The University of Melbourne - School of Electrical Engineering

Autonomous Systems Clinic - Final Report

Tuan Khoi Nguyen (1025294) Ze Kai Cheong (1182444) Nguyen Hong Phu Trinh (1086263)
Group F (Crippling Regression)

June 13th, 2023

THE UNIVERSITY OF MELBOURNE

Contents

Introduction

This project report will describe in details the process of designing an autonomous robot that performs multiple tasks without human intervention. This includes the reasoning for the application of each subsystem of the robot, the motivation behind the design decisions, and finally, experimentation process, from small-scale testing to the official workshop demonstration, is done to validate all choices along with the motivations. Additionally, logistic documents throughout the project will also be included in an Appendix. Through this report, the procedures and challenges involved in creating an effective system for autonomous robots, in both technical and teamwork aspects, will be displayed.

1 Context

This section introduces the project specifications, consisting of scopes expected in the outcome, testing course used for the project, and the provided robot platform to implement the autonomous system.

1.1 Project Description The aim of the project is to develop an autonomous system using an existing robot platform for the client (Unimelb Grounds Keeping Team). Without human intervention, the differential-drive wheeled robot has to traverse University of Melbourne gardens/grounds to look for plants in the area, and is expected to take at least 1 picture of each plant for inspection. The main outcomes of interest when evaluating the quality of the system's performance are images taken and navigation capability.

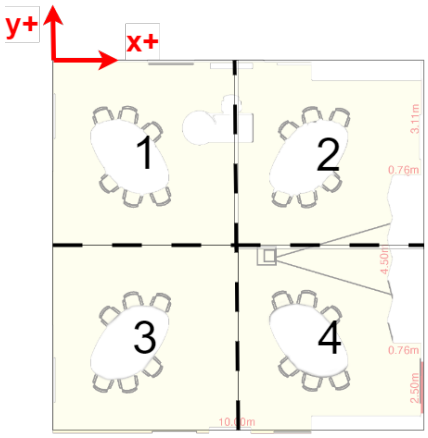


Figure 1: Map and division of EDS10 into quadrants with the origin defined in the top left corner

Course The designated course where the robot is expected to operate for a proof-of-concept demonstration for the client is a $10m \times 10m$ indoor classroom workshop space: EDS10 (Figure 1). The course has four fixed tables in each of the four quadrants, as well as chairs that are located at arbitrary locations but mostly near the tables, and potentially humans moving around. A third person view of the room from the perspective of the robot is shown in Figure 2. Further map and overview images of the course as one half of a larger space (EDS9 and EDS10) are shown in Appendix A.

Figure 2: Third person view from the robot, indicating its size with respect to the designated course

Robot The robot platform operates on the NVIDIA Jetson Xavier NX single-board computer (SBC), with 8 gigabytes of memory, 2-to-6-core CPU with maximum frequency of 1.9GHz and Volta GPU [1]. It has a range of sensors and actuators, including a 2D LiDAR, web camera mounted on two servo motors, and two quadrature-encoded DC motors coupled directly to wheels. [1]. All these components communicate using the Robot Operating System (ROS) middleware.

1.2 Aim This report will focus on the process of software development, specifically development of algorithms and communication pipeline for the robot in ROS. This consists of ideation and reasoning for the system design and validation testing process to ensure the feasibility of the ideas. This report will also mention attachments of extra hardware or equipment to the robot to expand functionality.

2 Requirements Formulation and System Architecture

To develop a design that addresses the client's needs, this section will introduce the baseline and extra tasks of the project and the metrics or success criteria to evaluate or validate them. The system architecture design and subsystem breakdown is also covered.

2.1 Tasks The requirements of the project can be divided down to a list of tasks that the robot can perform. This list

[1] Further details can be viewed in ASC documentation page: `https://people.eng.unimelb.edu.au/pbeuchat/asclinic/hardware/bom.html`

can be divided into 2 categories: baseline and extra.

2.1.1 Baseline Tasks The tasks in this category are strictly required to be achieved to meet the minimum requirements of the project. They directly correspond to the statements in the project description.

B1 - Course Traversal The robot has to be able to reach all accessible plants in the course.

B2 - Obstacle avoidance The robot has to be able to avoid fixed obstacles while navigating the course, assuming their locations are known beforehand.

B3 - Plant Detection & Capture The robot needs to be able to identify a plant in front of it, and take at least one picture upon detection.

2.1.2 Extra Tasks The tasks in this category are extensions that enhance the user experience and improve the appeal of the system.

E1 - User Interface (UI) An interface displaying the status and variables of the robot would be useful when the user wants to keep track of the robot's activity. It also helps in development, where debugging can be enhanced by showing the robot's status when a problem occurs.

E2 - Dynamic Obstacle Avoidance Being able to expand the obstacle avoidance functionality for items at unknown locations or dynamic objects such as a moving human makes the robot more applicable to the real-world situation, where not all obstacles are static and lie in a known fixed location.

E3 - Plant Capture Enhancement As image quality is one of the main assessments for functionality, it is essential to improve the photo-taking mechanism to provide better pictures. Possible examples include aligning camera frame to the plant, image stabilization, or plant approaching for closer observation.

2.2 Performance Metrics

After listing out the tasks that need to be done, metrics are needed to ensure that these objectives are fulfilled under the given project conditions. There are tradeoffs between these metrics and the system needs to balance them.

2.2.1 Autonomy The robot needs to be able to complete all tasks without external intervention. The robot processor is expected to perform all operations on its own after the starting command.

2.2.2 Speed In order to perform well and increase desirability, it is expected that the robot perform and complete the tasks in a timely manner.

2.2.3 Robustness The robot has to be able to handle the variability of conditions in the given space and perform as expected reliably and consistently. This is done by ensuring that subsystems can guarantee certain standards of performance which are maintained when the subsystems are integrated.

Subsystem	Task Contribution
Path Planning	B1, B2, E2
Path Following	B1, B2, E2, E3
Camera Tracking	B2, B3, E2, E3
Stretch Functionalities	E1, E2

Table 1: Summarizing table of the subsystems

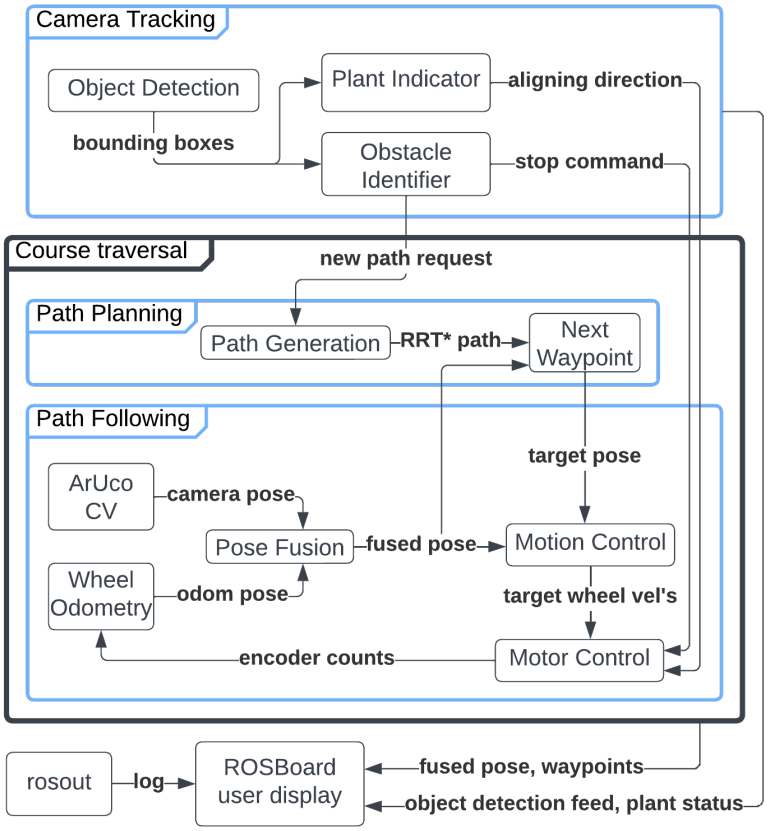


Figure 3: High level system architecture

2.3 System Architecture Overview

With the tasks and performance metrics listed out, the system can now be divided into suitable subsystems, where each will contribute to the implementation of one or more tasks given in 2.1. A comprehensive summary of the tasks that each subsystem can handle is shown in Table 1. Each subsystem may consist of multiple components, which are referred to as functionalities

throughout this report. The high level system architecture composed of these functionalities and the information flow between them is shown in Figure 3. The subsystems and their functionalities are further explained below.

2.3.1 Path Planning Given a known map of the course, a good path planning system is one that generates a path that is reasonably short and can scale well with any size map to accommodate larger gardens or grounds of the Unimelb campus. It should guarantee a path to any accessible goal location and ensures that the path avoids known static obstacles like landmarks and structures such that any part of the robot will not collide with them. The RRT* path planning algorithm is used to generate a set of waypoints to give to the Path Following subsystem one-by-one as a target pose. Extensions to the system include dynamic path planning to avoid obstacles spontaneously and reroute its path even if the robot is picked up and moved to a different location.

2.3.2 Path Following Given a planned path, this subsystem enables the robot to traverse it. It is vital that the robot track the planned path closely to ensure that the guarantees created by 2.3.1 are maintained. This requires an estimate of its current pose (localisation) using wheel odometry together with ArUco marker detection, a function to determine the required wheel velocities given its current pose and target pose (outer-loop motion control), and control over the wheel velocities (inner-loop motor control).

2.3.3 Camera Tracking This subsystem takes care of all functionalities associated with camera input processing. Its primary function lies in detection, encompassing the recognition of ARuCo markers and obstacles, and extracting vital information from these detections, such as localized positions or object sizes. Moreover, with the specific requirement for plant identification, this subsystem plays a crucial role in recognizing plants, making it an important component of the overall system. By developing the camera tracking subsystem, we can ensure accurate and reliable detection of objects, thereby fulfilling objectives set forth in project requirements.

2.3.4 Stretch Functionalities These functionalities are additional components that are not compulsory for baseline tasks, but enhance the overall performance and capabilities of the autonomous system in both operation and user experience. These include a user display interface using ROSboard to track the robot's location on a map of the course and stream the camera and object detection video feed. They also include extra functionalities that were not integrated or fully implemented in the demonstration due to the time constraints of the project (e.g. Hector SLAM package with 2D LiDAR).

3 Subsystem Design

In this section, the design of each subsystem and the functionalities they are composed of will be further described. Initial preliminary testing on separate functionalities will be presented where possible, and in-depth analysis of the evaluations that led to the initial component structure will be made. Overall, the reasoning behind design choices such as algorithm or tools of each functionality will be listed.

3.1 Path Planning Path planning is in charge of the overall planning of the paths in which the robot needs to take to traverse throughout the room. Successful implementation and working of this subsystem ensure completion of all baseline tasks and possibly can be extended for Tasks E2. Path planning involves two key tasks which are first, representing the environment specification as a graph or similar structure and only then will a traversing algorithm be used to generate a sets of waypoint from initial location to destination.

Path Generation

- **Function:** Plan a path from initial position to destination without collision and within appropriate time-frame
- **Input:** Map specification, current robot pose and destination location
- **Output:** Set of Waypoint in order of traversal from initial location to destination
- **Dependency:** Target and Robot Location

3.1.1 Static Mapping The first key tasks of path planning is to translate the map specification into an appropriate format which are then used as input to path planning algorithm. The provided map and nominal dimensions were converted to an Occupancy Grid cell format, using the pixels and nominal dimension, where each grid cell are of size $5 \times 5cm^2$. Each instances of the Occupancy Grid are represented as (x, y, I), where (x, y) indicates the locations of the grid cells relative to the origin point on the map. I elements indicate the probability or likelihood of the cells being obstructed or contains objects within it, and can be represented mathematically as $I \subseteq \{1, 100, -1\}$. Value of 1 indicates cell not being obstructed, 100 indicating cell obstruction and -1 means unknown. For simplicity, it was assumed that all obstructions (no-go zones) in the course are known, leading to $I \subseteq \{1, 100\}$.

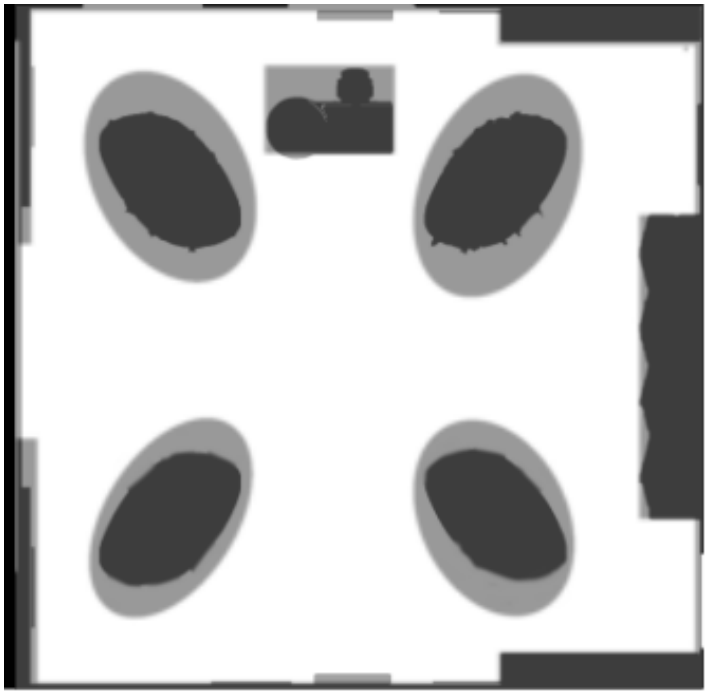


Figure 4: Extra margins added (gray) for EDS10 area

Safe margin The no-go zones in the course are defined to be the fixed tables in EDS10, cupboards and walls. Observation on the mapping shows that the floorplan (Appendix Figure 29), which the mapping is based on, may still have uncertainty as it is not guaranteed that the tables' position are the same (they can still be moved under large forces). Furthermore, there can still be potential uncertainty given from trajectory following system. As a result, a variable amount of margin is expanded into the no-go zones, based on the confidence of each section in the map, which is subjectively assessed by the developer. Figure 4 displays how this is done for EDS10 area.

The final generated map for the whole course (Figure 5) will be of black and white, where black indicates an exclusion zone and white a free zone.

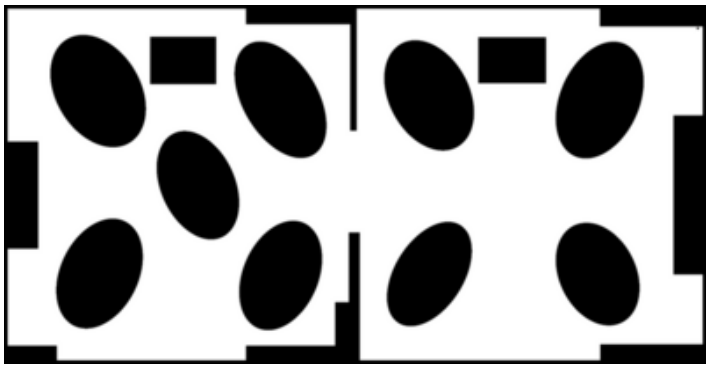


Figure 5: Generated Map for Occupancy Grid

3.1.2 Path Generation Methods In this part, a number of candidates for path planning and generation will be considered and evaluated.

Voronoi Diagram Voronoi diagram generate a set of points equidistant from two nearest exclusion zone across the entire space manifolds of the map [2]. This form a spanning trees whilst optimally maximises distance away from obstacles, therefore, can guaranteed that robot are able avoid obstacle and having a higher chance of task completion. However, this methods will impact the speed of traversal as the spanning trees will not indicates the most optimal path traversal throughout the map space. This process is done offline and is written to a file that is stored on the Jetson and to read at run-time. During run-time a graph traversal algorithm like Dijkstra or A* will be used onto the spanning tree to compute a path from start to destination when requested.

RRT* and RRT Rapidly-exploring random trees (RRT) is a path planning technique that deals with both creating a graphs and compute a path [2]. Sampling points are randomly generated on the graph and connect closest to the available node. Each new connected vertex and its chaining neighbour nodes will be check to ensure that it lies outside of an obstacle to only guaranteed that graph will always be outside of exclusion zone. RRT will only terminated when a nodes is generated within the goal location or a number of limit is hit.

RRT* extends further from RRT where it is an optimised version of RRT as it try to deliver the shorter path by keeping the cost of distance each vertex has traveled and when a closer vertex to a parent node is found, the shorter vertex will replaced the older one creating a more optimal path then RRT [3]. RRT* will terminate in the same way as RRT, but with more optimality. It is to be noted that this is the not the shortest possible path to the goal, as it will significantly increase the computational cost and performance of the path planning node if the algorithm were to be implemented with trying to the find shortest path. Furthermore, it is to be noted that due to the working principle of RRT and RRT*, different path will be generated every single time the node is run even with the same pair of starting/end location.

3.1.3 Next Waypoint Next waypoint subsystem receive message from the path generation and robot current pose to output to determine the waypoint in which the robot should be send to so that it incrementally approaches the required destination.

Next Waypoint

- **Function:** Give next waypoint to the robot path tracking so the robot approach the destination
- **Input:** Set of generated waypoints $(x_1, y_1), \ldots, (x_n, y_n)$ for path traversal, current robot pose and destination location
- **Output:** Next waypoint (x, y) for the robot to move along the path
- **Dependency:** Robot Location, Path planning

3.1.4 Waypoint Thresholding A threshold boundary was used to determine whether the current waypoint is reached and for the next waypoint to be send to the path following node. The threshold boundary used the current robot pose to compare against the targeted waypoint. 2 possible thresholding methods will be evaluated in this section: circular boundary and half-plane threshold.

Circular Boundary Figure 6 present the circular boundary that is implement in the system with the following equation:

$$P = \begin{bmatrix} X \\ Y \end{bmatrix}, D = \begin{bmatrix} X_D \\ Y_D \end{bmatrix}$$
$$H = \sqrt{(X_D - X)^2 + (Y_D - Y)^2}$$

Where:

- P: Current Pose Location
- D: Current Target Location
- H: Euclidean Distance between Robot and Target
- R: Threshold Value

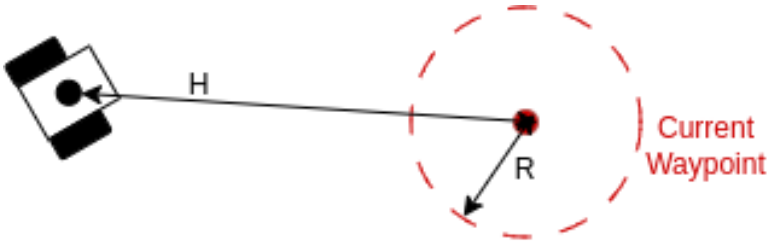


Figure 6: Circular Threshold

If the threshold value R is higher than Euclidean Distance H, it is determined that the robot is within the accepted vicinity of the current waypoint. Therefore, the next waypoint sent to the Path Following node. If it is not within vicinity of the current waypoint, it will be assumed that the robot is still travelling to the waypoint. Threshold value R is an adjustable parameter that can be tuned to ensure sufficient performance, as well as having acceptable overall time of path completion. Having a small threshold value R would ensure that robot location is close to the targeted waypoint hence, the actual robot path will be similar to that of the generated path. However, this would mean that it will increase the time of path completion, as the robot have to spend more time adjusting to perfectly track the generated path, rather than traversing through the path.

Half-Plane Threshold Figure 7 demonstrate a different thresholding technique that was implemented. This half-plane threshold technique tracks the difference in the robot position from the generated straight line path between previous waypoint and current waypoint through X and Y components. Here, both the robot's X and Y values have to be inside the set threshold to be considered a successful waypoint completion. This separation of X-Y values decreases the boundary area compared to the circular threshold, by preventing overshoot behaviour to happen, as the robot's final location for the particular way will never move beyond the straight line threshold that was set. This would allow better waypoint tracking performance especially in tracking in the heading of the robot.

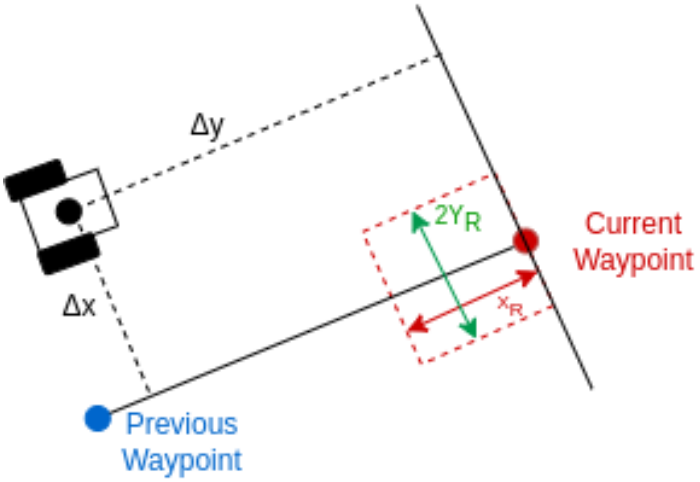


Figure 7: Demonstration of Half Plane Threshold

3.1.5 Design Decision In the final demonstration of the robot, design decision were made to select RRT* and Circular Boundary for Path Generation and Waypoint Thresholding respectively.

RRT* for Path Planning For Path Generation, RRT* were selected because of the versatility and the scalability compared to Voronoi. Even though Voronoi was easier to implement, the fact that the spanning tree has to be computed offline before run-time decreases the feasibility of the technique during real-world usage, as well as limiting the ability of the robot development to have extra capabilities. One example would be dynamic path planning as well as updating dynamic obstacles in real-time usage. Both of this could be added onto RRT* path planning with minor modification. Furthermore, RRT* have better performance when compared to Voronoi for larger maps space [2]. It is to be noted that both of the path generation techniques does not guarantee the shortest path through the entire graph, as Djikstra or A* will be performed on the computed spanning tree from Voronoi which does not neccsarily covers all points within the map. RRT* works in similar ways, however, with having better node computation, RRT* will optimised towards creating the optimal path.

Circular Boundary for Waypoint Thresholding For Waypoint Thresholding, circular boundary was selected due to ease of implementation over the Half-Plane technique since the Path following algorithm implemented ensures that the robot does not overshoot the targeted waypoint by slowing down the robot as it get closer to the waypoint. Therefore, by only using circular boundary check with the knowledge that robot does not overshoot, the circular threshold boundary is much smaller than that of the half-plane, hence is expected to have a better tracking performance by ensuring that the robot is closer to the current waypoint.

3.2 Path Following

Path following is in charge of moving the robot to the next waypoint (that is in a sequence of target waypoints along the path). As previously mentioned, to do this, it finds where it is by fusing pose estimates from wheel odometry and computer vision. Then it finds the appropriate wheel velocities required to move to the target location from its current location, and sends the commands to the Motor Control to move the wheels.

3.2.1 Wheel Odometry

A differential-drive kinematic model was used to derive the odometry update equations (Appendix B) to determine the incremental change in odometry pose over a small time interval $\Delta t = 0.1s$ and accumulate them over successive periods from the starting position to determine the absolute pose of the robot in the room.

Wheel Odometry

- **Function:** Calculate the change in pose with a differential-drive robot kinematic model
- **Input:** Encoder counts
- **Output:** Odometry pose
- **Dependency:** Wheel encoders

The update equations are summarised here:

$$x_{p,t} = x_{p,t-1} + \Delta s_t \cos(\phi_{t-1} + \frac{1}{2}\Delta\phi_t)$$
$$y_{p,t} = y_{p,t-1} + \Delta s_t \sin(\phi_{t-1} + \frac{1}{2}\Delta\phi_t)$$
$$\phi_t = \phi_{t-1} + \Delta\phi_t$$

Where:

$$\Delta s_t = \frac{r\Delta\theta_{l,t}}{2} + \frac{r\Delta\theta_{r,t}}{2}$$
$$\Delta\phi_t = \frac{r\Delta\theta_{r,t}}{2b} + \frac{r\Delta\theta_{l,t}}{2b}$$
$$\Delta\theta_t = \frac{2\pi}{1120}\Delta\theta_{counts}$$

It is given the measurements of the wheelbase $2b = 0.2157m$, wheel radius $r = 0.07176m$ and the number of encoder counts $\Delta\theta_{counts}$ of each wheel in Δt (Details, assumptions and reasoning were discussed in the Wheel Odometry Preliminary Report [4]).

3.2.2 Computer Vision (CV) using ArUco Markers .

ArUco CV

- **Function:** Determine the current pose of the robot
- **Input:** Pairs of ID-Location of ArUco markers in xyz World Frame
- **Output:** Pose of Robot in xyz World Frame
- **Dependency:** Pose of ArUco IDs, Camera

ChArUco Board Calibration Calibration process used for ArUco Marker detection of the camera is through utilising ChArUco Board, a hybrid checkerboard with alternating ArUco Markers of different IDs. Comparing to checkerboard, this board provides a more robust handle of occlusion or lighting problems in calibration, which works by extrapolating missing patterns from detected AruCo markers in pattern [5].

Multiple images of ChArUco Board will be taken from different perspectives. For each taken images with known ground truth for size of grid cells and pattern, the calibration algorithm will be able to accurately detect the markers corners. By applying the least square algorithm, the algorithm will be able to estimate the intrinsic parameters (Camera Matrix and Distortion Coefficient) for 1 image. Combining multiple images dataset under different conditions (lighting, occlusion, partial view), a final calibration parameters was found for pose estimation use. Furthermore, during calibration, there was an intermediate stage of image filtering to ensure that only non-blur images with full and clear details are used as input into the algorithm, further enhancing the calibration quality.

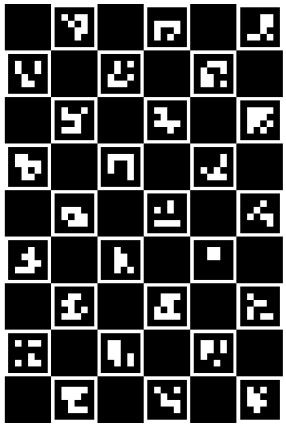


Figure 8: 9×6 ChArUco Board generated for use during calibration

The following final intrinsic parameters (Camera Matrix $\mathbf{A}$ and Distortion Coefficient $\mathbf{D}$) are calculated after calibration was done on the 9×6 board given in Figure 8:

$$\mathbf{A} = \begin{bmatrix} 1419.74 & 0 & 963.03 \\ 0 & 1415.00 & 534.30 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$\mathbf{D} = \begin{bmatrix} 26.04 \\ 91.58 \\ -0.004 \\ -0.002 \\ 4.77 \\ 26.08 \\ 89.91 \\ 9.69 \end{bmatrix} \tag{2}$$

Implementation Method The output from ArUco Marker detection gives relative distance between Marker and Camera in the Camera Frame; and to give the absolute position of the robot in the global frame will require frame transformation. Firstly, the ArUco Marker position ($Pos_{Aruco} = [X, Y, \theta]^T$) with respect to the global frame will need to be recorded. Using the ArUco Marker positions, we can build a transformation matrix between each associated marker to the global frame. Relative distance measurement between robot and marker in the camera frame will be transformed to the ArUco Marker frame, then transform once more to the global frame. Due to most of the frame transformation being constant, the main contributor affecting the accuracy of the ArUco ID pose estimation will be the localisation error in the relative distance to marker from camera. Figure 9 shows the localisation error between the camera and marker proximity, it is shown that at 8m there is approximately a 3% error. With the testing room having dimensions of 10.1 times 9.1 m; the ArUco localisation node is expected to have high accuracy percentage during operation.

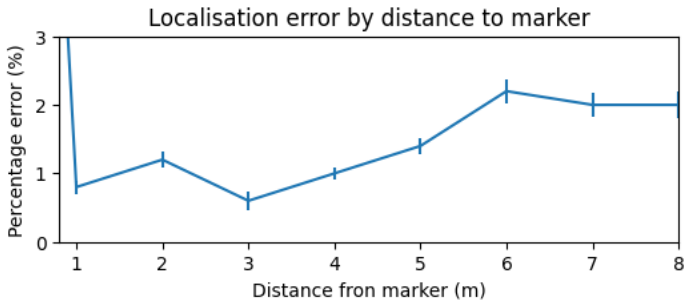


Figure 9: Plot showing localisation accuracy and its standard deviation relative to marker proximity

Closest ArUco Marker During operation, it was found that camera might detect multiple ArUco Markers in frame, leading to multiple readings of ArUco pose estimation. Acknowledging that there will be small measurement error when measuring ArUco position, the absolute position robot pose from multiple ArUco markers will be more varied when looking at only one. When the variance is big, this could lead to the robot getting confused on its true location and have oscillation effects where it tried to align itself to the accurate position most of the time. From Figure 9, it is known that closer proximity can have lower measurement error, hence it was decided to use only the closest ArUco Marker when multiple ArUco markers are detected. This would lead to a more stable behaviour where the robot only use one ArUco ID at a time as it true pose to move to the target waypoint.

Motion Blur Detection During operation, it was found that motion blur from the camera (will be discussed further in Experiments section - 4) have caused ArUco algorithm to detect false positive or detect the wrong ArUco marker. False detection of ArUco Marker will leads to wrong localisation positioning of the robot; this is due to Fused pose having high weighting on the ArUco Pose. Therefore, false detections will

significantly impact robot's ability to complete the task. It is important for the algorithm to detect blur during operation to disqualify blurry images and only use images that are not blurred for ArUco detection. The chosen method for this was utilising OpenCV implementation of Laplacian kernel to detect regions of image containing rapid intensity changes and applying the variance to find the spread of response, knowing that high variance will indicates a normal image, and low variance will indicates a blurry images, as there are less edge detection response from intensity. This is done by applying an appropriate threshold to not use any low-variance images below it, and apply the ArUco algorithm onto high variance images that are above the threshold value.

Avoid Blurring In addition to motion blur detection algorithm used to detect the blur for ArUco Marker, an additional stop was implemented into the Finite State Machine (FSM) of the robot. It was found that the odometry of the robot works with minimal drift within a short distance over a small timeframe; the additional stop was implemented into the robot for the purpose of re-localisation of the robot based on the ArUco. The stop was to ensure that the camera is able to detect a high quality image with minimal blurring for accurate pose estimation of the robot. By stopping the robot, robot pose would not be affected by encoder and hence pose fusion will be purely on the ArUco marker which have been demonstrated to have minimal error percentage (Figure 9).

ArUco False Positives During testing and integration, it was found that camera repetitively detect false positive ArUco even when there was no ArUco around. This misbehaviour only happens with specific ArUco IDs, and it was concluded that this could be due a combination effects of the ArUco design being easily misinterpreted (such as ID 17) and the operating environment within the room that could led to false detections of an ArUco ID. The proposed workaround for this problem was to not use these ArUco ID during trial runs, as well as blacklisting any ArUco ID that are not being used.

To find markers that are easily mistaken, an experiment was carried out where the robot, with ArUco detector enabled, is manually traversed through the course where no markers were actually in the field. Table 2 shows the counting table of wrong ArUco IDs identified during this experiment. All markers identified here will not be used during demonstration run and were blacklisted in the algorithm to instantly disqualify them even when the camera detects them.

Shape	ID	False Positive Count
	17	12
	28	4
	21	2

Table 2: Counting table of wrong ARuCo identifications

3.2.3 Pose fusion As it uses a proprioceptive sensor, Wheel Odometry naturally drifts over time, accumulating error proportionally to the distance travelled. At a speed of just $0.25ms^{-1}$, Figure 10 shows a run of the course from one end to the other exemplary of the initial tests conducted using just wheel odometry to localise. The measured error in the odometric versus actual end position was roughly $83cm$. This error only worsens as the speed of the robot increases, as detailed in the Preliminary report on Wheel Odometry [4]. Furthermore, preliminary testing also found the wheel encoders suffered from a sort of "crosstalk" caused by the movement of one motor, introducing noise in the encoder measurements of the other motor. Thus, pose estimates from an exteroceptive sensors such as a camera are necessary to get an estimate that is not dependent on distance travelled.

Pose fusion

- **Function:** Combine pose estimates from camera and wheel odometry
- **Input:** Camera pose and Odometry pose
- **Output:** Fused pose
- **Dependency:** ArUco CV and Wheel Odometry

ArUco-Odometry Comparison Preliminary testing in the report on using ArUco markers with computer vision to determine pose [6] found that errors were less than 1% (6cm) for distances under 6 meters and up to 2% (16cm) for 8 meters. Naively, with such decent accuracies one would think to only use ArUco marker pose estimation and forego wheel odometry. However, there are some tradeoffs that make using computer vision by itself undesirable or unreliable. The first of these is that for the robot to update its pose, at least one marker must be in the (limited field of view of the) camera frame, thus a high number of ArUco markers would be needed which decreases the desirability of the system. Even with the guarantee of seeing at least one marker at any one time, the markers may not remain in their correct poses or may be occluded by unplanned objects like humans walking in front of the markers. Additionally, test runs of the course using ArUco markers found pose estimates to be noisy due to

motion blur (Section 3.2.2). This noise can also be seen in an exemplary run using the odometry with computer vision to localise (method described below) as shown in Figure 11. Ideally, comparisons should have been made of the same path to follow, but regardless there is a clear distinction in the style of the measured path (red lines) between Figure 10 and Figure 11. With computer vision, the pose estimate is susceptible to jumps or discontinuities that are as far away as $2.25m$ (the two dots in the lower third of the image). On the other hand, using just wheel odometry produces a smooth trail.

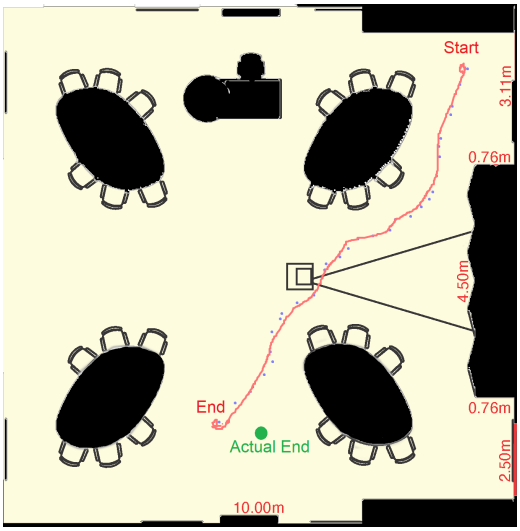


Figure 10: Test run using wheel odometry localisation. The bluish-purple dots are the planned path, the red line is the path measured using wheel odometry and the green dot is the actual ending location of the robot.

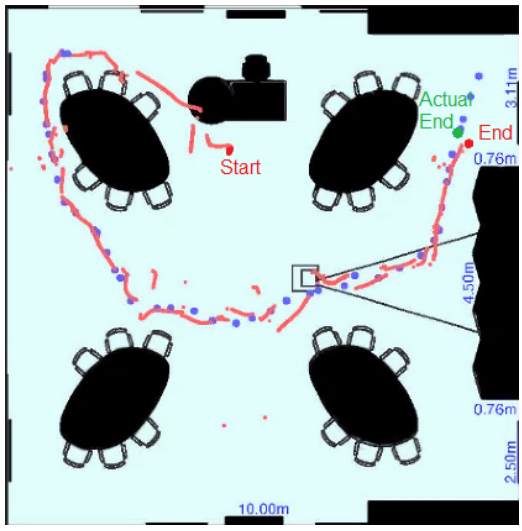


Figure 11: Test run using fused pose localisation. The bluish-purple dots are the planned path, the red line is the path measured using the fused poses and the green dot is the actual ending location of the robot.

Pose Estimation Cycle A simple method of fusing the estimates was implemented. The robot was made to periodically stop for 1 out of every 15 seconds to use the pose estimate (from the closest ArUco marker as previously explained) as the truth and then accumulate the changes in pose using wheel odometry in the other 14 seconds. Initially the robot was not made to stop to use the pose estimate from the camera. Further testing revealed the pause improved the performance. Figure 11 shows an example run where the robot demonstrated better success in tracking the planned path. The design proved to be satisfactory for slow speeds ($\leq 0.1ms^{-1}$) as the robot was able to navigate to its goal without help in 4 out of 5 runs. Unsuccessful runs were usually due to collisions with chairs being too far out from the table. Due to time constraints, other more sophisticated methods such as Kalman filtering were not implemented, and were left as potentials for further developments in the future.

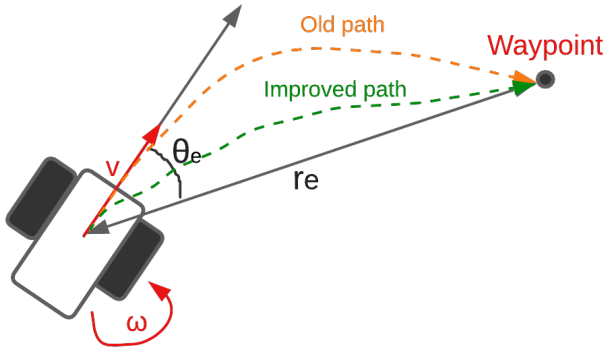


Figure 12: Angular error θ_e, linear error r_e and paths to waypoint before and after improvement

3.2.4 Motion control (outer loop)

The motion control of a two-wheel differential drive robot can be divided into two controllers - one for purely linear velocity v and the other for purely angular velocity ω. Therefore, developing a motion controller can be simplified into developing a function that determines the required linear and angular velocities to move the robot to the target waypoint.

Motion control

- **Function:** Determine the required wheel velocities to move to a target location
- **Input:** Target pose and Fused pose
- **Output:** Target left and right wheel velocities
- **Dependency:** Pose Fusion and Target Waypoint

"Artificial Potential Field" Controller A simple but effective method of motion control was used. This method acts like

an artificial potential field where the linear and angular velocity required of the robot is proportional to the difference/error in heading (θ_e) and position (r_e) respectively. In other words these can be thought of as proportional controllers

$$v = c_r r_e \tag{3}$$

$$\omega = c_\theta \theta_e \tag{4}$$

Optimal values for the constants of proportionality $c_r = 0.1s^{-1}$ and $c_\theta = 0.65s^{-1}$ were determined empirically by tuning each one independently and observing the performance. Higher values made the robot move too quickly and become unstable while smaller values made the robot move too slowly.

Waypoint Alignment Initial testing of simply moving to arbitrary waypoints from various starting poses revealed the robot was susceptible to encircling the waypoint indefinitely (like a moth encircling a light source) if it was not pointing within 5° of facing the waypoint when within $0.1m$ distance to it. Thus changes were made to only send linear velocity commands v once the robot was within 5° of facing the waypoint which meant the robot could turn most of the way to face the waypoint before moving forward. This would also allow it to run a more direct path to the waypoint rather than curving so much (if it started the movement with a large angular error θ_e).

Controller Characteristics This motion control method is smooth as the change in velocity is a continuous linear function of position - in other words the acceleration is never infinity or extremely high. Additionally, it is an inherently stable method as the robot's position will eventually converge to the target waypoint (given the improvements made above) - indeed this occurred 100% of the time in testing. Otherwise, note that it is possible to get runaway velocities (as they can increase without limit). Thus clamping (i.e. limiting the velocities) was implemented. In most cases, there is no need for clamping as the path planning algorithm will generate waypoints close enough to each other such that the velocities are always bounded. However, having clamping guarantees the robot is robust to edge cases where the robot is tricked into thinking it is somewhere much further away from the waypoint than it actually is (e.g. errors in computer vision) or is actually picked up and moved far away.

3.2.5 Motor control (inner-loop)

Given that there is no strict time constraint for moving between waypoints, and the Motion Control guarantees the robot eventually reaches and stops at the target waypoint, it is not vital that the robot and hence the wheels follow the desired setpoint velocities closely. There can be steady state error in the velocities. Thus a simple proportional controller was used to control the angular velocities of the left and right wheels independently.

Motor control

- **Function:** Control the angular velocities of left and right wheels
- **Input:** Target left and right wheel velocities
- **Output:** Encoder counts
- **Dependency:** Motion Control or Obstacle Identifier or Plant Indicator

The proportional gain was tuned empirically by finding the highest value that does not cause the velocity to oscillate or become unstable around the steady state value. As shown in Appendix C, different P values were tested, where their corresponding step responses were plotted during tuning. To improve the controller and track setpoints with less steady state error, as well as tune rise time, settling time and overshoot, the Integral (I) and Derivative (D) components of the PID controller can be added [7] with the effects of adding or increasing the terms shown in Table 3.

Increased parameter	Rise time	Overshoot	Settling time	SS Error
K_p	Decrease	Increase	Small change	Decrease
K_i	Decrease	Increase	Increase	Decrease
K_d	Small change	Decrease	Decrease	No change

Table 3: Effect of increasing/adding the P, I, D terms in a PID controller [7]

3.3 Camera Tracking

As previously mentioned, camera tracking subsystem handles all robot's tasks that are related to detection of objects, and the subsequent processing of these detection results to provide crucial information to other subsystems and satisfying the project's primary objective of plant inspection.

3.3.1 Object Detection (OD)

Object detection is in charge of identifying the presence and type of objects within the camera view. Its detection can help in identifying plants (Tasks B3, E3), and obstacles (Tasks B2, E2).

Object Detector

- **Function:** Detect a variety of objects
- **Input:** RGB image from webcam
- **Output:** Detection results in the form of rectangular bounding box & corresponding confidence
- **Dependency:** Camera

Method: Deep Learning There are various simple approaches to identify an object, such as color distribution or shape structure. Most are not robust however, due to variability of object characteristics and environmental conditions that can lead to varied color and shaping of the same object category. These challenges at present are addressed with deep learning, which uses neural network architectures trained with vast amount of imagery data of objects in different conditions. This way, a given object type will be detected in diverse scenarios, regardless of color or shape. Therefore, deep learning will be the main method for the robot's OD functionality.

Programming & Libraries: Python + `mmcv` package ROS uses 2 main programming languages, C++ and Python. Specifically for object detection, Python is implemented, as it is supported by many pre-built packages that make the implementation process more efficient and simple. The deep learning library `mmcv` under `pytorch` base is used. It is a comprehensive open source, free-to-use library on deep learning models, with each model have a pre-trained weights that can be directly loaded for predicting new data instances, as well as reporting metrics on performance and runtime. It integrates well with NVIDIA products, including the Jetson processors like the robot's computer, and is a base for the implementation of many applications or expansion libraries in deep learning.

Pretrained dataset: COCO Common Objects in Context (COCO) is chosen as the dataset for the candidate models to pre-train on, as it is the most commonly used dataset for various computer vision tasks in the training step. It consists of 385000 images with labels in 91 different categories [8]. They include 'potted plant' and 'vase' - which can be used in the project to identify the plants in the course, or 'human', 'desk' and 'chair', which are the majority of the obstacles in the project course. With vast amount of training data, they are expected to cover most variability of the object categories mentioned.

Model & Processor: YOLO on GPU 3 commonly used object detection models in `mmcv` are tested in order to choose the most suitable model for the system: Faster R-CNN [9], YOLO [10], and HRNet [11]. Each model will be integrated with the robot under either CPU or GPU processor and manually traverse a given path in the course. It has to detect the following labels while traversing: human, potted plant, vase and chair. While operating, the performance speed in FPS will be measured on each frame, and the output results will be manually counted for number of missing detection (false negatives) and incorrect detection (false positives). Results will be evaluated based on trade-off between performance speed and accuracy.

Model	Mean FPS (CPU/GPU)	Reported mAP [2]	Precision /Recall
Faster R-CNN	0.184/1.83	37.5	100%/72.5%
YOLO	1.180/10.78	27.9	100%/68.6%
HRNet	0.177/1.60	36.9	100%/71.1%

Table 4: Model comparison results on the robot

The result in Table 4 shows that the GPU indeed improved the inference speed vastly, approximately 10 times faster than the CPU-operated counterpart, showing that the GPU's ability to allow multiple processes to run concurrently can speed up the progress significantly [12]. For models, YOLO model shows outstanding performance speed comparing to other models, despite having lower mAP than other models. Lower mAP is proven to have little significance, as the number of missed-outs (reflected by recall) does not vary much between models. With the given results, YOLO model with GPU processor will be implemented in this project.

Problem Description	Count	Suggested Solution
Darkened object due to strong lighting	27	Attach fixed LED
Blurry details due to quick movement	130	Fixed shutter speed Slower movement Attach fixed LED
Mistaking multiple adjacent plants as one	5	Combine LIDAR scan
Irregular-shaped object	44	Customized training
Tables never detected	133	Customized training Treat as no-go zones

Table 5: Summarizing of preliminary experiment on OD, consisting of manually counted error types for 277 false negatives found in the test (YOLO on GPU), and suggested solutions

[2] Mean Average Precision (mAP) is a commonly used metric to evaluate OD correctness based on spatial correctness (overlap between prediction and ground truth bounding box). The library `mmcv` has mAP reported for each model.

False Negatives Table 4 also shows that the model is likely to miss out a plant under certain difficult conditions, but can hardly make any mistake predictions. Table 5 summarized the causes of false negatives and the corresponding solutions offered that were described in details in the preliminary report [13], with image examples from the experiment shown in Appendix D. These mentioned problems will be taken into consideration, where they become motivation to design another functionality in the Stretch Functionalities, or a possible cause to look for when experiment stage does not work as expected.

Figure 13: An occluded plant being detected

Confidence Thresholding Despite having no false positives, a problem observed is that it can detect a plant partially occluded by obstacles (Figure 13), which the robot may bump into when approaching the plant. As a result, a threshold of confidence is applied, where a detection has to have a high enough confidence to be considered. This is to ensure maximal prevention of false positives or occluded detection, which can disrupt the upcoming dependent functionalities. This is also a trade-off, as higher threshold can also lead to more false negatives, hence the selection of the threshold requires rigorous testing to pick the lowest threshold possible. The robot was made to observe multiple occluded plants in the EDS10, where OD may detect them. The highest confidence in these detections was 0.66, and therefore, we decided to make 0.7 as our threshold, taking additional safe margin into account.

Additional Hardware: LED As mentioned in Table 5, attaching a fixed LED is a feasible workaround for the backlighting and motion blur problem. It works by lighting up the area in front, providing better light balance between object and background, as well as preventing the problem of darkened image under higher shutter speed. It only requires battery power to operate and does not require any additional computation resource, making it feasible as an extra equipment for the robot. Further experiments will test the feasibility of this addition.

3.3.2 Plant Indicator (PI)

PI serves as an additional support to the plant detection part of OD functionality (Task E3). It extracts information from the plant detection results, and can determine the status of the plant, as well as determining the robot's position with respect to the plant and give the next movement direction accordingly.

Plant Indicator

- **Function:** Determine state of a detected plant and direction to approach it
- **Input:** Bounding box of the plant and its label
- **Output:** 1 binary label indicating if plant is found before, 1 label indicating plant status and movement direction in the form of $[x, y, \phi]^T$
- **Dependency:** Object Detection, Robot pose

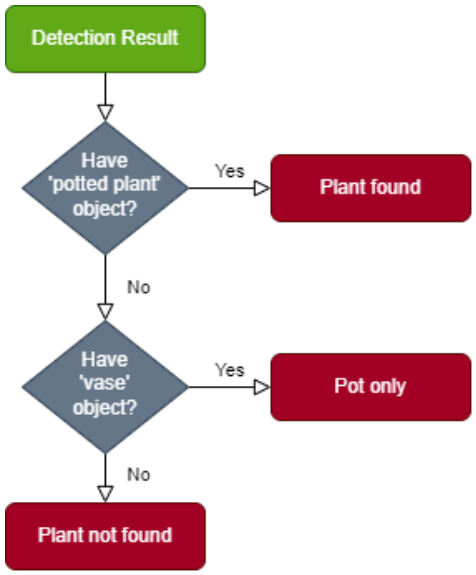


Figure 14: Logic diagram to indicate plant status

Status Indicator In COCO labels, there are 2 labels that are able to indicate a pot: 'potted plant' and 'vase'. Observation from OD preliminary experiment shows that label 'potted plant' is only assigned when a pot is found with a plant in it, and 'vase' is usually assigned when no plants are found, either missing or heavily occluded (see Appendix Figure 40). Therefore, 2 possible status indicator - 'Plant found' and 'Pot only', will be indicated by logic shown in Figure 14.

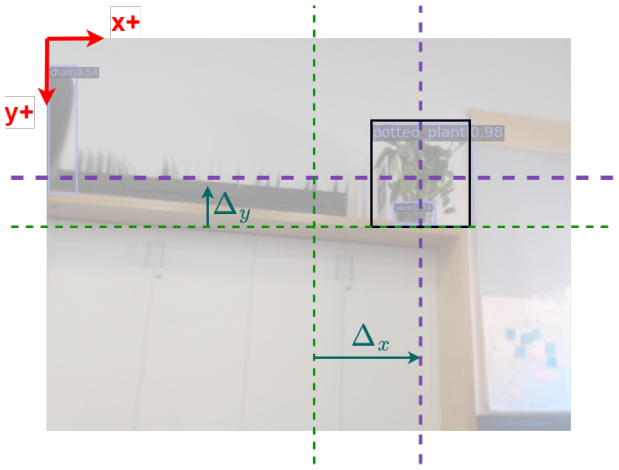


Figure 15: Image convention & center offset variables Δ_x, Δ_y

Aligning Movement The bounding box given of a plant can be used to determine where the plant is relative to the robot, and alignment movements can be made accordingly. Specifically, the x and y axis center offset between the bounding box and the camera frame can be used to determine which direction is the plant at, shown in Figure 15. When multiple plants are detected in the same frame, only the one with the biggest bounding box will be used for the process, under the assumption that preference is given to the nearest plant.

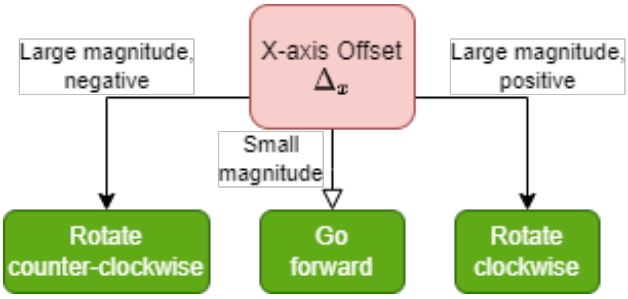


Figure 16: How the robot will align to the detected plant

Offset of x-axes can maneuver the rotation of the robot in order to align orientation, as shown in Figure 16. A small offset indicates that the plant is right in front of the robot, hence the move forward action. When the plant is close enough, indicated by the bounding box height, the camera will take a picture, and no longer approach the plant.

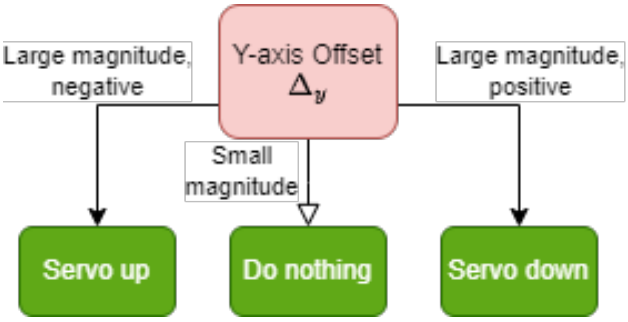


Figure 17: How movement of servo motor is determined

Offset of y-axes can control the servo motor to go up or down to adjust the camera so that the plant will be in the middle of the frame, giving better composition for picture taking purposes. A demonstration figure is shown in Figure 17.

Plant Occupancy Based on the position estimation of the robot and the plant, the status of whether a plant has been seen or not may also be calculated. It can be useful in known area by removing redundancy when a plant is observed twice. Futhermore, this functionality can help enhance user experience, by letting them know the status of the robot's progress, or the plants themselves. The functionality requires one function that detects the zone a plant is in, under the assumption that each plant are at least distance from each other (i.e. no adjacent plants). For demonstration purpose, this functionality will be done for EDS10 sections divided as shown in Figure 1, each of which is assumed to have 1 plant. The plant's zone will be determined when the robot is up close (based on the bounding box height), where the zone containing the robot will be determined to be the plant's zone.

3.3.3 Obstacle Identifier (OI)

Similar to PI, OI can be used to extract useful information of the objects, with the main difference is that this is used for obstacles instead of plants (Task E2). It is able to estimate the proximity of an obstacle detected, and handle information accordingly. It is noted here that this functionality presumes that a dynamic path planning functionality is operational.

Obstacle Identifier

- **Function:** Determine the presence of an obstacle in front of the robot
- **Input:** Bounding box of obstacles
- **Output:** Decision of whether to stop, move, or generate a new path
- **Dependency:** Object detection, Dynamic Path Planning

Obstacle Type Assumption To cut down computation time, out of 91 labels, it will be assumed that only 2 types of obstacle will be appearing: chairs and humans. While tables are obstacles in the area, preliminary experiment have shown that OD models could not detect the tables in the course due to irregular shaping (see Appendix Figure 41). Along with the fact that the tables are large and fixed in the area, areas with the table will be treated as a no-go zone instead of getting detected through OD functionality, which can be avoided with path planning or LIDAR detection (where available).

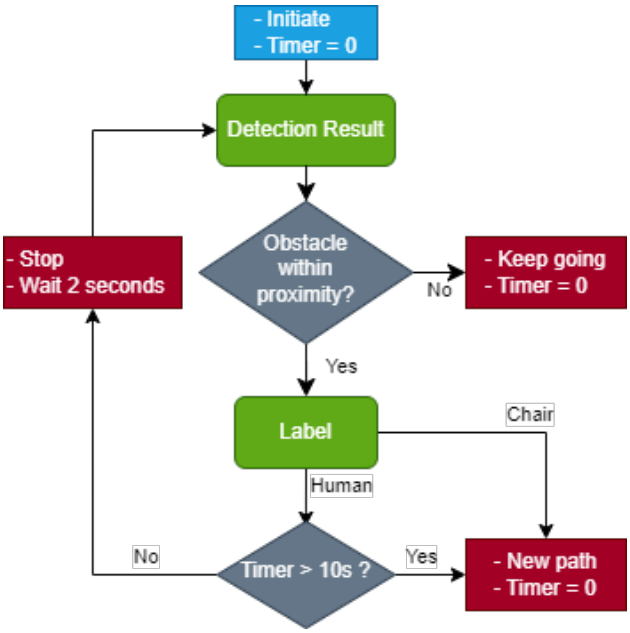


Figure 18: Logical flow for obstacle avoidance

Proximity Handle: Stop when up close For every detected obstacle type, similar to plants, the bounding box size (i.e. height ratio between box and image frame) can be used to determine whether the obstacle is within a specified proximity of the robot. When the obstacle is considered to be close enough, 2 possible cases will happen:

- For chair obstacles, as they are presumably fixed, the functionality would send a signal to path planning to make an alternative path instead.
- For human, the robot will stop and perform a check every 2 seconds. If the human is no longer there, it will continue the movement. Otherwise, if 10 seconds is exceeded since the first detection, the robot will plan an alternative path instead.

The flowchart summarizing the process is shown in Figure 18.

3.4 Stretch Functionalities

This subsystem is a set of functionalities that operate as a supplement to the tasks performed by other subsystems. While these functionalities are not obligatory for the fundamental tasks, they offer valuable enhancements to the overall robot system.

ROSBoard user display

- **Function:** Display ROS topics in real time
- **Input:** ROS topics
- **Output:** UI displaying all topics of interest
- **Dependency:** None (displays any active topic)

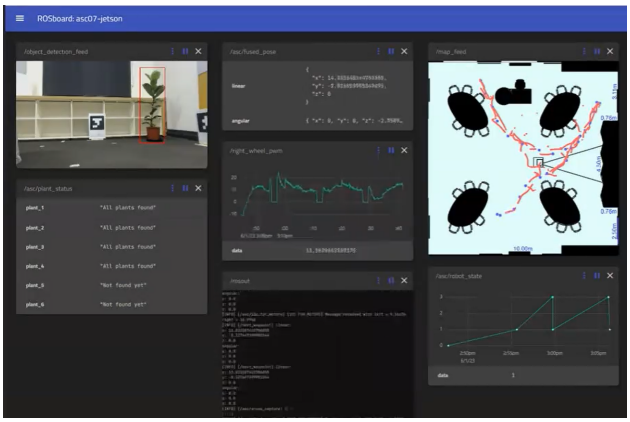


Figure 19: Overview of ROSBoard on web browser

3.4.1 ROSBoard User Interface

ROSBoard provides an interactive, user-friendly interface within ROS. It allows user to monitor all functionalities inside the robot, by displaying all active topics in real time. It can be accessible by any devices within the same network, making the process of visualizing sensor data or analyze the robot's performance convenient and interruption-free.

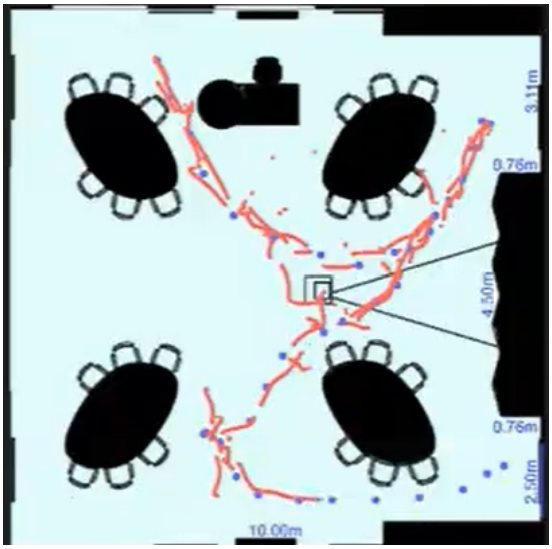


Figure 20: A map streaming sample on ROSBoard

Map Streaming A real-time map is created to keep track of path planning and path following processes. As shown in Figure 20, it displays 2 main items. The blue dots represent the intended waypoints generated by RRT* algorithm, displaying the intended path that the robot will traverse. The red lines indicates the robot's localization, where each dot represents the robot's self-estimated position in the map at a time instance. These 2 visual components will help user indicating the performance of path generation (how far the waypoints are from no-go zones), path following (how well the movements aligned to the waypoints) and localisation (how continuous is the red line).

Figure 21: Live display for OD on ROSBoard

OD Streaming It is noticed that while ROSBoard can display any topic, visualising OD results slows down the capture speed by half, due to 2 reasons: the image size of 1920×1080 pixels is large and takes time to render, and in sequential programming, drawing bounding box comes after the detection, adding an extra step that the publisher has to wait for. Therefore, a streamer (Figure 21) is dedicated for displaying object detection, where it will receive image and detection information simultaneously. Not only that it can perform visualisation in parallel with the detection process, it also resizes the image by half, reducing the rendering time by a fourth.

3.4.2 Hector SLAM using a 2D LiDAR

Although it was not implemented in the system, the potential for using SLAM was explored by installing an existing package implementing the Hector SLAM algorithm [14]. The potential exists to use this package to create a map of the course and use the map to localise. Figure 22 shows the result in rviz visualizer after scanning a testing room.

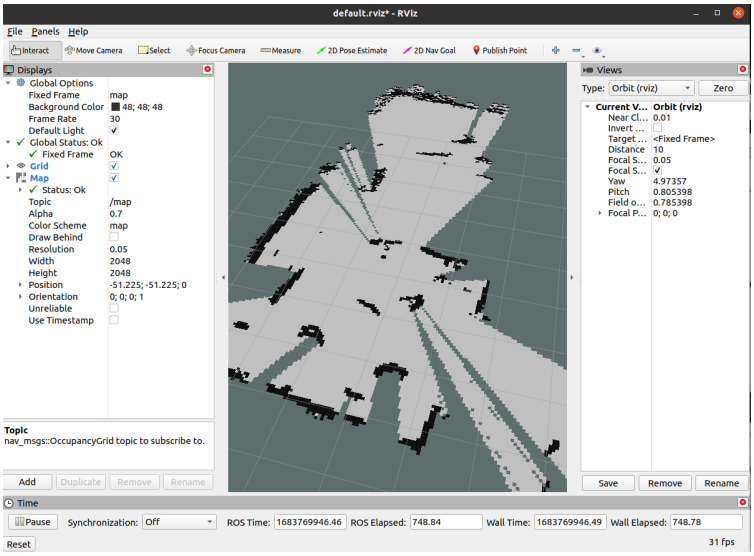


Figure 22: 2D point cloud map of a room using Hector SLAM and a 2D LiDAR in rviz

4 Trial Experiments & Adjustments

In this section, the robot will be comprehensively tested as a whole under conditions expected during demonstration. In specific, it ensure validity of adjustments made during the preliminary experiment, as well as feasibility of the functionalities described in the previous section. On a larger scale, it evaluates how well these functionalities can integrate, communicate and work together. Subsequently, decisions will be made for which functionalities to keep, modify or discard in the demonstration prototype. In short, through this rigorous testing and decision-making process, the robot's capabilities and overall functionality will be refined and optimized for optimal performance.

4.1 Set-up

This part of the section describes the environment where the testing of the robot will be conducted, as well as assumptions that were made accordingly to the design and functionalities of the robot.

4.1.1 Course

Map As mentioned, the tasks for this robot will be operated on EDS10 part, with sections and coordinate conventions shown in Figure 1. Fixed tables are cupboards, along with positions outside EDS10 are treated as no-go zones for the robot.

ARuCo Marker Distribution ArUco markers are strategically placed around all side of the room and firmly against the wall to allow for sufficient coverage of the entire room, ensuring that there are at least one ArUco seen by the robot

for the room. Furthermore, by placing it against the wall, this allows minimal obstacle or obstruction within the room and less modification needed to the operating space, hence decreases setup time for the clients. In addition, there is a trio of markers placed at the centre of the room, these 3 ArUco markers represent the middle point of the room. They further enhance vehicle localisation by improving ArUco localisation algorithm, as each point are within under 6 metres from the nearest ArUco. Therefore, there are approximately 1% percentage error of localisation expected when compared to higher percentage error if the robot sees an ArUco that is 9 to 10 meters away. Moreover, to enhance table avoidance especially in tight space maneuver, there are always an ArUco within line of sight so that the robot will have minimal drifting during operation and thus reduces potential collision. Overall, 17-18 ArUco markers were used during both demonstration and testing, however the number could be further reduced, when more testings can be carried, and redundant markers can be further found and removed from the course.

Randomised Starting Position The robot will have arbitrary starting position, as it is expected to be able to localise itself upon seeing an ARuCo marker.

Day-Night Testing Due to limited development time given and the need to share space with other project participants, the testing runs of the system will be conducted at both daytime and nighttime conditions. This on the other hand, would enable a thorough evaluation of the system's performance under different lighting and environmental scenarios, effectively assessing robot robustness and adaptability.

4.1.2 Assumptions

A number of assumptions will be made during experiment, to help ensuring optimal demonstration of all robot functionalities. These assumptions serve as guidelines to enhance performance and effectiveness of the overall system. By aligning experimental conditions with these assumptions, we aim to showcase capabilities of the robot in their best possible light.

1 plant in each zone This is to demonstrate the performance of the Plant Indicator functionality, where repeated detections can be neglected, and status of each plant can be updated.

Obstacles are either chairs or humans This is to ensure that the obstacle identification works as expected, without any interruptions due to unseen categories.

[3] Later on this was unable to be done, due to resource unavailability

4.2 Results & Adjustments

After performing testing on the robot, a number of observations were made, and corresponding changes or adjustments to the systems were done to refine the robot for the demonstration.

4.2.1 Performance in general

Overall, the robot was not able to operate fully during the first test runs. A number of problems were detected, and incremental changes are made until continuous baselines were met. The next part of this section will list out the problems observed, and corresponding changes made.

4.2.2 Image Quality Degrade

Nighttime testing showed an increasing amount of false negatives for detection of ARuCo markers and objects. Inspecting images taken shows that the environment is too dark, causing the image to be either blurry (Appendix Figure 39) or heavily darkened (Appendix Figure 43). Both have made the objects unrecognizable to the model. The allocated LED attachment has limited lighting range, hence was only able to improve the situation when object is up close.

Design changes

- The robot will move slower when operating in nighttime.
- Replace current LED attachment to a brighter one when available [3].

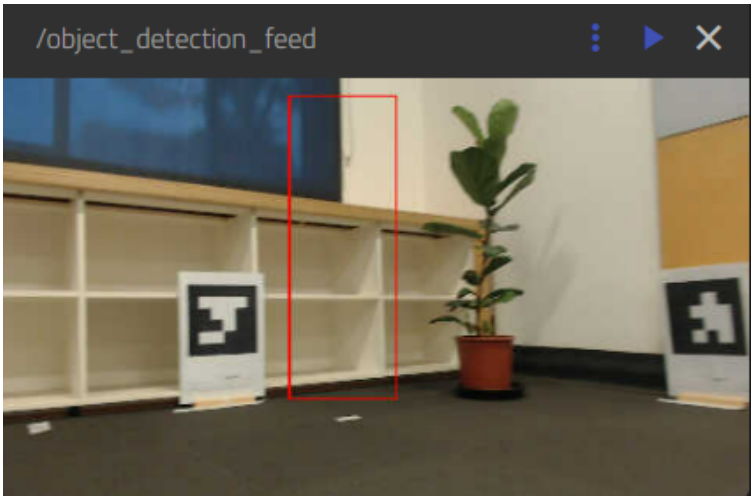


Figure 23: A delayed detection of plant when robot was rotating left

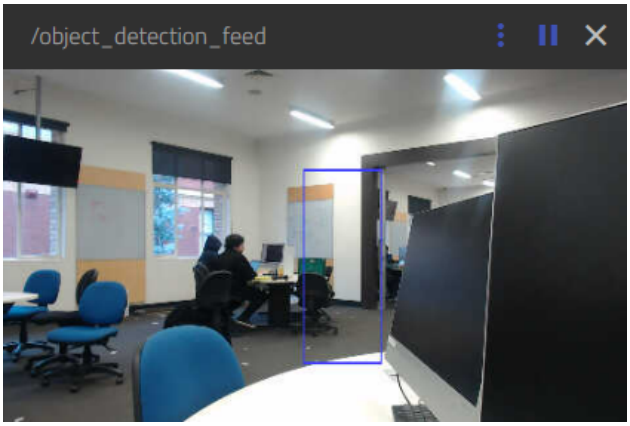


Figure 24: A delayed detection of a human who was standing 2 seconds before

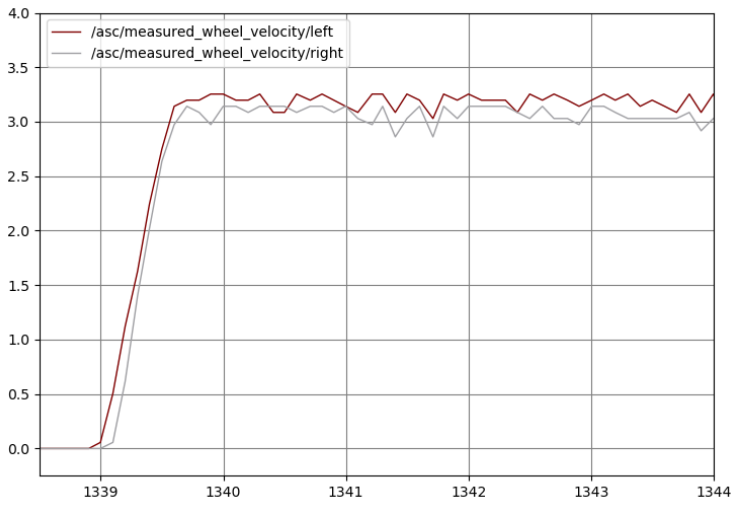


Figure 25: A difference in moving time between 2 wheels occurred during the testing

4.2.3 Delay Testing subsystems together also showed a significant delay for topic communication comparing to the preliminary experiment. Setting timer shows that publishing OD bounding boxes takes an average of 2.66 seconds for all subscribers to retrieve and perform the dependent functionalities. Consequences of this problem includes:

- Inaccurate OD results (Figures 23, 24)
- Over-rotation when aligning the robot to the plant
- Late stopping making the robot hit the obstacle
- State controller also suffered this problem when the state update was not published in time, causing the robot to make unpredictable behaviors.
- Different starting time between 2 wheels (shown in Figure 25), causing incorrect heading.

Design changes

- Lower movement speed to prevent overshooting. After letting the robot operate and detect objects under different speeds, a 40% wheel speed reduction is found to be the minimum reduction that keep all tasks functional reasonably.

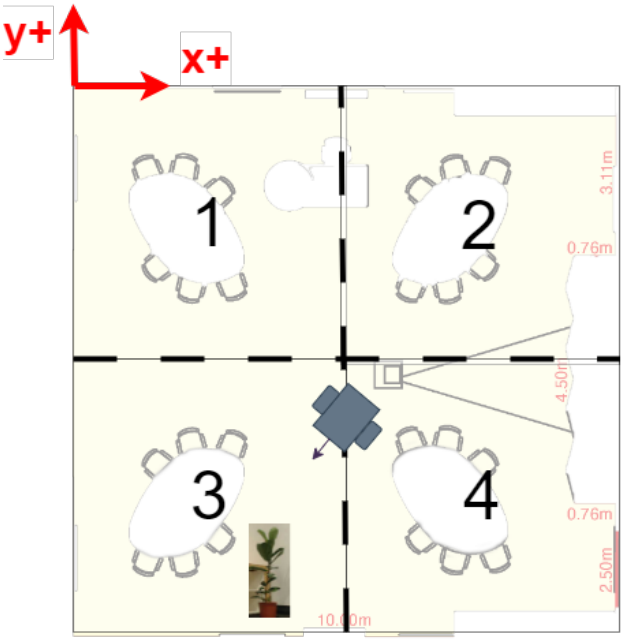


Figure 26: The robot's position during wrong zone mistake

4.2.4 Wrong zone detection Another problem is observed when the robot identifies the plant in the wrong zone. Inspection shows that this happens when both the robot and plant are near or inside a zone border, as shown in Figure 26. This is expected, as the localisation is expected to have error, and it can be possible for a robot to observe a plant in the different zone, making the assumption of the current method not hold (plant and robot are in the same zone).

4.2.5 Unavailable Dependency Due to limited time and resources constraints, some functionalities could not be fully tested, or run successfully when implemented. Not only that they cannot operate, functionalities that were developed but rely on the unimplemented dependencies cannot be deployed as well. The problems that arisen due to availability, and their corresponding changes or consequence includes:

- Dynamic Path Planning was not completed in time. As a result, Object Identifier is undeployed, as there are no nodes left that subscribe it.
- There were no empty pots available in the course, only potted plants. Therefore, Plant Identifier cannot demonstrate its status indicator task.

4.2.6 Solution: Simplified Demonstration Functionalities

After testing and identification of problems, a simplified system design was made for the robot, under the priority of satisfying the baseline tasks within limited development time. The following are summaries of how the simplified version of the robot is structured on each subsystem. 2 graphs showing the ROS topics before and after amendments are also available in Appendix Figure 57.

Path Planning After testing, it was decided that path following algorithm perform really well therefore, path generation was modified to only generated and send every second waypoint along the path to reduced the number of waypoint needed to travel along the path but also to increase the speed and decrease time that the robot moves around the space significantly. It is to be noted that the first and last waypoint was kept constant and the middle waypoint were reduced to only every second waypoint. For waypoint thresholding, the threshold boundary was increase as well to decrease the time in which robot moves from start to destination.

Path Following The motion control and motor control functionalities were not altered as they had already been verified. However the pose fusion was simplified by pausing the robot and only using camera pose estimates if the robot is not moving to guarantee motion blur is not affecting the measurement.

Camera Tracking As the state controller is no longer controlling plant approach process, the whole process of alignment and reaching the plant, as well as conditional picture taking tasks are removed. The logic was simplified to be setting the camera to take a picture for every successful detection of a plant or pot on the way. As label information is retained, the status indicator is still kept, which means the robot is still able to tell whether the pot detected has plant, as well as which zone the detected plant was in.

5 Demo & Evaluation

After a series of preliminary experiment and integration testing, the final implementation version of the robot system is showcased in the demonstration session of the project. This section will describe the demonstration session and outcomes in terms of robot performance, followed by analysis and assessments on how well our design has met the requirements of the project.

5.1 Demonstration

5.1.1 Set-up

The setup in the demonstration session is similar to our robot integration experiment, where the full operation will be done in EDS10 area. The robot starts at an arbitrary position, which it were put at the border of Zone 1 and 2, and 4 pots of plant of varied shapes are distributed arbitrarily in 4 zones of the course so that each zone has 1 plant. The demonstration is done during daytime, where the robot has a 25-minute time block to operate. Each participant will only have 1 attempt to run the demonstration.

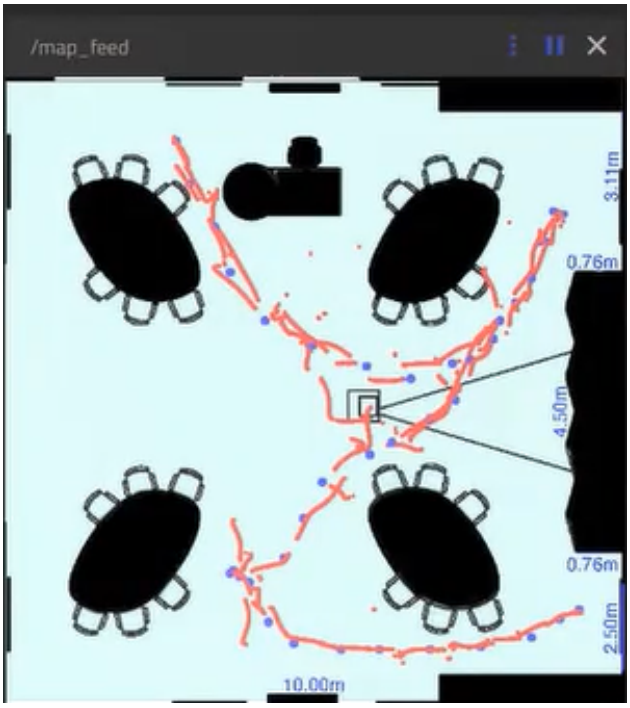


Figure 27: The ROSBoard map showing the robot's traversed path during the demonstration

Figure 28: Images of 4 plants taken during the demonstration

5.1.2 Outcome

Overall, the robot showed reasonable performance within the given attempt, where it was able to accomplish all the baseline tasks, and partial success in the extra tasks. Figure 27 shows the demonstration path that the robot traversed. Detailed of the outcome is listed as follows:

Completion The robot was able to traverse 4 sections of EDS10, and completed all tasks in approximately 20 minutes.

Path Generation The robot was able to create successful path of waypoints that head from one section of the course to another, and eventually, able to generate the traverse path for the whole course.

Path Following & Localisation The robot was able to follow the path at a slow pace. Drifting is observed frequently, but the robot managed to correct itself with ARuCo markers.

Plant Tracking The robot was able to take pictures of all 4 plants on its way (Figure 28). The status indicator works well, but demonstrated the same border mistake mentioned in the last experiment. As a result, when detecting a Zone 3 plant, both Zone 3 and Zone 4 are marked as 'plant found' when the plant at Zone 4 was not found yet.

Task	Working	Autonomy	Speed	Robustness
B1	Yes	Yes	Slow	Good
B2	Yes	Yes	Slow	Good
B3	Yes	Yes	Fast	Good
E1	Yes	Yes	Fast	Good
E2	No	N/A	N/A	N/A
E3	Partial	Yes	Fast	Reasonable

Table 6: Evaluation of tasks in demonstration session

5.2 Result Evaluation Overall, all baseline tasks were achieved autonomously, along with few extra tasks. Most success tasks were able to demonstrate good robustness when traversing different area of EDS10, which have varied lighting conditions and objects on the way. A summary evaluation table is shown in Table 6.

5.2.1 Path Planning (B1, B2, E2) As expected, the overall path generation and next waypoint generated have successfully generate a path when requested and send the targeted waypoint when the robot is within the threshold boundary. Course Traversal (task B1) and Obstacle Avoidance (task B2) was demonstrated to have work successfully as well as provided potential expansion or improvement for task E2 which was not implemented due to time constraint.

5.2.2 Path Following (B1) As shown in the red dots far from the rest of the trail and the discontinuities of the measured poses over time in Figure 27, there is noise in the pose estimate caused by the computer vision. Having more sensors or higher quality sensors and a more intelligent process of fusing them would reduce the variability of the localisation estimates and improve accuracy. Otherwise, the motion control and motor control did well to move the robot to target waypoints, following them closely.

5.2.3 Object Detection (B3, E2, E3)

Plant Detection (B3) As expected, YOLO model was able to successfully detect the presence of plants in different areas, under varied conditions of lighting and plant shapes, at a decent mean frame rate of approximately 10 FPS. This makes operation in real time achievable for task B3.

Plant Detection Enhancement (E3) Only part of expansions to the Plant Detection task was implemented and demonstrated in the operating robot, hence the evaluation will only be partial. The implemented addition was the Plant Occupancy functionality, which similar to OD, demonstrated quick operation. It is robust in cases where the plants are far away from zone borders, but for one plant near the border, the same problem of wrong zone identification occurred again, degrading the overall performance of this functionality.

Obstacle Detection (E2) The Obstacle Detection functionality of the OD system was not applied, due to incomplete dependencies. Therefore, the task cannot be accessed as working.

5.2.4 Stretch Functionalities - ROSBoard (E1) ROSBoard was the only additional functionality that were implemented along with the robot system during demonstration. Being independent of other topics to start, It successfully demonstrated the operation in showing all active topics within the robot in real-time. Under stable internet, it is shown to be robust and accessible from all devices within the network as expected.

6 Project Discussion

6.1 Limitations The results shown in the demonstration was promising. It showed that the robot was capable of performing the primary tasks that the project client specified, along with some additional functionalities. However, there were still a lot of downsides observed, which have regrettably imposing limitations on the development process, and subsequently, the capability of the robot.

Timing Constraint The project assigns a 3-month duration for the development of the robot, including learning the necessary knowledge and gain familiar of the technical skills needed. Undoubtedly, this limited duration, combined with developer's other commitments in the university, presents a

significant challenge in achieving the complete development of the product, including the incorporation of additional capabilities. The primary requirements has to be prioritised, leaving room for many potential functionalities as a result.

Hardware Constraint The hardware components given for the robot has proven to be useful to carry out the given tasks. However, many functionalities were limited due to constrained capability of certain hardware elements, such as short lighting range of LED, or camera having low image quality in the dark. This have significantly limited the ability of the robot in terms of accomplishing the tasks.

Computation Constraint Computation power is a special but vital case of hardware constraint. It is the main resources for most activities happening inside the robot, only with exception of external attachments like LED that operates on a separate battery. As the resources in both CPU and GPU are limited, having too many simultaneous processes can cause delays due to inadequate power distribution. Furthermore, as different processes may have dependency to another one, slow performance of one process may lead to larger delays adding up together.

Knowledge Constraint This is also critical to most of the limitations in the robot capability. The developers of this robot are students, most of which have little practical exposure to skill-sets or knowledge topics needed to accomplish this project. With better familiarity, many limitations in development would have been solved at a faster pace, and more functionalities could have been developed.

6.2 Potential Improvements

Here lies the potential ideas with substantial benefits for the system, which could not be implemented due to the constraints and limitations mentioned above. When conditions are more amenable in the future, these ideas can be considered for development, presenting opportunities to further augment the capabilities of the robot.

6.2.1 Path Generation Optimisation

Other potential algorithms could be investigated that can guaranteed more optimal path with obstacle avoidance capabilities can be investigated. One example, would be to investigate into optimisation algorithm for travelling salesman problem through use of algorithm like Ant Colony Optimisation. However, the implementation of such algorithm will be difficult compare to other algorithms or RRT* presented here which may not be feasible for this project time frame.

6.2.2 Deep Learning based Plant Inspection

With deep learning, the plant detection can also be extended into an inspection system that can look for certain conditions such as species, sickness status or water leveling. It requires training data, consisting of pictures taken of plants in different conditions and angles, with labels or conditions annotated accurately. However, the amount of training data required is deep learning is large, which makes the project time frame may not be enough for data collection.

6.2.3 Adding LIDAR Obstacle Detection to Dynamic Path Planning

Dynamic path planning proposed as an extension in 2.3.1 would be enhanced by using the point cloud scan from the onboard 2D LiDAR to detect and avoid dynamic or unplanned obstacles such as humans and new structures. This would improve the safety of the autonomous system as the robot would be more aware of its surroundings as it changes in real time. If the unplanned obstacle is static, the robot will update the known map and get the robot to reroute around the new obstacle. The robot should also stop or slow down if it encounters dynamic obstacles such as a walking human. A method of Simultaneous Localisation and Mapping (SLAM) could be used to do all this in real time while the robot is moving.

6.2.4 Multi-thread Optimization

Most of the current functionalities in the robot are not deployed due to the processor having limited capability in handling large concurrency. When more knowledge on multi-core architectures and programming is known, the processor can be further utilized without having to provide any extra processing power to the robot, optimizing the hardware resources and cost at the same time, and allowing more functionalities to be implemented together.

7 Conclusion

Overall, our developed robot was successful in demonstrating its capabilities in performing the primary tasks given in the project. It can traverse the allocated course, and is able to detect the presence of plants in hindsight, and take pictures accordingly. The robot also demonstrates some additional functionalities that improved its desirability, by having an interactive, easy-to-access UI on ROSBoard that displays all the essential topics of interest, as well as an indicator system to check the visiting status of the plant in each designated section.

The success of the robot however, is still limited by many factors surrounding compatibility between subsystems and functionalities, as well as constraints in time, resources and knowledge. Had there been more relaxed condition, the robot would have been able to have a more rigorous testing, and

more functionalities could have been implemented to enhance its desirability. Regardless, this project has enabled us to get hands-on experience with the theoretical contents learned in university, and more knowledge on the world of autonomous systems.

REFERENCES

[1] NVIDIA Jetson. *NVIDIA Jetson Xavier NX System-on-Module Data Sheet*, 2019. v0.1.

[2] Dingfeng Chen, Qingchuan Xu, Jie Liu, Meiyuan Zou, Wenzheng Chi, and Lining Sun. A generalized voronoi diagram based robot exploration method for mobile robots. In *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1029–1035, 2022.

[3] Wei Wang, Hongli Gao, Qize Yi, Kaiyuan Zheng, and Tengda Gu. An improved rrt* path planning algorithm for service robot. In *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, volume 1, pages 1824–1828, 2020.

[4] Ernest Ze Cheong. Elen90090 autonomous systems clinic - prelimniary report. The University of Melbourne, 2023.

[5] Ngoc Trung Mai, Ren Komatsu, Hajime Asama, and Atsushi Yamashita. Pose estimation for event camera using charuco board based on image reconstruction. In *2023 IEEE/SICE International Symposium on System Integration (SII)*, pages 1–6, 2023.

[6] Phu Trinh. Elen90090 autonomous systems clinic - prelimniary report. The University of Melbourne, 2023.

[7] Control Tutorials for MATLAB and Simulink - Introduction: PID Controller Design.

[8] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Computer Vision – ECCV 2014*, pages 740–755. Springer International Publishing, 2014.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Jun 2017.

[10] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[11] Ke Sun, Yang Zhao, Borui Jiang, Tianheng Cheng, Bin Xiao, Dong Liu, Yadong Mu, Xinggang Wang, Wenyu Liu, and Jingdong Wang. High-resolution representations for labeling pixels and regions. *CoRR*, abs/1904.04514, 2019.

[12] Tao Xia and Jiabin Yuan. Research and implementation of GPU-based parallelization of spiking neural network. In *2022 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*. IEEE, April 2022.

[13] Khoi Nguyen. Elen90090 autonomous systems clinic - prelimniary report. The University of Melbourne, 2023.

[14] Stefan Kohlbrecher, Oskar von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 155–160, 2011.

APPENDIX

A Project Course

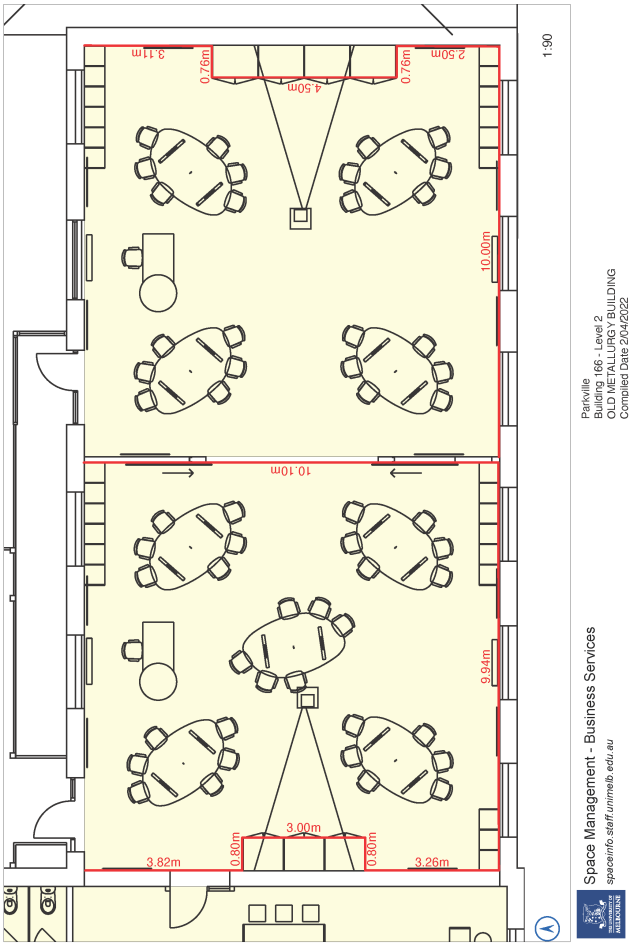


Figure 29: Course floorplan, consisting of rooms EDS9 (bottom) and EDS10 (top)

Figure 30: Overview of Room EDS9

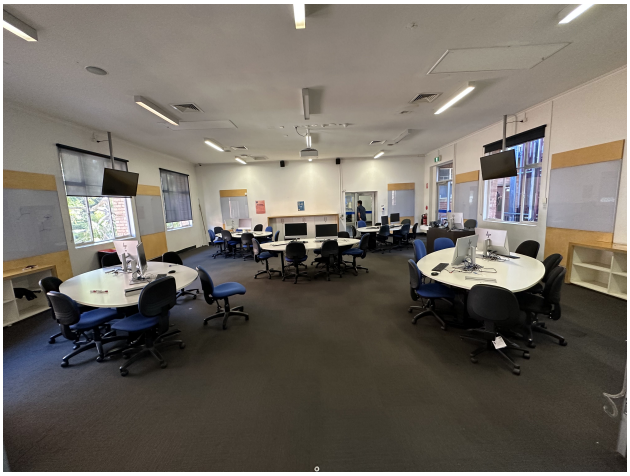


Figure 31: Overview of Room EDS10

B Robot model description and Wheel Odometry update

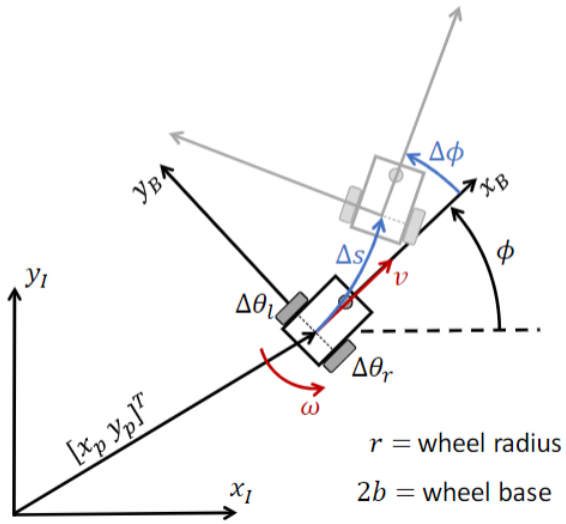


Figure 32: Robot body movement in the inertial frame: definitions. Sourced from ELEN90090, L08- "Odometry", P. Beuchat (2023)

B.1 Robot model description: Two-wheel differential drive The following definitions are used to model the robot and shown in the diagram of a two-wheel differential drive robot in Figure 32:

- Inertial frame represented by the axes: (x_l, y_l)
- Body frame represented by the axes: (x_B, y_B)
- Position and heading represented by the x, y translation and z rotation (respectively) of the Body frame with re-

spect to the Inertial frame: $\begin{bmatrix} x_p \\ y_p \\ \phi \end{bmatrix}$

- Left and right wheel angular displacements/positions: θ_l, θ_r
- Left and right wheel angular velocities: $\dot{\theta}_l, \dot{\theta}_r$
- Linear velocity along x_B: v
- Angular velocity about z_B: ω
- Wheel radius: r
- Half of the wheelbase: b

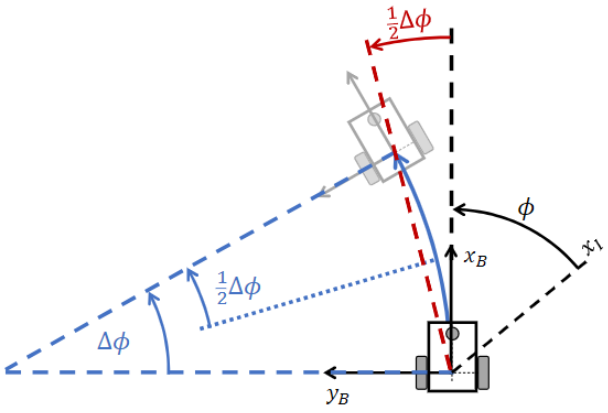


Figure 33: Robot body movement in the inertial frame: geometric analysis. Sourced from ELEN90090, L08- "Odometry", P. Beuchat (2023)

B.2 Derivation of Wheel Odometry update equations The aim is to determine the relative changes in robot pose over a small time interval Δt from $t-1$ to t over which the wheel encoder counts were incremented. We can then accumulate these relative changes to get the absolute pose in the inertial frame. Given the assumptions mentioned in the Preliminary report on Wheel Odometry (e.g no slip of wheels) and assuming that the angular velocity of the wheels remains constant over that time interval, the robot moves along an arc. As shown in Figure 32 and Figure 33 we define these small changes over Δt:

- Left and right wheel angular displacements: $\Delta\theta_{l,t}, \Delta\theta_{r,t}$
- Arc length path traced out by the robot: Δs_t
- Heading angle: $\Delta\phi_t$

Given that the counts per revolution is 1120 (with directional encoding as detailed in the Wheel Odometry Preliminary report), the conversion to angular displacement in radians for either wheel from the number of encoder counts $\Delta\theta_{counts}$ is calculated using

$$\Delta\theta_t = \frac{2\pi}{1120}\Delta\theta_{counts} \tag{5}$$

Given known values b, r and $\Delta\theta_{l,t}, \Delta\theta_{r,t}$ from above:

$$\Delta s_t = \frac{r\Delta\theta_{l,t}}{2} + \frac{r\Delta\theta_{r,t}}{2} \tag{6}$$

$$\Delta\phi_t = \frac{r\Delta\theta_{r,t}}{2b} + \frac{r\Delta\theta_{l,t}}{2b} \tag{7}$$

Thus the updated heading angle is

$$\phi_t = \phi_{t-1} + \Delta\phi_t \tag{8}$$

By geometric analysis as shown in Figure 33, the angle between the inertial frame axis x_I and the line joining the start and end points is $\phi + \frac{1}{2}\Delta\phi$. For small time intervals, Δs_t can be approximated by a hypotenuse in a right angled triangle with the two smaller sides parallel to (x_I, y_I), thus the change in x and y positions in the inertial frame are

$$\Delta x_{p,t} = \Delta s_t \cos(\phi_{t-1} + \frac{1}{2}\Delta\phi_t) \tag{9}$$

$$\Delta y_{p,t} = \Delta s_t \sin(\phi_{t-1} + \frac{1}{2}\Delta\phi_t) \tag{10}$$

Thus the updated position is

$$x_{p,t} = x_{p,t-1} + \Delta x_{p,t} = x_{p,t-1} + \Delta s_t \cos(\phi_{t-1} + \frac{1}{2}\Delta\phi_t) \tag{11}$$

$$y_{p,t} = y_{p,t-1} + \Delta y_{p,t} = y_{p,t-1} + \Delta s_t \sin(\phi_{t-1} + \frac{1}{2}\Delta\phi_t) \tag{12}$$

C Tuning proportional motor controller

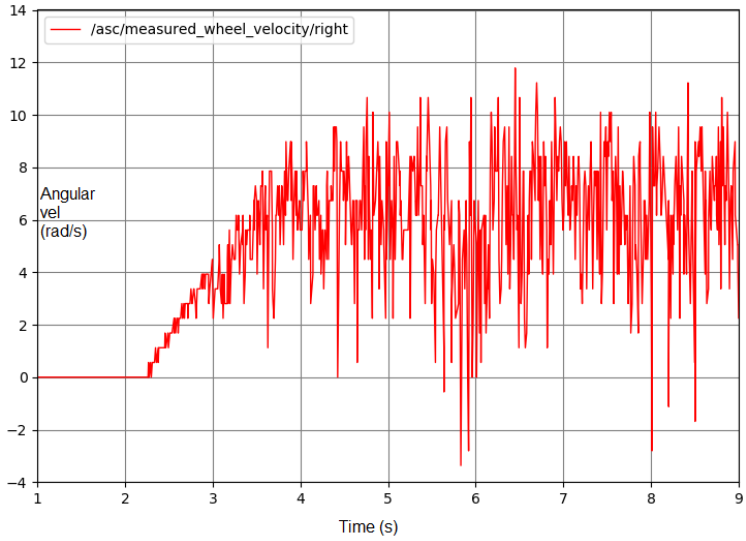


Figure 34: 1. Starting with a high proportional gain P=40

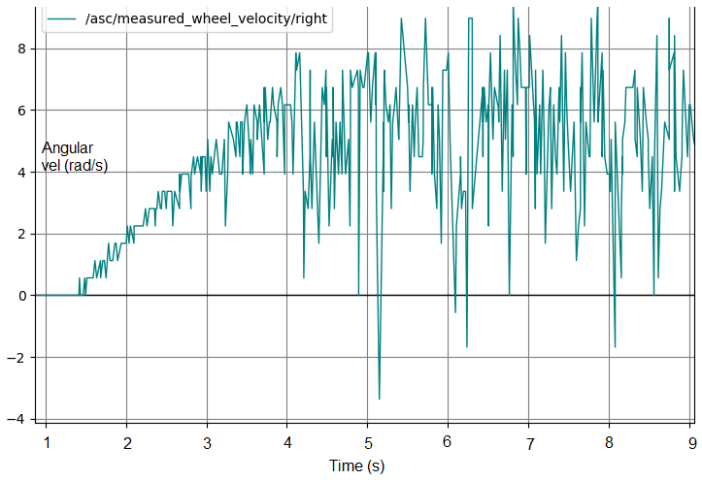


Figure 35: 2. Halving the gain P=20

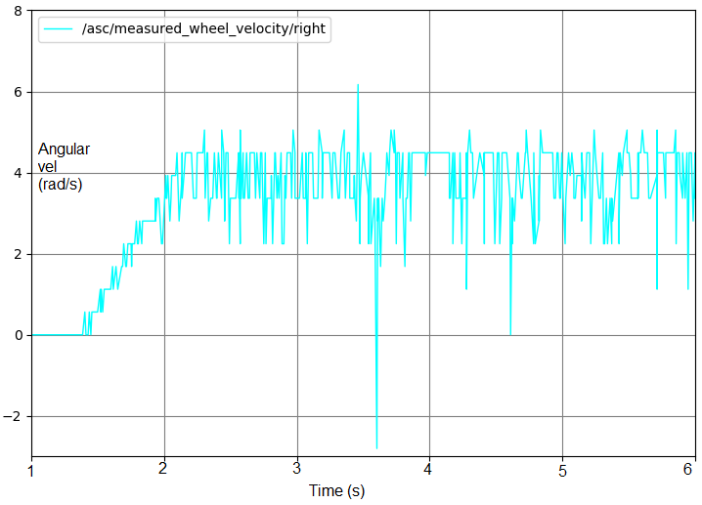


Figure 36: 3. Trying a low proportional gain P=5

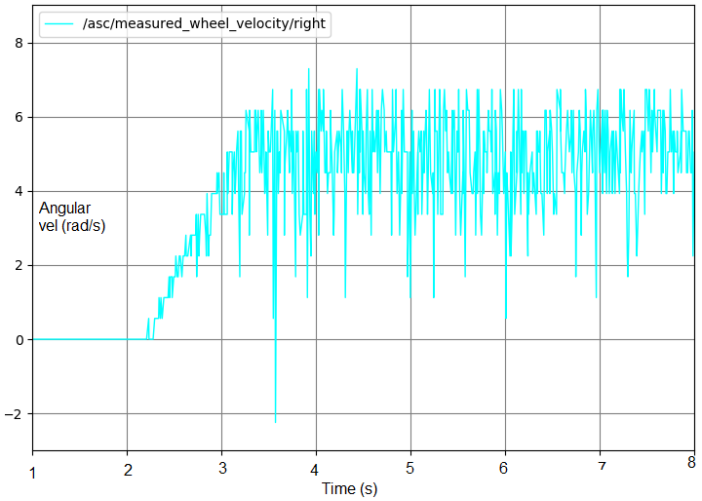


Figure 37: 4. Doubling to find a sweet spot P=10

D Samples of object mislabels & misses

Figure 38: One of the plants being mislabelled due to motion blur and backlighting

Figure 39: Plants being mislabelled due to motion blur

Figure 40: Plants being mislabelled, as only the pot is visible

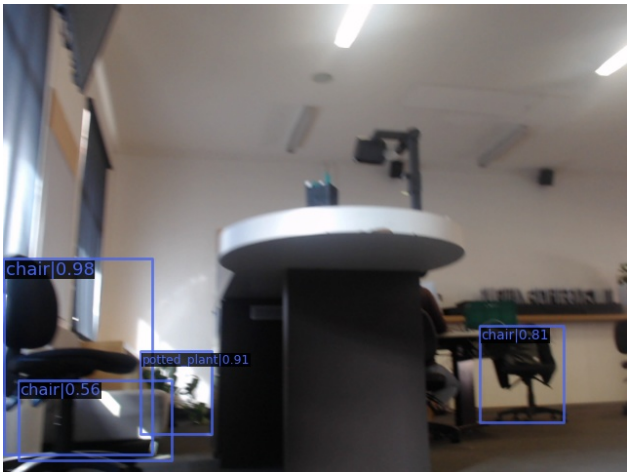


Figure 41: A desk that is not detected due to irregular shape

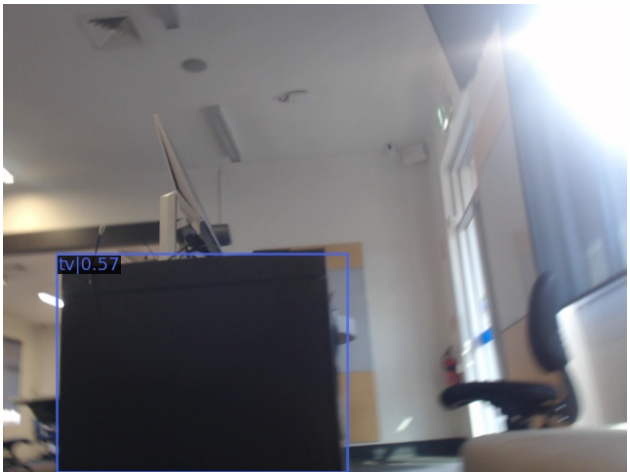


Figure 42: A desk that is mislabelled as `TV`, and an unlabelled chair

Figure 43: Object detection under backlighting. The chairs where the detected people were sitting on was not detected

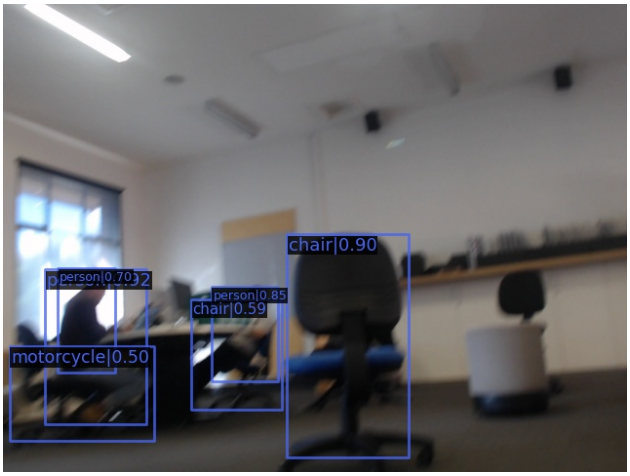


Figure 44: Few chairs are not detected or mislabelled due to motion blur

E Team Management Documentation

E.1 Task Allocation and Progress Tracking

Meeting minutes and task tracking for team members.

Week	Tasks done	To-do
1		Sign up as a team
2	Team signed up	Brainstorm what can be done
3	Briefing some directions	- Continue brainstorming - Phu trying voronoi diagram
4		Sustainability assignment
5		Take charge of subsystem for prelim report - Phu: aruco detection - Ernest: trajectory following - Khoi: object detection
6	Ongoing research and implement for prelim report	Continue on developing subsystem for prelim ROS technical preview
7	Separate tasks can work on the robot	Review each other's report Prelim report due
8	Prelim done!	- Phu look for path planning alternatives - Ernest test odometry with aruco - Khoi try implement yolo, look for faster alternatives Midsem
9	Yolo implemented at 10fps Detection info for plant and obstacles is done	- Phu set up RRT or RRT* - Ernest try kalman filter - Khoi test the plant alignment
10	Rosboard UI done Success plant alignment RRT* implemented Kalman not work in time	- Phu finalize aruco set up - Ernest make state controller, look for kaman alternative - Khoi add streaming topics for rosboard
11	Streaming object detection, map Speed adjustment done State controller still needs review	- Phu modify state controller - Ernest tune robot speed - Khoi complete zone detection for plant and status writer, modify state controller Starting testing all tasks together for demo
12	Not fully functional Problems at nighttime testing	- Phu debug the waypoint following, halve down waypoints - Ernest help Phu - Khoi simplify object detection and camera operations Finalize, prepare for demo on friday!
SWOTVAC	Demo not success	DEBUG! Retry demo
Exam 1	Demo done	Gather all ideas for final report before Friday
Exam 2		- Phu, Ernest finalize ideas before weekend - Khoi finalize report Report due
Exam 3		- Phu, Ernest film and finalize video Video due

Table 7: Meeting minute table, with each row updated incrementally on the Friday of every week. Team members have 1 brief (15-minute) meeting every week up to Week 8, and 2 to 4 meetings of varied time from Week 9 up to SWOTVAC, when demonstration is done. After that, task tracking is done via direct communication (i.e. messaging on social media)

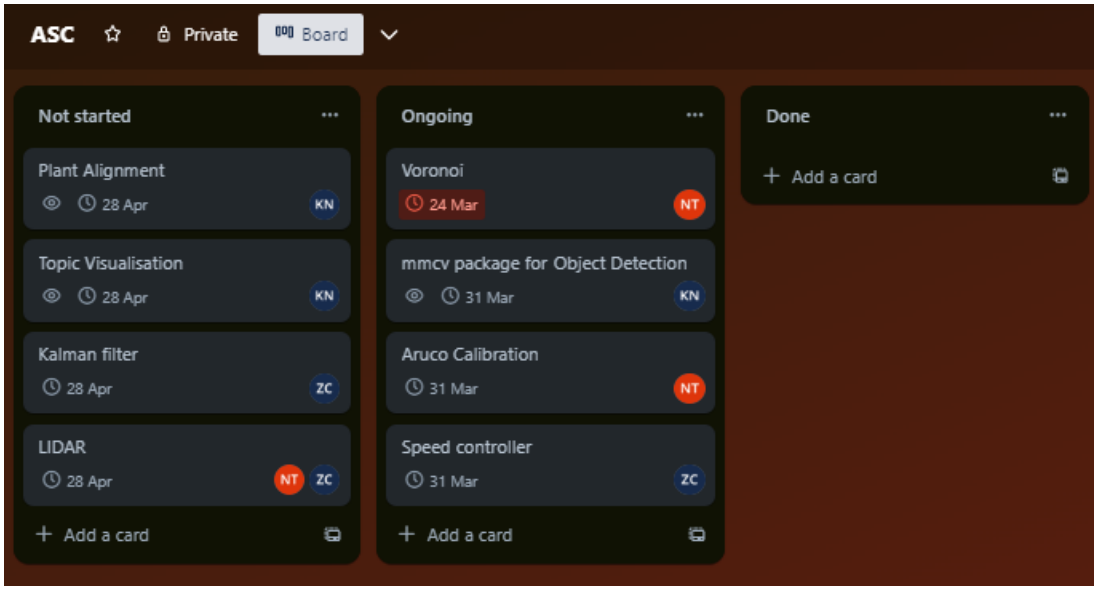


Figure 45: Trello was used for high-level task tracking, as well as idea generation. Every week, status of each card will be updated by the assigned member. The image was taken on week 5, when this board was first created and discussed

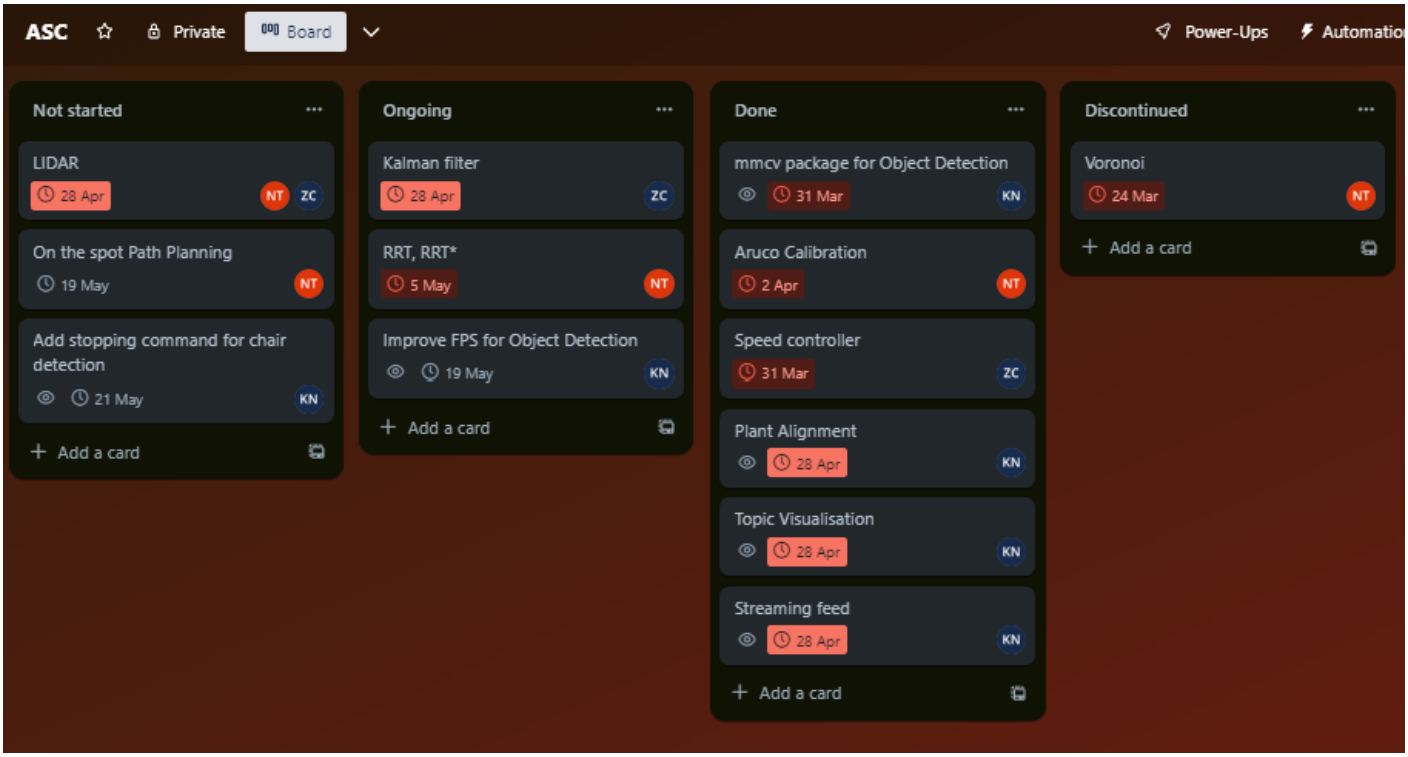


Figure 46: Trello was used for high-level task tracking, as well as idea generation. Every week, status of each card will be updated by the assigned member. The image was taken at the end of week 8, when all lectures have been delivered and development for robot demonstration started

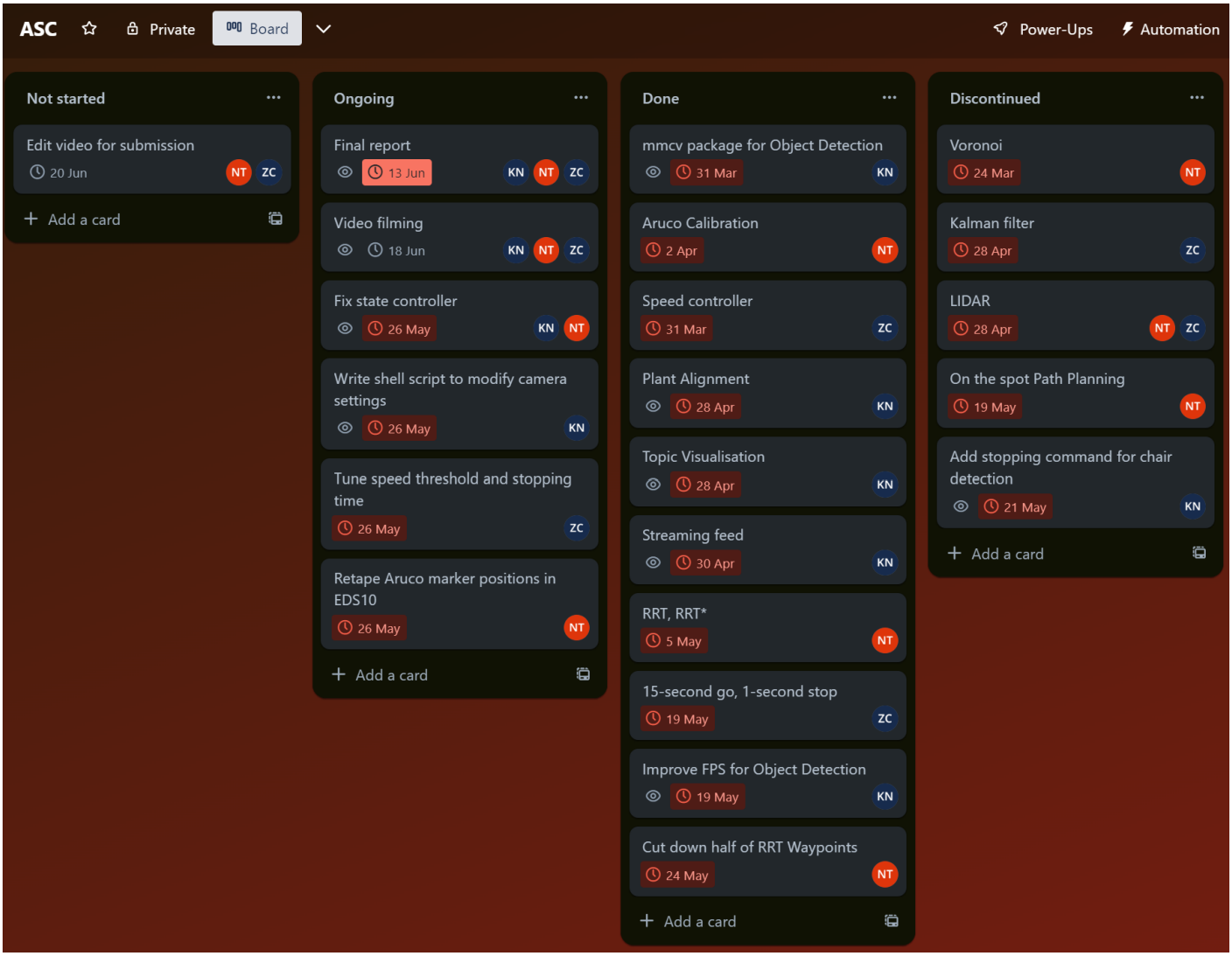


Figure 47: Trello was used for high-level task tracking, as well as idea generation. Every week, status of each card will be updated by the assigned member. The image was taken 1 day before the demonstration, where most of robot design have been finalized

Task	**Old deadline**	**New deadline**	**Reason**
Finalize subsystem allocation	Week 3	Week 4	URT competition
Integrate state controller with object detection	Week 11	23/5	Phu and Ernest have Robotics demonstration
Integrate path generation with path following	Week 11	23/5	Phu and Ernest have Robotics demonstration
Finalize design for demonstration	26/5	6/6	Demonstration failed

Table 8: Deadline amendments were also kept track to a table

E.2 Management of critical information Storing of important details for this project, such as architecture diagrams and code. Some details such as state machines, were discussed on paper and iteratively edited on for ease of conversation.

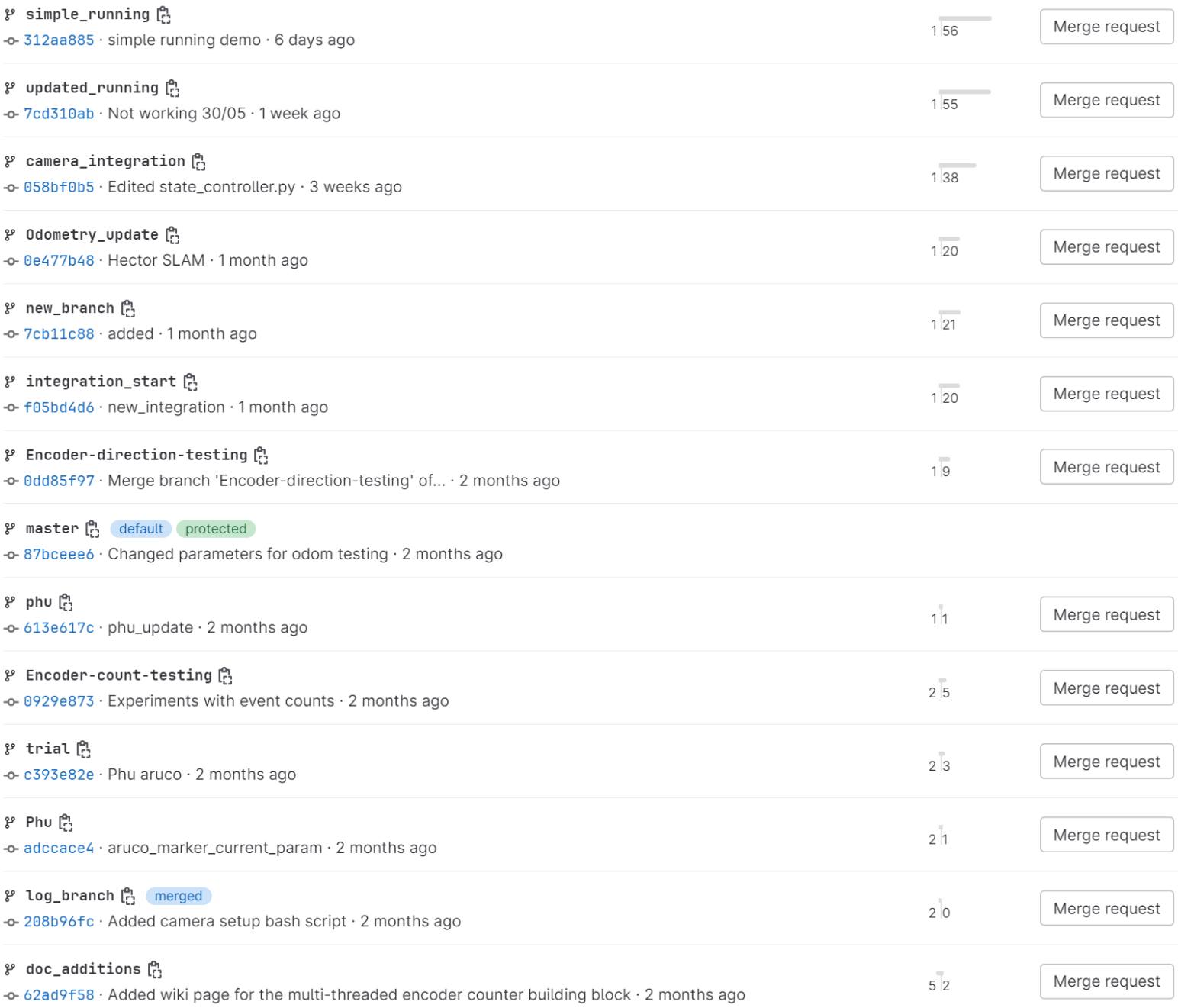
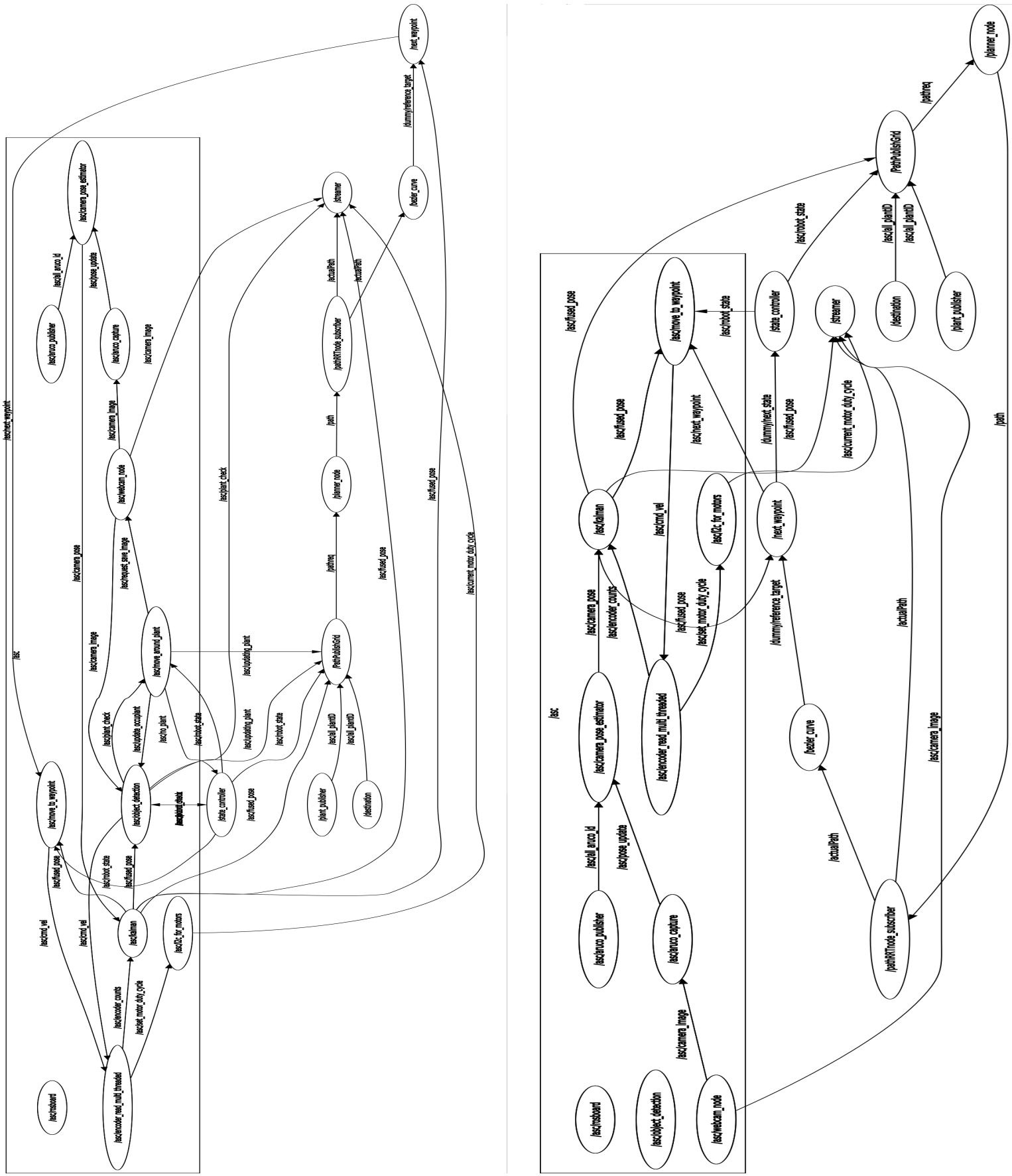


Figure 48: List of all code branches in git. To ensure proper version control, and code can be stored for different purposes (e.g. testing functionality expansion, demonstration), different branches are created for different purposes, and is merged via pull requests when needed. 2 main branches were used in this project: `updated_running` is where all functions are enabled but not yet working properly, and `simple_running` is where the simplified working version of the robot is stored for demonstration

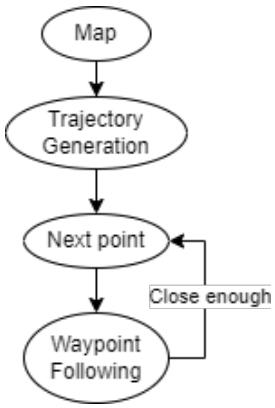
Figure 49: Git development graph for `updated_running` branch - the branch that contains all developed functionalities of the robot, displaying branch developments and commits in bottom-up chronological order



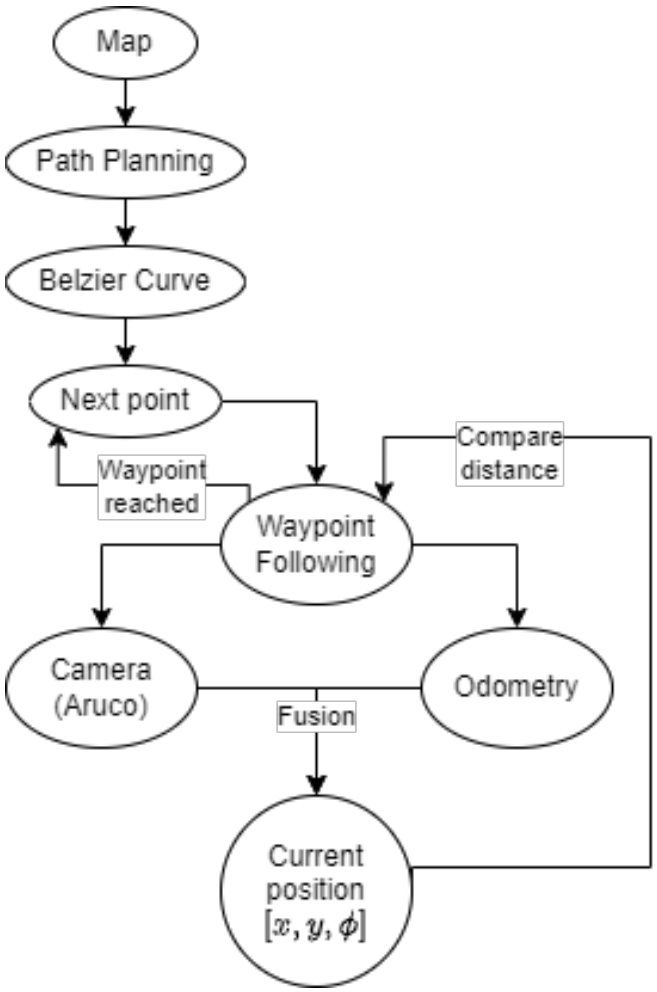
(a) Fully functional state, where all developed functions are linked

(b) Simplified working version that was used in demonstration

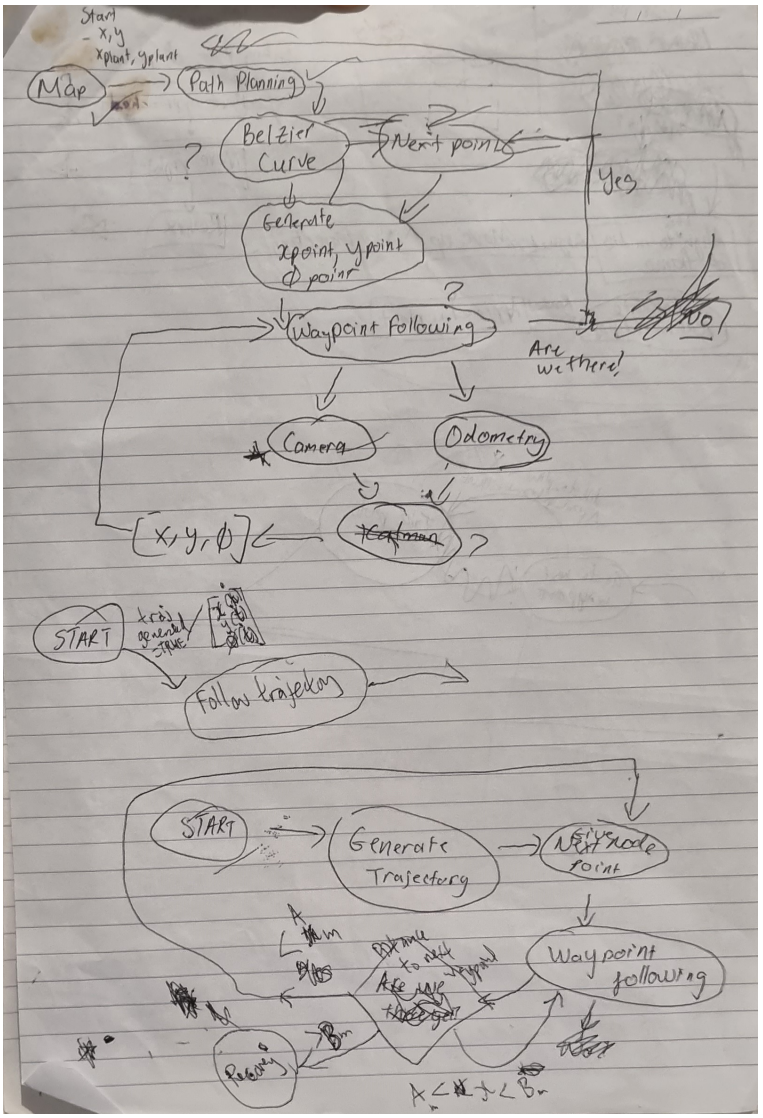
Figure 50: 2 versions of ROS topic graphs during the robot's final stage of development



(a) Diagram in week 6



(b) Diagram in week 8 (Fusion method was 'Kalman filter' in week 8, and later changed to 'Fusion cycle' in week 10)



(c) Hand-drawn version of the flowcharts

Figure 51: Development of path generation and path following logic

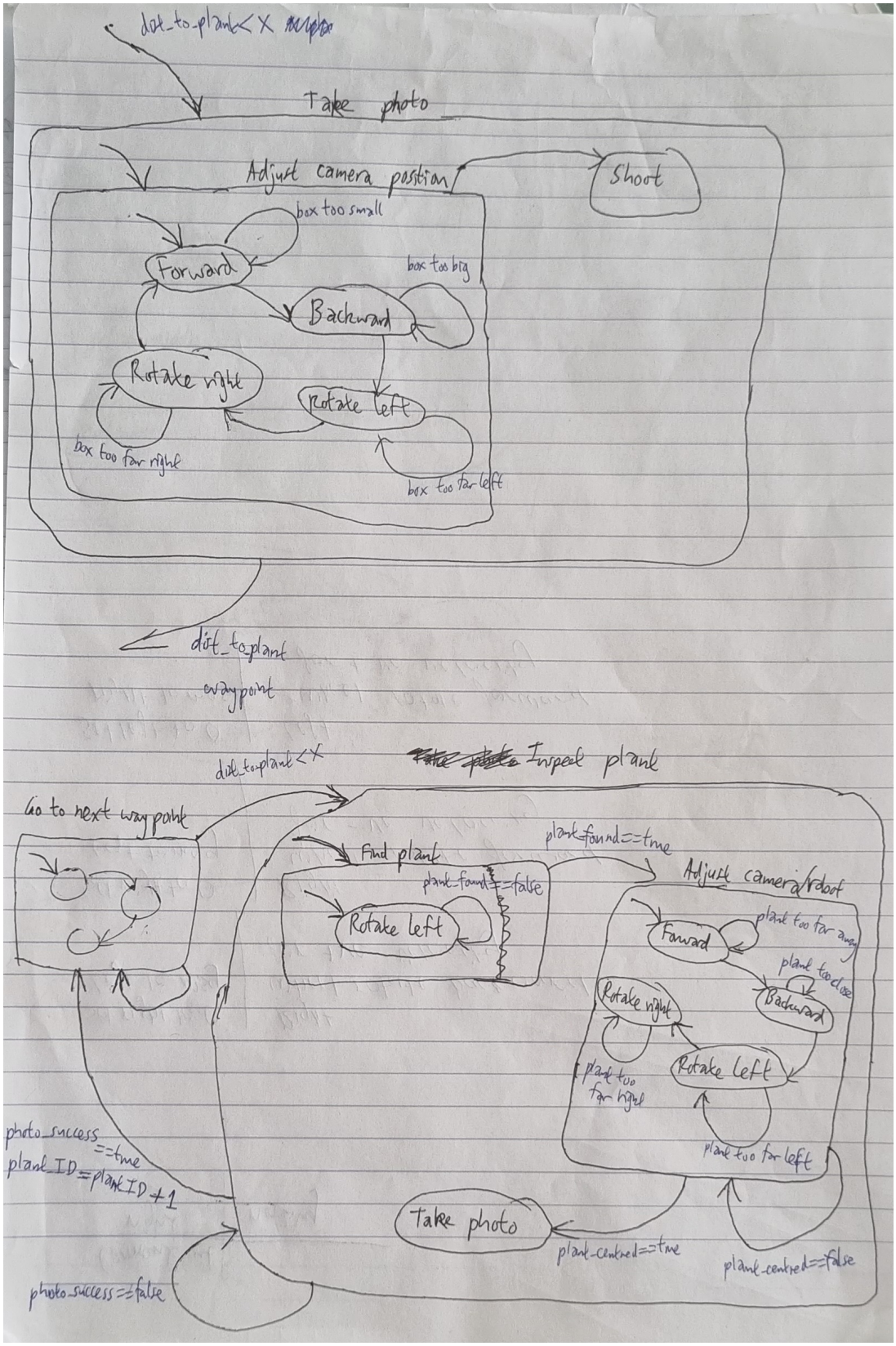


Figure 52: Hand-drawn sketch of finite state machine for plant approaching and photo taking. Upper diagram: initial development in Week 7. Lower diagram: further development of the functionality, and how it will be send and retrieve information to other nodes, made in week 9

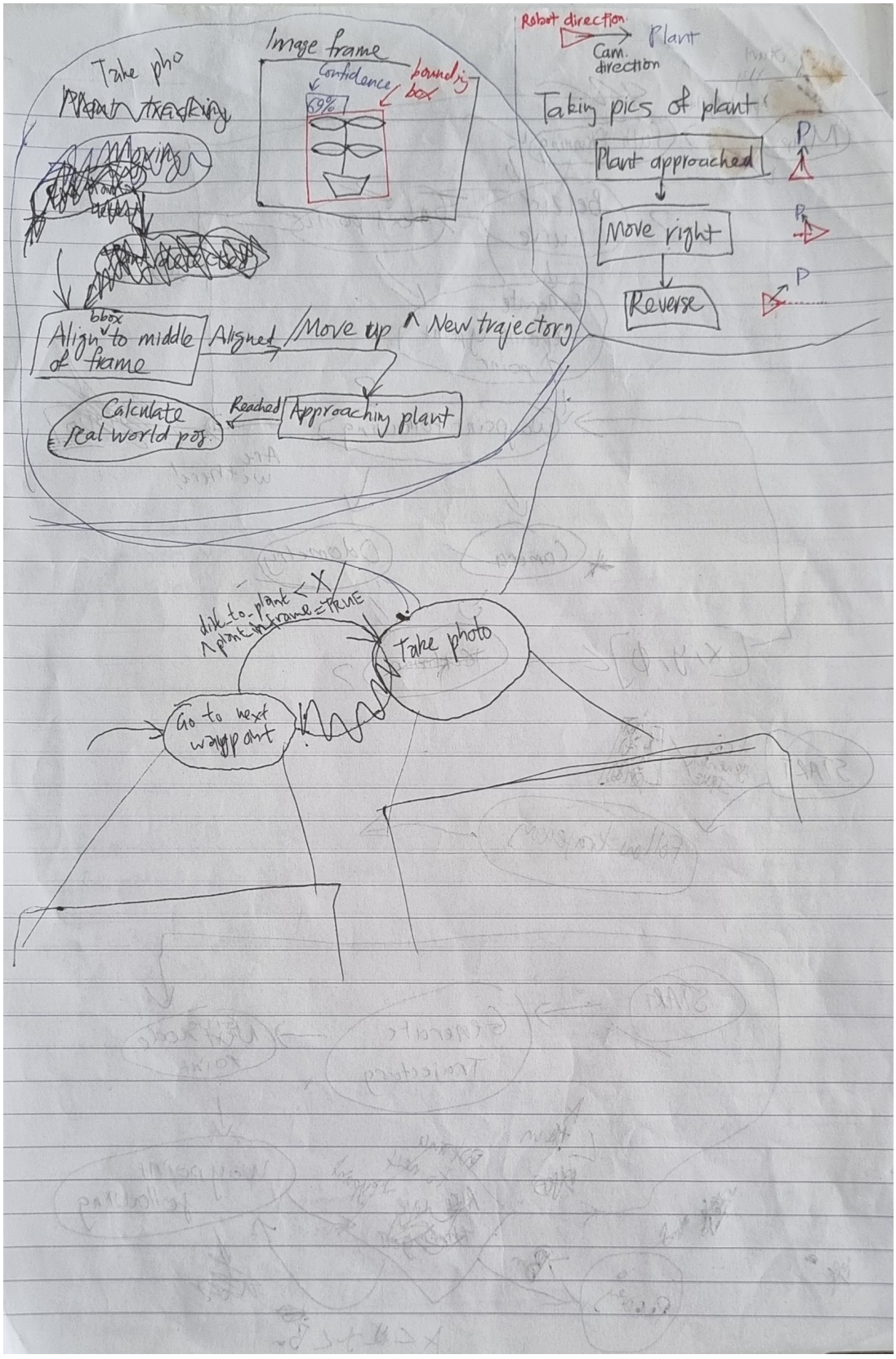


Figure 53: First drafted state machine of how to approach a plant drawn in week 7. An initial plan (top right) was suggesting the robot to move a fixed path upon reaching the plant to take pictures from 3 different angles

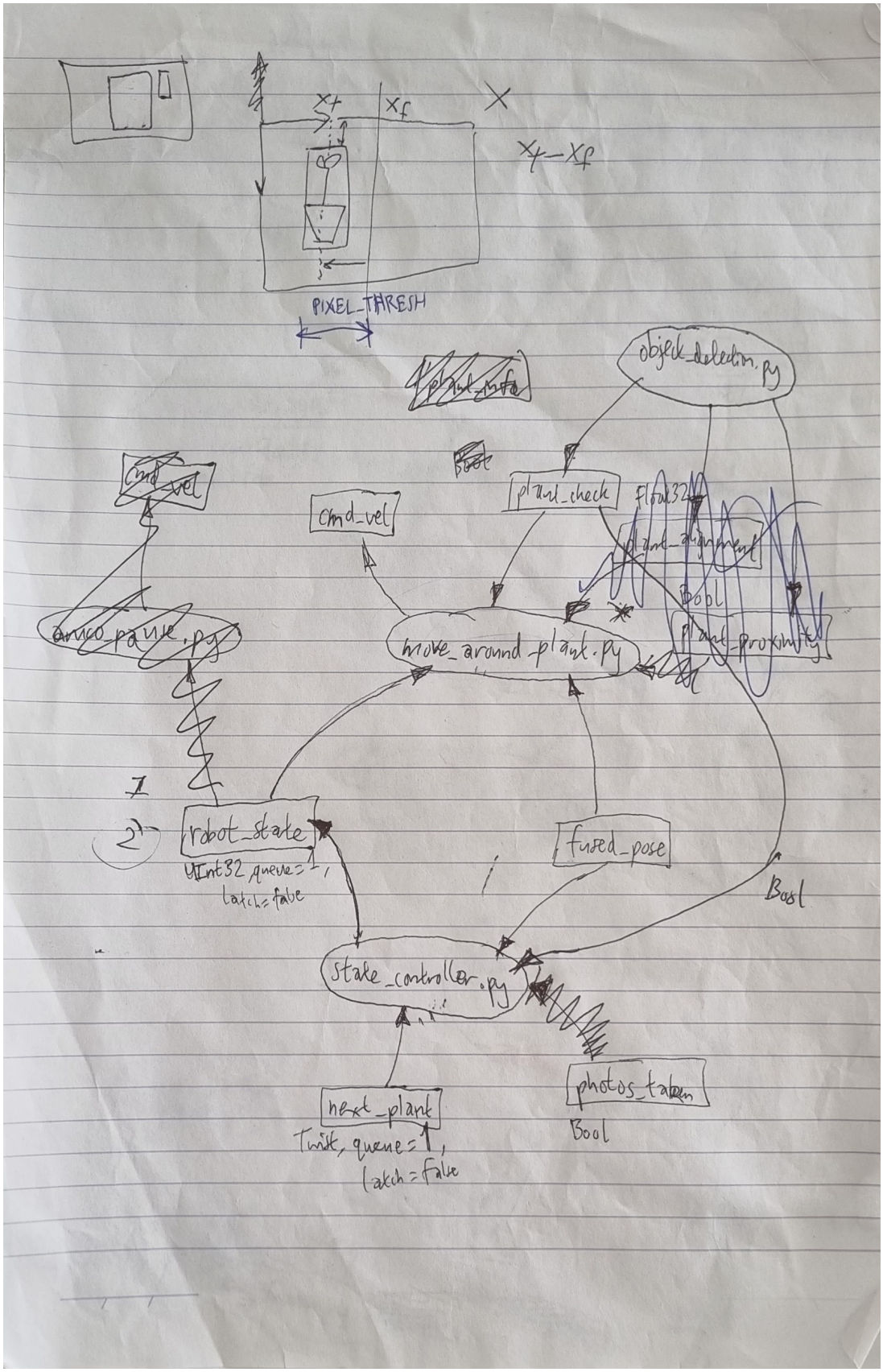


Figure 54: Second drafted state machine of how to approach a plant drawn in week 9, showing how this would integrate with other ROS nodes

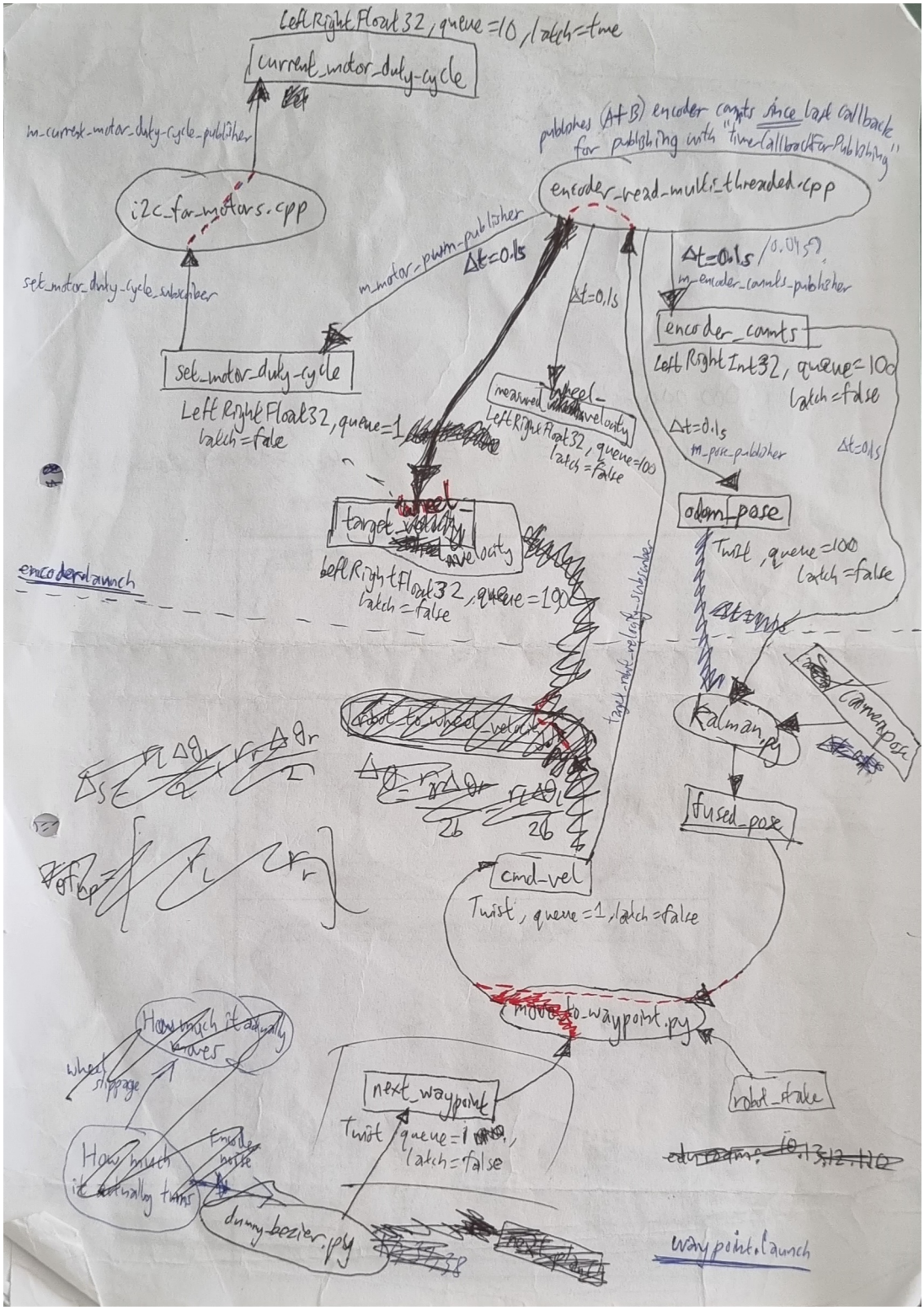


Figure 55: Drafted state machine of velocity control and waypoint tracking, iteratively drawn on from week 7 to week 10

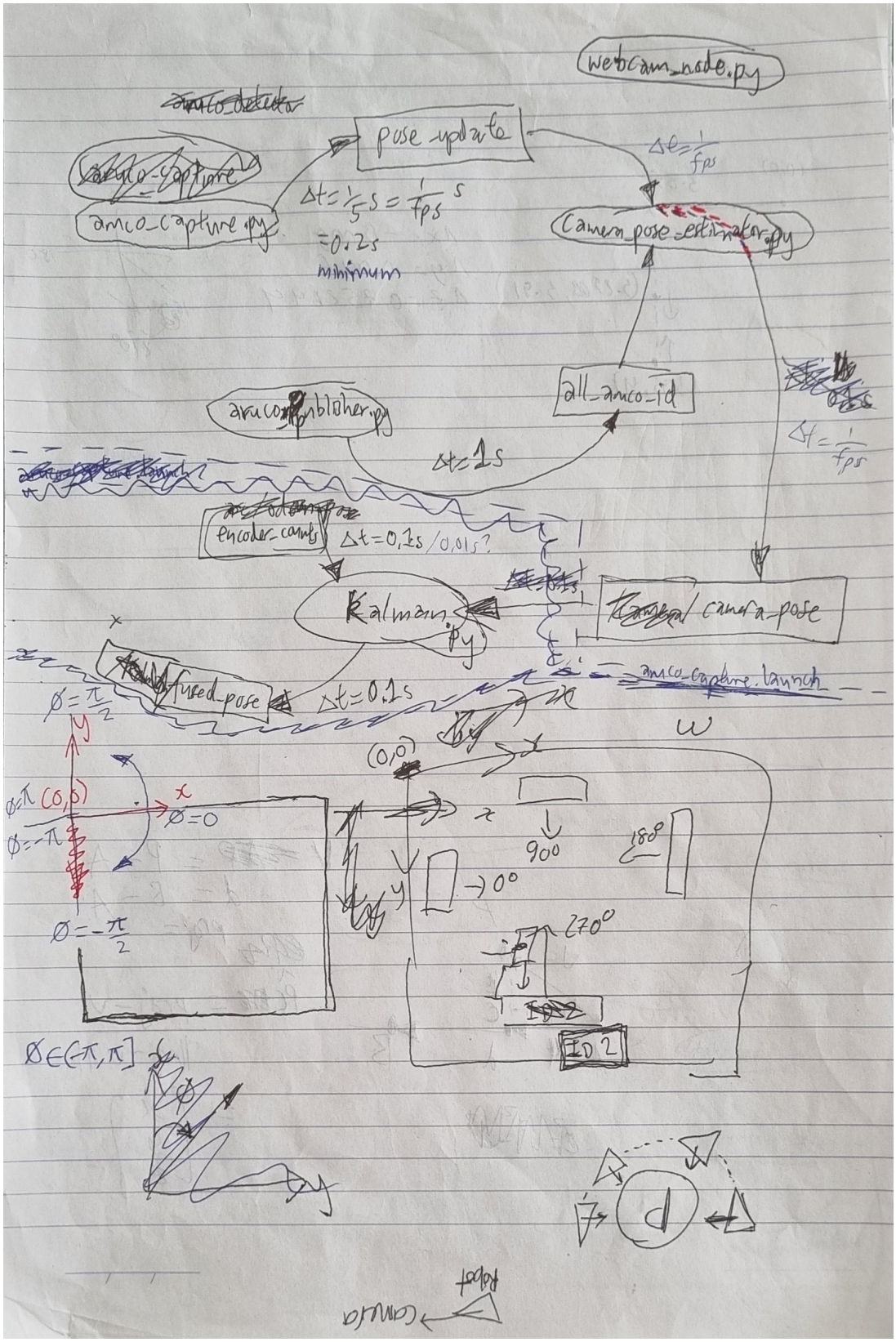


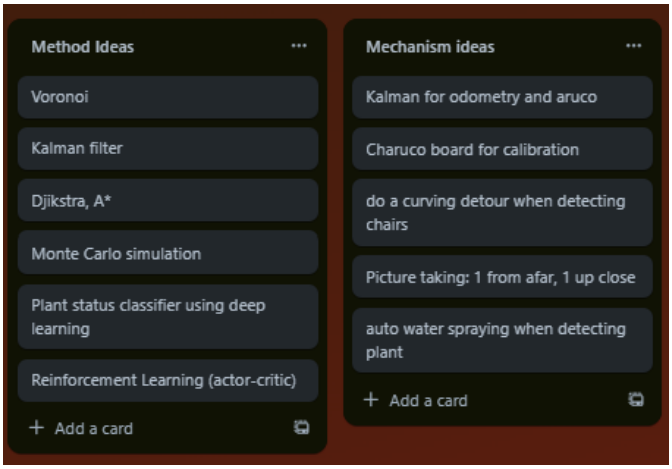
Figure 56: Drafting sketch for robot pose estimation. Upper diagram is the state machine (drawn in week 8), the bottom diagrams shows the finalised pose convention to be used in the course (drawn in week 9)

E.3 Innovation and team building strategies

Strategies made that utilise team interaction.

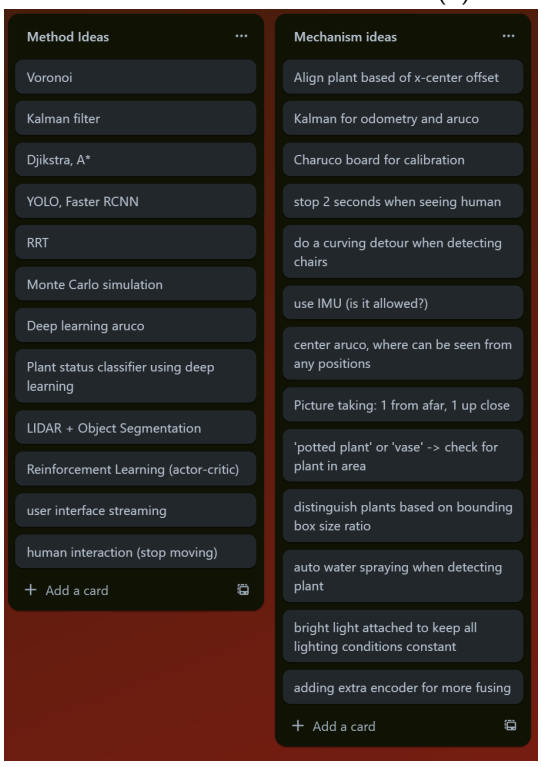
E.3.1 Idea brainstorming

In weeks 5, 8, 10, during weekly meetings, teams members will also try to come up with ideas on what algorithm can be used, what way can the robot perform a task, or what extra functionality can be added. The ideas are stored on Trello. On implementation stages (week 6 - 1 week before preliminary report and week 10 - 2 weeks before demonstration), the list will be revised, and selected ideas will be assigned to members and move to tasks section like in Figure 47.



(a) Accumulated brainstorming ideas in week 5

(b) Accumulated brainstorming ideas in week 8



(c) Accumulated brainstorming ideas in week 10

Figure 57: Results of 3 brainstorming sessions

E.3.2 Team protocol An initial meeting was done in week 2, where team members agreed on a general working protocol:

- Anytime one finishes work, keep everyone updated of the status.
- When interfering with another teammate's code of function, make sure to let them know ASAP.
- Any problem observed in teamwork should be communicated ASAP.
- Any interesting ideas should be put on both communication channel and Trello.
- Commit any progress to git before heading off, and leave meaningful comments.
- Team meeting will be done weekly to keep in touch.
- Fortnightly, try to make a brainstorming session (due to commitment limitation, we were not able to follow this closely, as seen in the previous session).

E.3.3 Team evaluation With the opportunity to reflect teammates in week 8 peer review assessment, team members spent 1 session to sit down and evaluate each other, as well as giving advice to make the teamwork process better.

Member	Strength	Weakness
Khoi	- Take initiative in many tasks - Come up with many ideas - Keep everyone in the team engaged	- Needs to work at faster pace - Should prioritize 1 job
Phu	- Task focused - Good background experience with ROS	- Needs more effective communication - Know how to assign work to teammates would be nice - Needs to know how to ask for help - Should be more flexible in working
Ernest	- Hard-working - Good communication	- A bit shy - Should not be afraid to state out an observed problem

Table 9: Concluding table from the team discussion on Peer Review task

E.3.4 Post-demo Evaluation After successful demonstration, on the 2nd of June (Week 1 of Exam), team members have a discussion together to evaluate the effectiveness of teamwork. The following main ideas were agreed upon:

- All team members did well in terms of coming up with ideas.
- Team members were a bit down on wrong strategy: the team did not performed as well as other teams due to prioritising to just meet the baselines, expecting that not all teams would achieve the baseline (which in turn they did). Solution: be able to ask around the progress of peers to know the relative performance.
- A lot of inconsistency happened in namespace for topics. Solution: they should be discussed and agreed upon before proceeding.
- Communication is really important, especially when debugging a problem. Some problems that one member took so long to solve, ended up being solved in 5 minutes by another member, as they are experieced with that problem type.
- Some team members should be more initiative, and should not be afraid to ask for help. A lot of procedures in code inspection or set up could have been done faster if they asked their teammates who already know the process, instead of spending days looking up the internet.