figure 8.6, will have safety features such as microphones and an emergency stop button if the robot has software malfunctions. The LED matrix will be the indicator for taskmaster. RPLiDAR 360 S3 will be used for Graph SLAM with the ROS interface package provided.

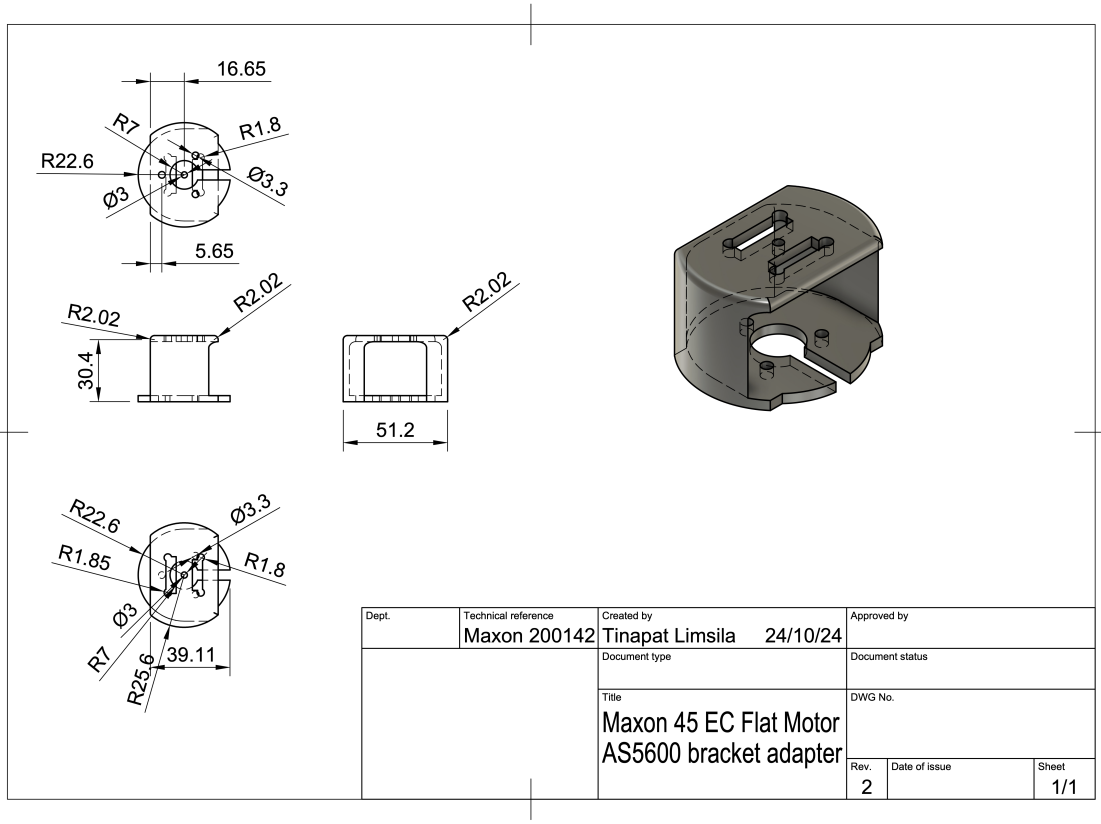

Figure 8.3: CAD drawing of the bracket for the Maxon 45 EC Flat motor and the Magnetic Encoder for AS5600

Figure 8.4: Image of the two swarm robot with end effectors with communication via WiFi

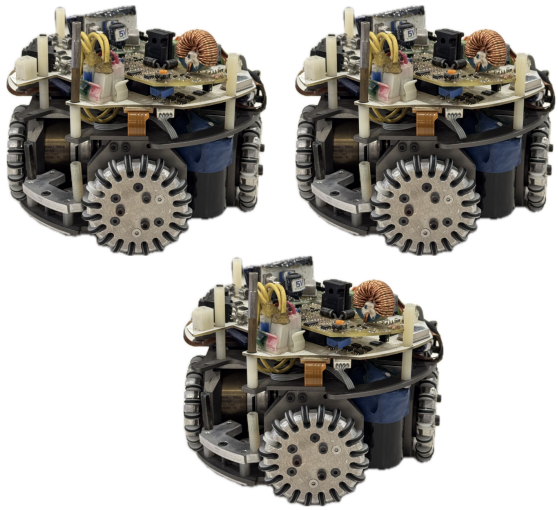

Figure 8.5: Image of the 3 acquired swarm robot pre-modification

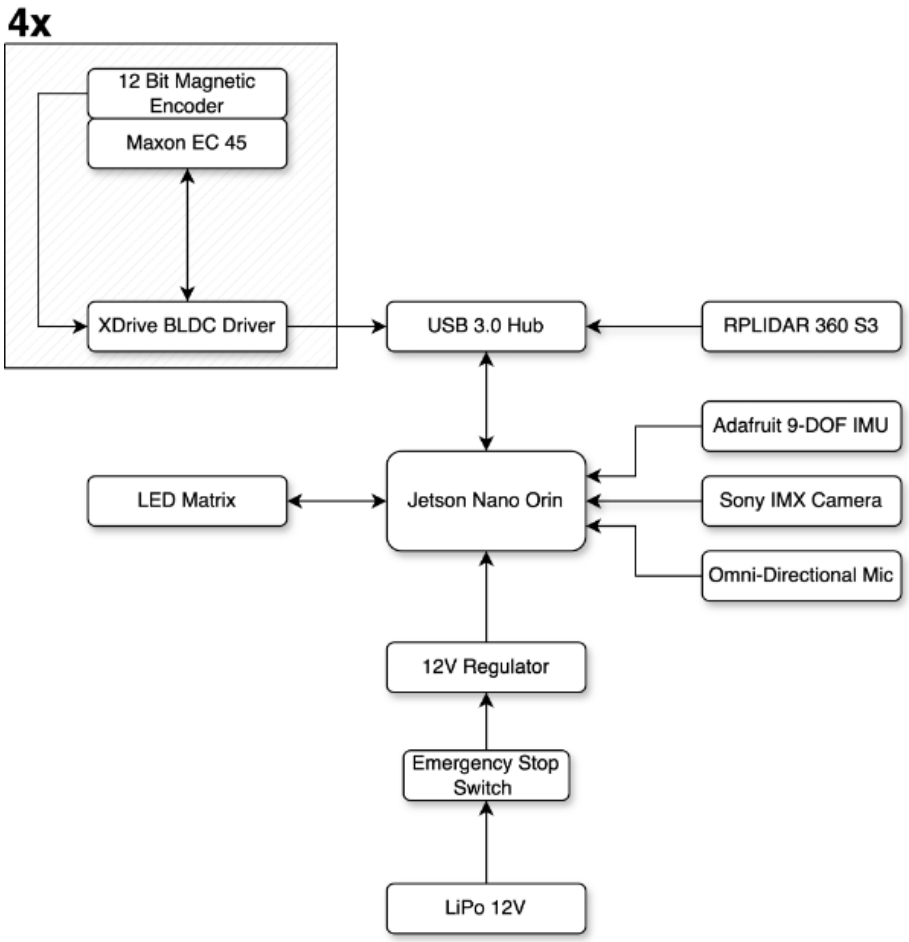

Figure 8.6: Image of a the hardware architecture and I/O of the system

Chapter 9

Conclusion

In summary, the project is making steady progress with many core modules demonstrated in simulation. While some completed components may need further refinement to accommodate dependencies from other subsystems, the current state is aligned with our objectives. As we proceed, our efforts will follow two main paths in parallel.

On one track, we will test and enhance the Graph SLAM algorithm within the Webots simulation environment. This testing phase will help evaluate the algorithm's performance under conditions closely resembling real-world scenarios and guide any necessary improvements.

Concurrently, another part of the team will shift focus towards hardware development. This involves finalising 3D designs for the robot's physical components, ensuring they meet the required specifications. Once the designs are ready, the team will move on to integrating various elements like sensors, actuators, and computational units. See Figure 9.1 for a the visualisation of the additional step to our flow

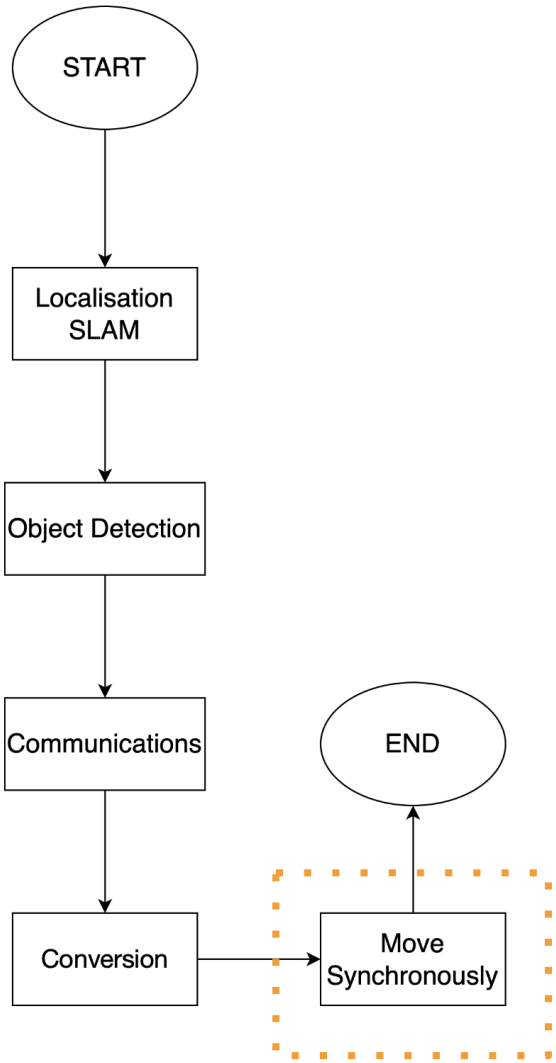

Figure 9.1: Future plans for the project

By advancing both software and hardware components simultaneously, we aim to achieve a fully integrated and functioning platform. This dual approach will ensure that our final system is both robust and reliable, setting the foundation for seamless transition from simulation to real-world implementation.

Bibliography

[1] L. Türkler, T. Akkan, and L. Ö. Akkan, "Usage of evolutionary algorithms in swarm robotics and design problems", *Sensors*, vol. 22, no. 12, p. 4437, 2022. DOI: 10.3390/s22124437.

[2] S. Das, "Bio-inspired communication strategies in swarm robotics", in *Advances in Computational Intelligence and Robotics*. 2024, pp. 77–100. DOI: 10.4018/979-8-3693-1277-3.ch006.

[3] R. Ibrahim, M. Alkilabi, A. R. H. Khayeat, and E. Tuci, "Enhancing robustness of swarm robotics systems in a perceptual discrimination task", *International Journal of Computing and Digital Systems*, vol. 15, no. 1, pp. 1213–1222, 2024. DOI: 10.12785/ijcds/160189.

[4] A. Ayari and S. Bouamama, "Evolutionary swarm robotics: A methodological approach for task and path planning", in *2023 IEEE Afro-Mediterranean Conference on Artificial Intelligence (AMCAI)*, 2023. DOI: 10.1109/amcai59331.2023.10431514.

[5] S. M. Perera, R. J. Myers, K. Sullivan, K. Byassee, H. Song, and A. Madanayake, "Integrating communication and sensor arrays to model and navigate autonomous unmanned aerial systems", *Electronics*, vol. 11, no. 19, p. 3023, 2022. DOI: 10.3390/electronics11193023.

[6] S. S. R., S. Mohanty, and D. S. Elias, "Control and coordination of a swarm of unmanned surface vehicles using deep reinforcement learning in ros", *arXiv (Cornell University)*, 2023. DOI: 10.48550/arxiv.2304.08189.

[7] Z. Wang, J. Li, J. Li, and C. Liu, "A decentralized decision-making algorithm of uav swarm with information fusion strategy", *Expert Systems With Applications*, vol. 237, p. 121 444, 2024. DOI: 10.1016/j.eswa.2023.121444.

[8] M. Yasser, O. Shalash, and O. Ismail, "Optimized decentralized swarm communication algorithms for efficient task allocation and power consumption in swarm robotics", *Robotics*, vol. 13, no. 5, p. 66, 2024. DOI: 10.3390/robotics13050066.

[9] N. Nedjah, L. M. Ribeiro, and L. De Macedo Mourelle, "Communication optimization for efficient dynamic task allocation in swarm robotics", *Applied Soft Computing*, vol. 105, p. 107 297, 2021. DOI: 10.1016/j.asoc.2021.107297.

[10] E. Beck, B. Shin, S. Wang, T. Wiedemann, D. Shutin, and A. Dekorsy, "Swarm exploration and communications: A first step towards mutually-aware integration by probabilistic learning", *Electronics*, vol. 12, no. 8, p. 1908, 2023. DOI: 10.3390/electronics12081908.

[11] S. Zhang, E. Staudinger, R. Pohlmann, and A. Dammann, "Cooperative communication, localization, sensing and control for autonomous robotic networks", in *2021 IEEE International Conference on Autonomous Systems (ICAS)*, 2021. DOI: 10.1109/icas49788.2021.9551201.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[13] V. Kumar, N. Michael, and J. P. How, "Multi-robot coordination in warehousing", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 232–239.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", in *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[15] W. Chen, Y. Li, Z. Tian, and F. Zhang, "2d and 3d object detection algorithms from images: A survey", *Array*, vol. 19, p. 100 305, 2023. DOI: 10.1016/j.array.2023.100305.

[16] B. Karin, A. Gonzalez, and X. Zhou, "A comparative analysis of stereo camera systems for depth sensing in robotics", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 1234–1241.

[17] S. Wang, L. Zhang, Z. Zhang, and S. Zhang, "Radar and camera fusion for robust object detection and tracking", *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 5, pp. 2035–2047, 2020. DOI: 10.1109/TITS.2019.2901681.

[18] K. A. Tychola, I. Tsimperidis, and G. Papakostas, "On 3d reconstruction using rgb-d cameras", *Digital*, vol. 2, no. 3, pp. 401–423, 2022. DOI: 10.3390/digital2030022.

[19] N. V. K. Medathati, H. Neumann, G. S. Masson, and P. Kornprobst, "Bio-inspired computer vision: Towards a synergistic approach of artificial and biological vision", Inria Sophia Antipolis, Tech. Rep. 8698, 2016, ffhal-01131645v3f, p. 71.

[20] Y. Li, J. Su, L. Liu, and P. Liu, "Object detection based on the fusion of sparse lidar point cloud and dense stereo pseudo point cloud", in *IEEE*, Jan. 2024. DOI: 10.1109/NNICE61279.2024.10498214.

[21] T. Eppenberger, G. Cesari, M. Dymczyk, and R. Dube, "Leveraging stereo-camera data for real-time dynamic obstacle detection and tracking", in *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. DOI: 10.1109/IROS45743.2020.9340699.

[22] J. Rodriguez, "A comparison of an rgb-d camera's performance and a stereo camera in relation to object recognition and spatial position determination", *ELCVIA Electronic Letters on Computer Vision and Image Analysis*, vol. 20, no. 1, p. 16, 2021. DOI: 10.5565/rev/elcvia.1238.

[23] Y. Wang, C. Wang, P. Long, Y. Gu, and W. Li, "Recent advances in 3d object detection based on rgb-d: A survey", *Displays*, vol. 70, p. 102 077, 2021. DOI: 10.1016/j.displa.2021.102077.

[24] E. Arican and T. Aydin, "Object detection with rgb-d data using depth oriented gradients", in *Proceedings of the International Conference on Engineering and Natural Sciences (ICENS)*, Hungary - Budapest, 2017.

[25] A. Paigwar, D. Sierra-Gonzalez, Ö. Erkent, and C. Laugier, "Frustum-pointpillars: A multi-stage approach for 3d object detection using rgb camera and lidar", in *Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, IEEE, 2021, pp. 2926–2933. DOI: 10.1109/ICCVW54120.2021.00327.

[26] C. Tao, H. Zhang, J. Liu, and J. Li, "F-pvnet: Frustum-level 3-d object detection on point–voxel feature representation for autonomous driving", *IEEE Internet of Things Journal*, vol. 10, no. 9, pp. 8031–8045, 2023. DOI: 10.1109/JIOT.2022.3231369.

[27] Z. Ren and E. B. Sudderth, "3d object detection with latent support surfaces", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2018)*, Salt Lake City, UT, United States: IEEE Computer Society, 2018, pp. 104–112. DOI: 10.1109/CVPR.2018.00104.

[28] Z. Zhang *et al.*, "H3dnet: 3d object detection using hybrid geometric primitives", in *European Conference on Computer Vision*, Springer, 2020. DOI: 10.48550/arXiv.2006.05682.

[29] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA: IEEE Computer Society, 2005, 886–893, Vol. 1. DOI: 10.1109/CVPR.2005.177.

[30] P. Viola and M. Jones, "Robust real-time face detection", in *Proceedings of the Eighth IEEE International Conference on Computer Vision (ICCV 2001)*, Vancouver, BC, Canada: IEEE Computer Society, 2001, p. 747. DOI: 10.1109/ICCV.2001.937709.

[31] X. Zhou and G. Lin, "Review of target detection algorithms", *Frontiers in Computing and Intelligent Systems*, vol. 4, no. 3, pp. 17–19, 2023. DOI: 10.54097/fcis.v4i3.10736.

[32] B. Karbouj, "Comparative performance evaluation of one-stage and two-stage object detectors for screw head detection and classification in disassembly processes", *Procedia CIRP*, vol. 122, 2024, Conference: 31st CIRP Conference on Life Cycle Engineering (LCE 2024), Turin. License: CC BY-NC-ND 4.0. Lab: Bsher Karbouj's Lab. DOI: 10.1016/j.procir.2024.01.077.

[33] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector", in *Computer Vision – ECCV 2016. Lecture Notes in Computer Science*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9905, Springer, Cham, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2.

[34] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936–944. DOI: 10.1109/CVPR.2017.106.

[35] R. Girshick, "Fast r-cnn", in *2015 IEEE International Conference on Computer Vision*, IEEE, 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[36] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", in *Advances in Neural Information Processing Systems (NIPS)*, 2015. DOI: 10.48550/arXiv.1506.01497.

[37] V. K. Kaliappan, M. S. Shanmugasundaram, L. Ravikumar, and G. B. Hiremath, "Performance analysis of yolov8, rcnn, and ssd object detection models for precision poultry farming management", in *2023 IEEE 3rd International Conference on Applied Electromagnetics, Signal Processing, & Communication (AESPC)*, Bhubaneswar, India: IEEE, 2023, pp. 1–6. DOI: 10.1109/AESPC59761.2023.10389906.

[38] V. K. Kaliappan, R. Thangaraj, P. Pandiyan, K. Mohanasundaram, S. Anandamurugan, and D. Min, "Real-time face mask position recognition system using yolo models for preventing covid-19 disease spread in public places", *International Journal of Adaptive and Autonomous Systems*, vol. 34, no. 1, pp. 73–82, 2023. DOI: 10.1504/IJAHUC.2023.128499.

[39] A. Barbadekar, S. Raut, R. Gaikwad, T. Gadad, S. Ghulaxe, and N. Dhabale, "Exploring enhanced localization techniques: Lidar-slam for mobile robots", in *2023 Global Conference on Internet of Things and Cyber-Physical Systems (GCITC)*, 2023. DOI: 10.1109/gcitc60406.2023.10426008.

[40] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i", *IEEE Robotics and Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006. DOI: 10.1109/mra.2006.1638022.

[41] T. Chong, X. Tang, C. Leng, M. Yogeswaran, O. Ng, and Y. Chong, "Sensor technologies and simultaneous localization and mapping (slam)", *Procedia Computer Science*, vol. 76, pp. 174–179, 2015. DOI: 10.1016/j.procs.2015.12.336.

[42] B. Udugama, "Evolution of slam: Toward the robust-perception of autonomy", in *arXiv preprint*, Available at: https://doi.org/10.48550/arXiv.2302.06365, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2302.06365.

[43] A. Sahoo, S. K. Dwivedy, and P. Robi, "Advancements in the field of autonomous underwater vehicles", in *Ocean Engineering*, vol. 181, Elsevier, 2019, pp. 145–160. DOI: 10.1016/j.oceaneng.2019.04.011. [Online]. Available: https://doi.org/10.1016/j.oceaneng.2019.04.011.

[44] Y. Bisheng, L. Fuxun, and H. Ronggang, "Progress, challenges, and perspectives of 3d lidar point cloud processing", *DOAJ (Directory of Open Access Journals)*, 2017. DOI: 10.11947/j.agcs.2017.20170351.

[45] C. Cadena, L. Carlone, H. Carrillo, *et al.*, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age", *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016. DOI: 10.1109/tro.2016.2624754.

[46] B. Langmann, K. Hartmann, and O. Loffeld, "Depth camera technology comparison and performance evaluation", in *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP)*, Barcelona, Spain: SciTePress, 2012, pp. 438–444. [Online]. Available: https://doi.org/10.5220/0003778304380444.

[47] C. Peng, "Depth camera point cloud sharpening", in *Proceedings of the IEEE International Conference on Consumer Electronics (ICCE)*, Taipei, Taiwan: IEEE, 2023, pp. 102–106. DOI: 10.1109/icce-taiwan58799.2023.10226686.

[48] K. Huang, S. Zhang, J. Zhang, and D. Tao, "Event-based simultaneous localization and mapping: A comprehensive survey", *arXiv*, 2023. DOI: 10.48550/arxiv.2304.09793.

[49] R. Beņkis, E. Grabs, T. Chen, *et al.*, "A survey and practical application of slam algorithms", in *2024 Progress In Electromagnetics Research Symposium (PIERS)*, 2024, pp. 1–10. DOI: 10.1109/piers62282.2024.10618240.

[50] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: A versatile and accurate monocular slam system", *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015. DOI: 10.1109/tro.2015.2463671.

[51] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 611–625, 2018. DOI: 10.1109/tpami.2017.2658577.

[52] X. Gu, X. Wang, and Y. Guo, "A review of research on point cloud registration methods", *IOP Conference Series Materials Science and Engineering*, vol. 782, no. 2, p. 022 070, 2020. DOI: 10.1088/1757-899x/782/2/022070.

[53] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time", in *Proceedings of Robotics: Science and Systems (RSS)*, 2014. [Online]. Available: https://doi.org/10.15607/rss.2014.x.007.

[54] W. Xu and F. Zhang, "Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter", *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3317–3324, 2021. DOI: 10.1109/lra.2021.3064227.

[55] W. Chen, X. Wang, S. Gao, *et al.*, "Overview of multi-robot collaborative slam from the perspective of data fusion", *Machines*, vol. 11, no. 6, p. 653, 2023. DOI: 10.3390/machines11060653.

[56] X. Liu, J. Lei, A. Prabhu, *et al.*, "Slideslam: Sparse, lightweight, decentralized metric-semantic slam for multi-robot navigation", *arXiv*, 2024. [Online]. Available: https://doi.org/10.48550/arxiv.2406.17249.

[57] P. Lajoie and G. Beltrame, "Swarm-slam: Sparse decentralized collaborative simultaneous localization and mapping framework for multi-robot systems", *IEEE Robotics and Automation Letters*, vol. 9, no. 1, pp. 475–482, 2024. DOI: 10.1109/lra.2023.3333742.

[58] A. Mehta and A. Modi, "Robust sliding mode protocols for formation of quadcopter swarm", 2024.

[59] F. Martinez and A. Rendon, "A swarm-based flocking control algorithm for exploration and coverage of unknown environments", 2023.

[60] E. Bonabeau, A. Sobkowski, G. Theraulaz, and J. L. Deneubourg, "Adaptive task allocation inspired by a model of division of labour in social insects", in *Biocomputing and Emergent Computation: Proceedings of BCEC97*, London, UK: World Scientific, 1997, pp. 36–45.

[61] M. Brambilla, C. Pinciroli, M. Birattari, and M. Dorigo, "Property-driven design for swarm robotics", in *Proceedings of 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Richland: IFAAMAS, 2012, pp. 139–146.

[62] D. St-Onge, J. Le Ny, and G. Beltrame, "Swarm-slam: Sparse decentralised collaborative simultaneous localization and mapping framework for multi-robot systems", *IEEE Robotics and Automation Letters*, vol. 8, no. 3, pp. 1720–1727, 2023.

[63] L. L. Bayindir and E. Şahin, "A review of studies in swarm robotics", *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 15, no. 2, pp. 115–126, 2007. [Online]. Available: https://journals.tubitak.gov.tr/elektrik/vol15/iss2/2.

[64] A. R. Cheraghi, S. Shahzad, and K. Graffi, "Past, present, and future of swarm robotics", in *Lecture Notes in Networks and Systems*. 2021, pp. 190–233. DOI: 10.1007/978-3-030-82199-9_13.

[65] S. E. Ghazouali, Y. Mhirit, A. Oukhrid, U. Michelucci, and H. Nouira, "Fusionvision: A comprehensive approach of 3d object reconstruction and segmentation from rgb-d cameras using yolo and fast segment anything", *Sensors*, vol. 24, no. 9, p. 2889, 2024. DOI: 10.3390/s24092889.

[66] M. Kegeleirs, G. Grisetti, and M. Birattari, "Swarm slam: Challenges and perspectives", *Frontiers in Robotics and AI*, vol. 8, 2021. DOI: 10.3389/frobt.2021.618268.

[67] Y. Koifman, A. Barel, and A. M. Bruckstein, "Distributed and decentralized control and task allocation for flexible swarms", *arXiv*, 2024. DOI: 10.48550/arxiv.2405.13941.

[68] N. Nedjah and L. J. Silva, "Review of methodologies and tasks in swarm robotics towards standardization", *Swarm and Evolutionary Computation*, vol. 50, p. 100 565, 2019. DOI: 10.1016/j.swevo.2019.100565.

[69] J. Qian, Y. Lv, Y. Gao, and J. Wang, "Frustum-ldgcnn: A high efficient 3d object detection network with frustum proposal", in *2023 IEEE International Conference on Automation Science and Engineering (CASE)*, 2023. DOI: 10.1109/case56687.2023.10260350.

[70] E. Şahin, "Swarm robotics: From sources of inspiration to domains of application", in *Lecture Notes in Computer Science*, 2005, pp. 10–20. DOI: 10.1007/978-3-540-30552-1_2. [Online]. Available: https://doi.org/10.1007/978-3-540-30552-1_2.

[71] A. Bicchi and V. Kumar, "Robotic grasping and contact: A review", in *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, 2000.

[72] L. E. Parker, "Alliance: An architecture for fault-tolerant multirobot cooperation", *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 220–240, 1998.

[73] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement", *arXiv preprint arXiv:1804.02767*, 2018.

[74] S. Thrun, "A probabilistic approach to concurrent mapping and localization for mobile robots", *Autonomous Robots*, vol. 5, no. 3, pp. 253–271, 2003.

[75] G. Beni and J. Wang, "Swarm intelligence in cellular robotic systems", in *Proceedings of NATO Advanced Workshop on Robots and Biological Systems*, 1989.