ELEN90090 Autonomous System Clinic Project Report

Quang Trung Le
987445, quangtrungl@student.unimelb.edu.au

Hai Dang Nguyen
860308, hain4@student.unimelb.edu.au

Hoang Viet Pham
872135, hoangp2@student.unimelb.edu.au

Contents

List of Abbreviations **5**

1 Introduction and Motivation **6**

1.1 Background . . . 6

1.2 Objective . . . 6

1.3 Scope and Limitation . . . 6

2 System Analysis **6**

2.1 Robot Chassis . . . 6

2.2 Hardware Selection . . . 7

2.2.1 Sensors and actuators . . . 7

2.2.2 Computer module . . . 8

2.2.3 Loading mechanism concept . . . 8

2.3 Electronics and Communication Architecture . . . 9

2.4 System-level Design . . . 10

3 Design Procedure **12**

3.1 Path planning . . . 12

3.1.1 Reference paths . . . 12

3.1.2 Algorithm . . . 12

3.2 Motor Control . . . 15

3.2.1 Motor Modelling . . . 15

3.2.2 Open loop controller . . . 15

3.2.3 Closed loop controller . . . 16

3.3 Pose Control . . . 17

3.3.1 Kinematic Model . . . 17

3.3.2 Trajectory Tracking Modelling . . . 18

3.3.3 Pose Estimation Using Odometry . . . 19

3.3.4 Pose Estimation Using Camera . . . 20

3.3.5 Pose Controller . . . 21

3.4 Finite State Machine . . . 23

3.5 Mechanical Design . . . 24

3.5.1 Loading Mechanism Design . . . 24

3.5.2 Camera assembly . . . 25

3.6 Signal Processing . . . 26

3.6.1 Debouncing the Proximity Sensor . . . 26

4 Testing & Implementation **27**
4.1 Functional Test . . . 27
4.1.1 Open-loop angular velocity control . . . 27
4.1.2 Encoder monitors . . . 28
4.1.3 Pose estimator using Odometry . . . 28
4.1.4 Closed loop angular velocity control . . . 28
4.1.5 Pose estimator using Camera . . . 28
4.1.6 User command acknowledgement . . . 29
4.2 Implementation . . . 29

5 Validation **29**
5.1 Reachability Analysis . . . 29

6 Discussion **30**
6.1 Observation . . . 30
6.1.1 Motor control . . . 30
6.1.2 Pose control . . . 30
6.1.3 Pose estimation using camera . . . 31
6.2 Limitations . . . 31
6.3 Further Improvements . . . 31

7 Conclusion **32**

Appendices **33**

A ROS Architecture **33**

B Fine Tune Velocity Control **35**

C Camera Calibration **36**

D Transformation Matrix for Markers **38**

E Euler Angles for R_b^g **39**

List of Figures

1 Robot assembly and modelling . . . 7
2 Hardware Configuration on Robot chassis . . . 9
3 Schematic of electrical connections between the hardware components of the robot . 10
4 System-level design diagram . . . 11
5 Warehouse modelling . . . 12

6 Stages in development of path-planning algorithm 13
7 Comparison of two path-planning algorithms 14
8 Simplified model of DC motor 15
9 Actual and modelled rotation angle of two motors at |U(t)| = 1.2V 16
10 Open loop control response 16
11 Simulation result in Angular Velocity Response 17
12 Kinematics of the robot [11] 17
13 Pure rotation 19
14 Coordinate system in relative with warehouse map 20
15 Closed loop system 21
16 Pose Control block diagram 22
17 Step Response of the pose control 23
18 Finite State Machine 24
19 Container 3D model 25
20 Marker position in camera fov 25
21 Camera cover 26
22 Simulated output logic of Proximity Sensor 27
23 System response comparison 31
24 ROS architecture 33
25 Angular velocity comparison 35
26 Example of output image from camera calibration step 37

List of Tables

1 User requirements 6
2 Estimated internal parameters 15
3 Controller parameters 16

List of Abbreviations

AGV Automatic Guided Vehicle. 6

AI Artificial Intelligence. 8

AMR Autonomous Mobile Robot. 6, 8, 10, 32

CPU Central Processing Unit. 8

FSM Finite State Machine. 12, 23

GND Ground. 9

GPIO General Purpose Input/Output. 8, 9

GPU Graphics Processing Unit. 8

I2C Inter-Integrated Circuit. 10

ICR Instantaneous Center of Rotation. 17

MDF Medium-density fibreboard. 8, 9, 24

PCB Printed Circuit Board. 7, 9

PLA Polylactic acid. 9

PWM Pulse-Width Module. 8, 10, 17, 18, 34

ROS Robotics Operating System. 23, 29, 33

1 Introduction and Motivation

1.1 Background

The history of warehouse automation has started since the middle of 20 century. It plays an essential role in increasing the efficiency, reliability and speed, leading to great advantage to stay competitive.

Among the logistic segments, one of the most expensive fulfillment operations is picking and packing, which takes roughly 60% the operational costs. A range of technologies have been widely embedded in this field, such as good-to-person (GTP), Automatic Guided Vehicles (AGVs) or Autonomous Mobile Robots (AMRs). However, the sophisticated, flexible and cost-effective AMRs makes it become the priority for warehouse automation.

1.2 Objective

The project focuses on designing software and hardware for an AMR, based on a given robot platform. The functionality is to collect items from the dispensers and delivery to the receptacles. The research objective remains to achieve the user requirements (UR) in Table 1.

No.	Requirements
UR1	The collect-and-delivery (C&D) cycle is less than 5 minutes.
UR2	At least 1 type of item can be handled.
UR3	Robot does not access NO-GO zone and warehouse boundary.
UR4	The items are not dropped during delivery.

Table 1: User requirements

There is a note for user requirements that UR1 and UR2 play as the baseline in this project. The higher throughput is preferred (UR1) and many different types of items should be collected (UR2).

1.3 Scope and Limitation

The scope of this project is limited to the aspects outlined below:

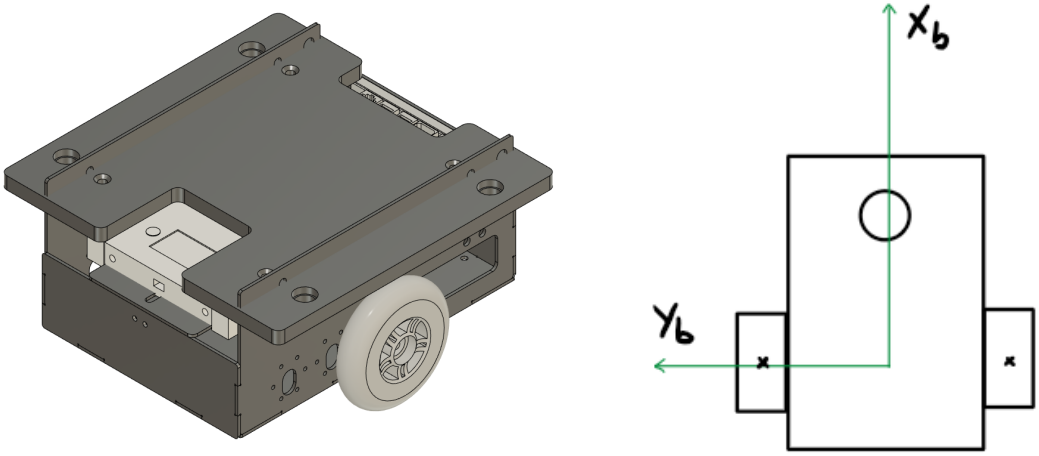
- The solution of robotic prototype deals with a variation of objects in two shapes/sizes and colour: stress ball with diameter 50mm and bouncy ball with diameter 25mm.
- A limited options on the electrical components can be allocated. The detailed list can be found in [1].
- The most significant application of AMRs is re-routing after detecting obstacles. However, the obstacle avoidance is out of scope in this project.
- The robot manipulator to actively grasp the items is not considered. In stead, the items are manually loaded onto the “loading mechanism” of the robot.

2 System Analysis

2.1 Robot Chassis

In vehicle based robotic application, there are a wide range of robot configuration such as: 2, 3, 4, 6 differential wheels of omni, mecanum, conventional types. The 2 differential wheel configuration

is often equipped with 1 or 2 passive wheel(s) of caster ball and caster wheel.



(a) Robot assembly (b) Robot modelling

Figure 1: Robot assembly and modelling

The robot chassis in this project is the most basic configuration (2 differential wheels and 2 caster balls), as can be seen in Figure 1a. This configuration makes the control strategy and additional hardware setup simpler in comparison with other models, while the slippage in omni and mecanum wheels can be avoided. However, there will be one redundant passive wheel while driving in case of inaccurate setup. Besides, some other mechanical setups include two 100mm diameter wheel, 250mm wheel base.

Note: Although the chassis configuration is 2 driving wheels 2 caster balls, it can be modelled as 2 driving wheels 1 caster ball, as can be seen in Figure 1b. The reason is that they share the same specifications (for example, the robot center locates at the midpoint of 2 wheels' centers) and the same behaviors.

2.2 Hardware Selection

2.2.1 Sensors and actuators

Driving motors

The 2 differential wheels are driven by 2 servo motors **Pololu 50:1 Metal Gear Motor (37Dx70L with 64 CPR Encoder)**. These motors consume 12 Volts electricity and generate no-load angular velocity 200 rpm. The attached encoder resolution is 800 cpv when monitoring single channel single edges or 3200 cpv when monitoring both channels both edges [2].

Proximity sensor

A sensor is required to detect whether an object is present in the container. The sensor would be polled each time an object is loaded or unloaded. The Polulu Carrier PCB, embedded with a Sharp GP2Y0D810Z0F [3] digital distance measuring sensor was chosen for its availability and simplicity to operate. The sensor is to be mounted at the unload side of the container, scanning for any object docked at the revolving door. Notably, it features:

- Ranging capabilities from 2cm to 10cm, matching the width of the revolving door section.

- Its operation is based on an infrared LED (IRED) ranging technology, which should reliably detect most types of objects.
- The sensor supports a maximum polling rate of up to 400 Hz, largely sufficient for the its implementation in the project.
- Additional electrical characteristics include a low current draw, a wide range of supply voltages, and an active-low output voltage.

Camera

The provided Logitech C922 Webcam [4] allowed the robot to self-localise by detecting ArUco markers placed at known locations around the warehouse, using OpenCV. The camera supports a resolution of 1920 x 1080 pixels (1080p) at 30 frames per second as well as 1280 x 720 pixels (720p) at 60 frames per second. Notably, this particular webcam no longer features hardware encoding and is interfaced with a USB 2.0 connection.

Servo motor

The Polulu 1501MG [5] was selected to actuate the container revolving door. The operation is controlled by sending PWM signals to the servo driver. Features include:

- An operation voltage range from 4.8V to 6.0V. The servo motor will be supplied +5V from the servo driver.
- A rotational arc of up to 180 degrees, allowing the revolving door to either engage and block the balls or to disengage in the other position (rotated 180 degrees away), allowing the balls to roll out of the container.
- A stall torque of over $15.5kg - cm$, largely adequate to actuate the thin MDF-made door.

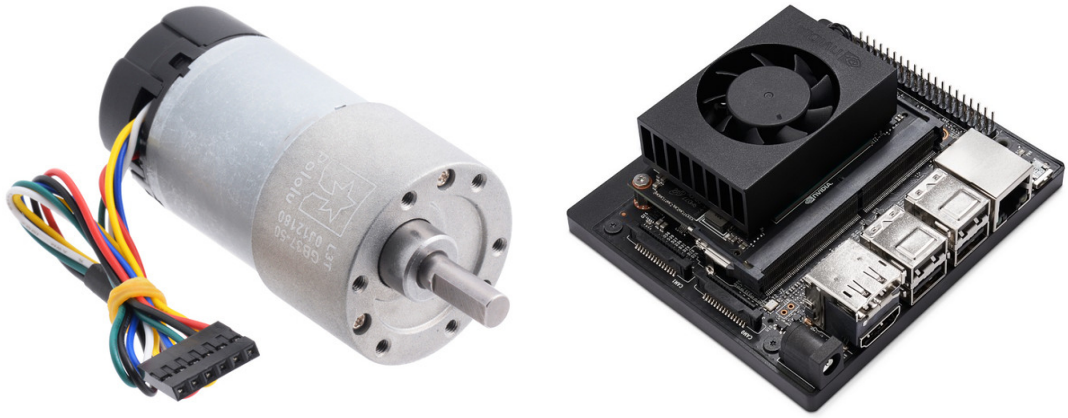
2.2.2 Computer module

The NVIDIA Jetson Xavier NX (Figure 2b) is used as the main computer of the robot. This is a power-efficient and compact module with dedicated CPU and GPU for full-featured, multi-modal AI applications, as future improvement of this AMR design. The 40-pin expansion header providing access to the I2C bus and GPIO pins of the Jetson.

2.2.3 Loading mechanism concept

For the concept of the loading mechanism these are different criterion taken into consideration including material, feasibility, reliability, integrability, ease of manufacturing, customisability. The objectives of the loading mechanism are:

- Light weight
- Minimal use of motors
- Take less time and easy to manufacture
- Able to be mounted on top of the robot
- Able to load and unload multiple items



(a) Pololu motor 50:1 Metal Gear Motor (37Dx70L with 64 CPR Encoder)

(b) NVIDIA Jetson Xavier NX module

Figure 2: Hardware Configuration on Robot chassis

Due to time constraint and hardware limitation, there are two types of material being considered throughout the design process: PLA (3D print) and MDF (Laser cut). The idea was to have a light-weight loading mechanism, which contribute minimal load on the robot's wheels, hence minimise the friction with the warehouse's surface during operation. In terms of weight, both PLA and MDF can be ideal for this application. However, the quality of 3D printing is one of the concerns as it introduces more roughness to the surface, especially for the slider which will be further explained in the nex part. Moreover, comparing the time for manufacturing and accuracy of the two, MDF laser cutting is definitely a better option than 3D printing, allowing for multiple design iterations, facilitating testing and developing process.

The mechanism is designed to take the advantage of the kinetic, potential energy and momentum generated by the items after being released from the dispensers due to gravity, which leads us to the concept of using an inclined plane. Having a roughness surface for the slider will definitely introduce more friction that resists the movement of those items, especially the stress ball, hence the items may not automatically be unloaded at the receptacles. Moreover, due to the assembly process with MDF laser cut, the slider can be easily replaced with a different mechanism suitable for different types of item.

2.3 Electronics and Communication Architecture

Figure 3 details the electrical connections between the hardware components of the robot. Notably, the type of connection and specific Jetson pin numbers are shown:

- Power lines V+ and GND supply each component with the required voltage to operate. The robot operates on two main voltage levels:
  - 12V, sourced from the 24,000 mAh Voltaic V88 external battery, powers the Jetson Xavier and the motor controller modules.
  - 3.3V, corresponding to the digital logic level, is supplied by several pins on the Jetson and is used by the different PCB modules such as the proximity and current sensors.
- GPIO pins, in particular, GPIO Pin 7 polls the output of the proximity sensor. Pins 29, 31, 33, 35 respond to events such as falling and rising edges and are utilized to read the wheel

encoder data.

- I2C, is a synchronous, serial communication bus. An "internal" I2C bus, embedded in the robot frame and connecting the Jetson and motor controllers, is differentiated from an "external" bus, which interlinks the remainder modules and sensors on the exterior of the robot.
- PWM pulses are used to control the wheel speed and the rotation of the servo-motor.

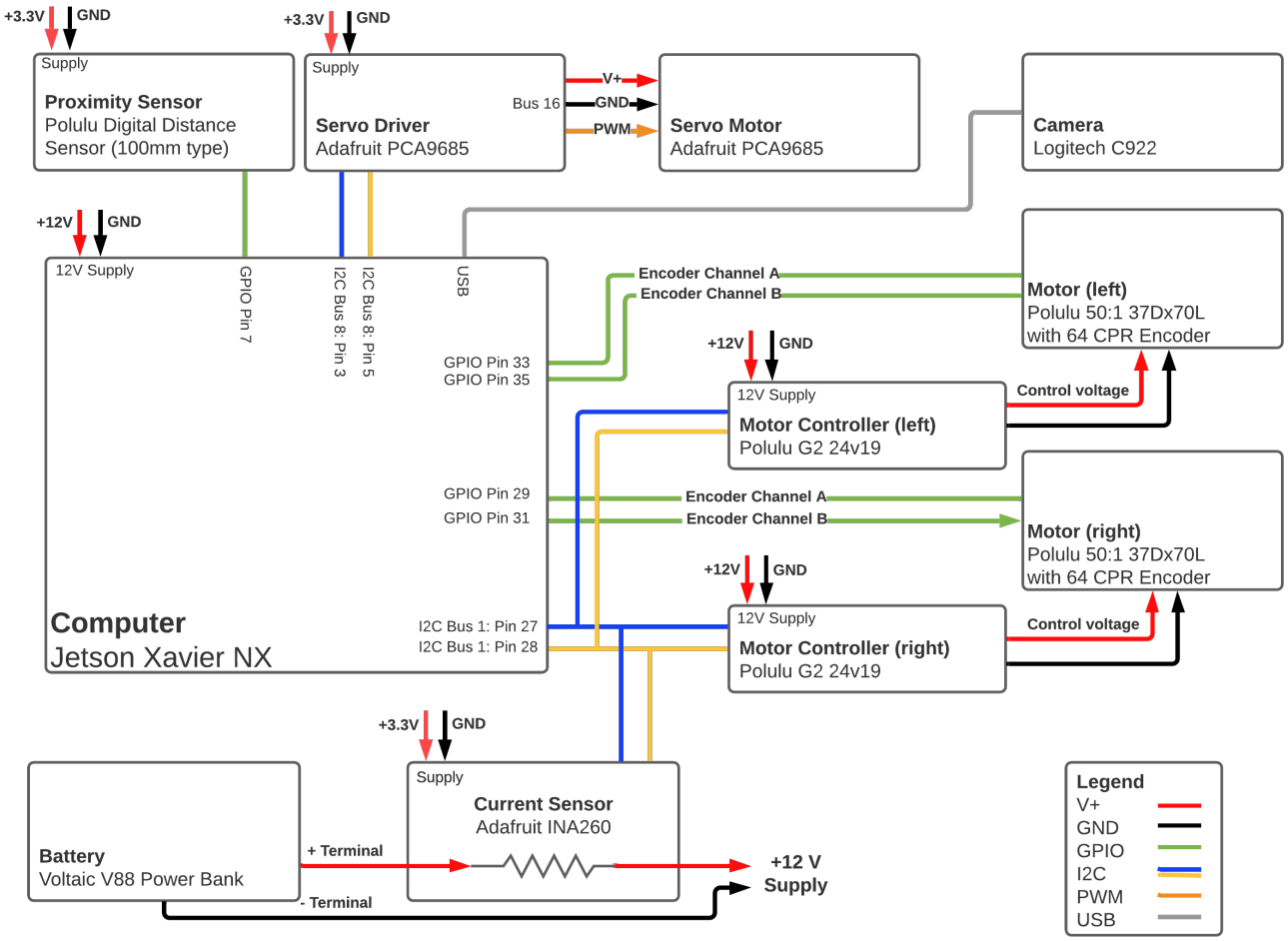


Figure 3: Schematic of electrical connections between the hardware components of the robot

2.4 System-level Design

Figure 4 presents the concept for operating the AMR. The main concept is having a dedicated main computer module that can run the main program autonomously and continuously. The main computer needs to be powerful enough to handle computer vision tasks, a part of sensing and localising system. The main program of the robot is designed as a loop of subsystems to perform a specific task. The task determination process takes the object and robot's pose data as an input to make decision on the next task to be performed. This process will also take user command as an initial request or interrupt action needed in emergency circumstances. Further detailed description of the system-level design is presented in the Project Plan.

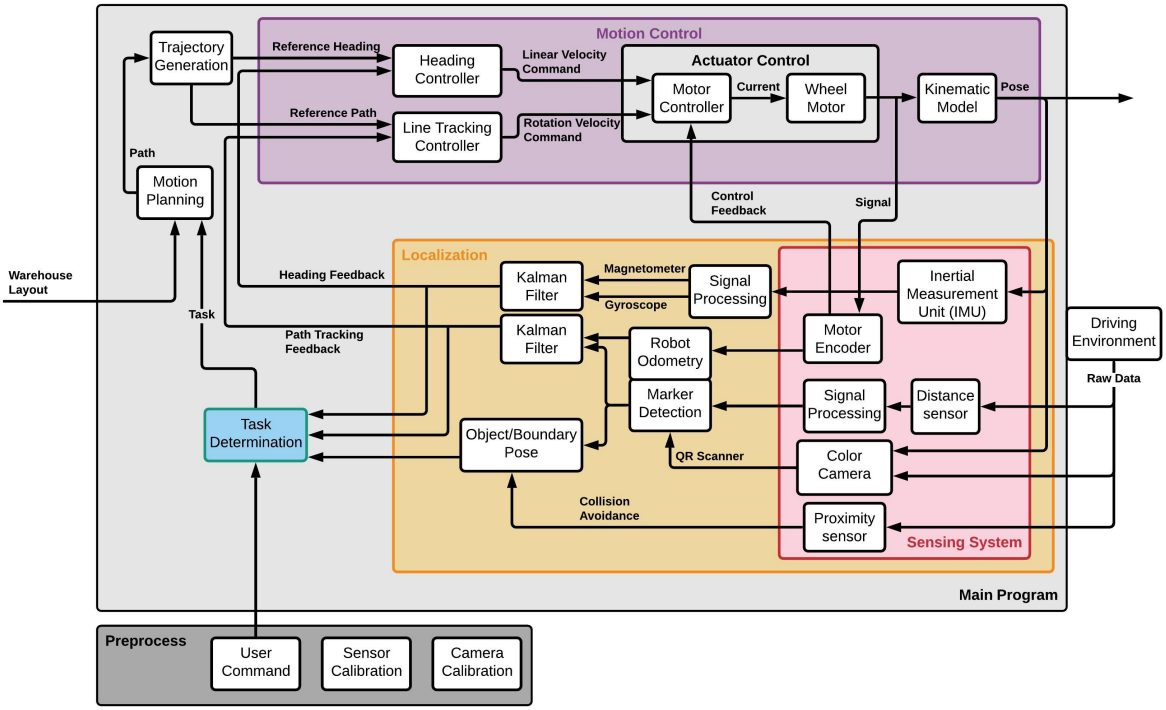


Figure 4: System-level design diagram

3 Design Procedure

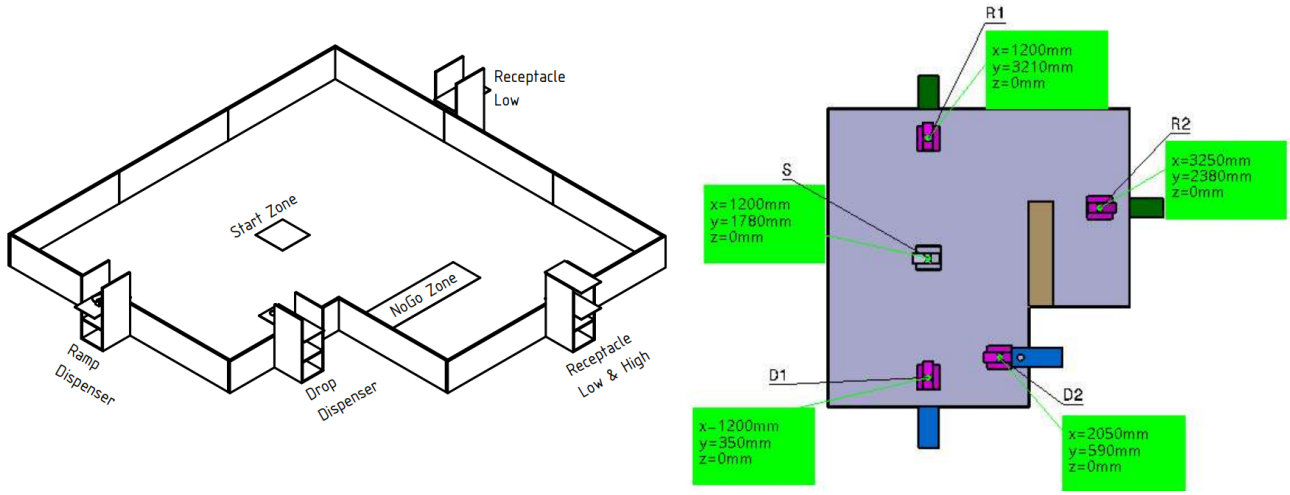
3.1 Path planning

3.1.1 Reference paths

Due to time and ability constraints, the design of the robot will focus on operation within the specific warehouse shown in Figure 5a. In order to finish one cycle, the robot needs to start from the start zone and go to the requested dispenser to pick up an item before delivering it to a receptacle. Then, the robot is required to go back to the start zone.

For convenience, the description of the following terms are provided

- Transit path: the path taken by the robot to get to a dispenser.
- Delivery path: the path taken by the robot from a dispenser to a receptacle.
- Return path: the path taken by the robot from a receptacle to the start zone.
- S, D1, D2, R1, R2: robot's end points, which is equivalent to starting point, ramp dispenser, drop dispenser, low receptacle and low-high receptacle respectively (Figure 5b).



(a) 3D model of warehouse

(b) Warehouse Layout with End-points

Figure 5: Warehouse modelling

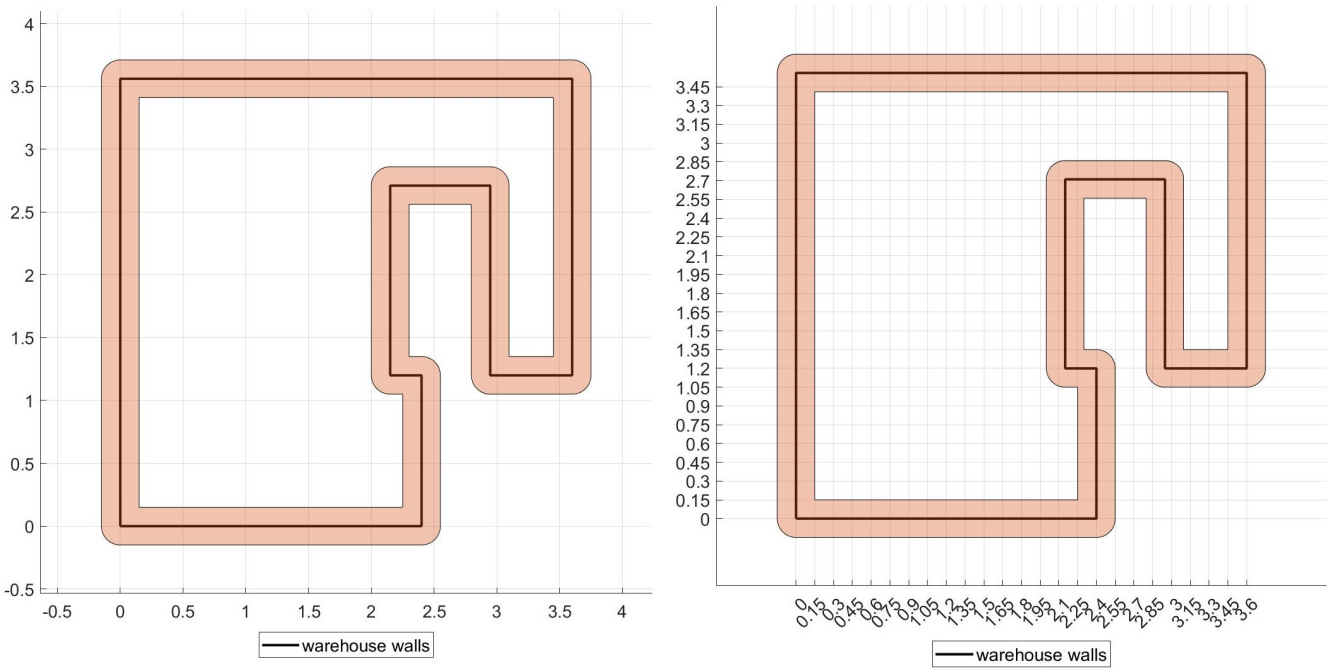
For one cycle the robot needs to perform, there consists of three paths, transit, delivery and return. The robot is designed to received the user command as an array of 2-digit numbers. For each element in the array, the former digit indicates the label dispenser and the latter indicates the receptacle label. The robot will then process each cycle, firstly by importing the three paths from its database, then process through each path in sequence shown in the FSM of Figure 18. It will automatically move to the next cycle once it finishes.

3.1.2 Algorithm

To navigate autonomously, the robot must be able to generate a geometric path between any two locations within the warehouse. A path-planning algorithm was developed, using MATLAB. The specifications chosen for the path-planning algorithm are such that:

- For efficiency, the path taken by the robot should be the path of smallest distance. Intuitively, by always taking the "fastest" routes, the robot would more likely be able to perform more cycles within a 5-minute span. Research suggests that such path is generated by A* (A-star) algorithm, an extension of Djikstra's graph traversal algorithm [6]. An initial, more primitive algorithm was first implemented based on smallest distance to target location (Greedy).
- To preserve the structural integrity of the robot, it is preferable that the robot does use wall collision to aid in navigation. Additionally, with the container's ramp perching outside the boundaries of the robot (base), it is recommended to keep a minimum distance of 15cm from any wall.
- The path must not cross the specified no-go zones. Since the layout of the warehouse is known beforehand, the no-go zones will be considered as walls.

The layout of the warehouse has been provided by the client from which a model was constructed in MATLAB. To abstract the robot model as a single point in space, a buffer distance must be added alongside the perimeter of the warehouse to account for the robot radius. Such task effectively defines the configuration space (C-space) of the warehouse, the "navigable" space. Extra precaution was given to reduce the chance that the robot crosses a no-go zone, by adding an additional 25cm buffer around the no-go zone, alongside the aforementioned 15cm (Figure 6a). The C-space was then divided into a matrix of equidistant points by cell decomposition. The cell dimension was chosen as the radius of the robot, approximately 15cm (Figure 6b). The horizontal and vertical axes denote the x and y coordinates respectively. Increasing the cell resolution increases the computational complexity, which is $O(b^d)$ for A*, where b denotes the branching factor, the average number of successors per state, and d the number of nodes.



(a) Warehouse configuration space

(b) Warehouse layout, after cell decomposition

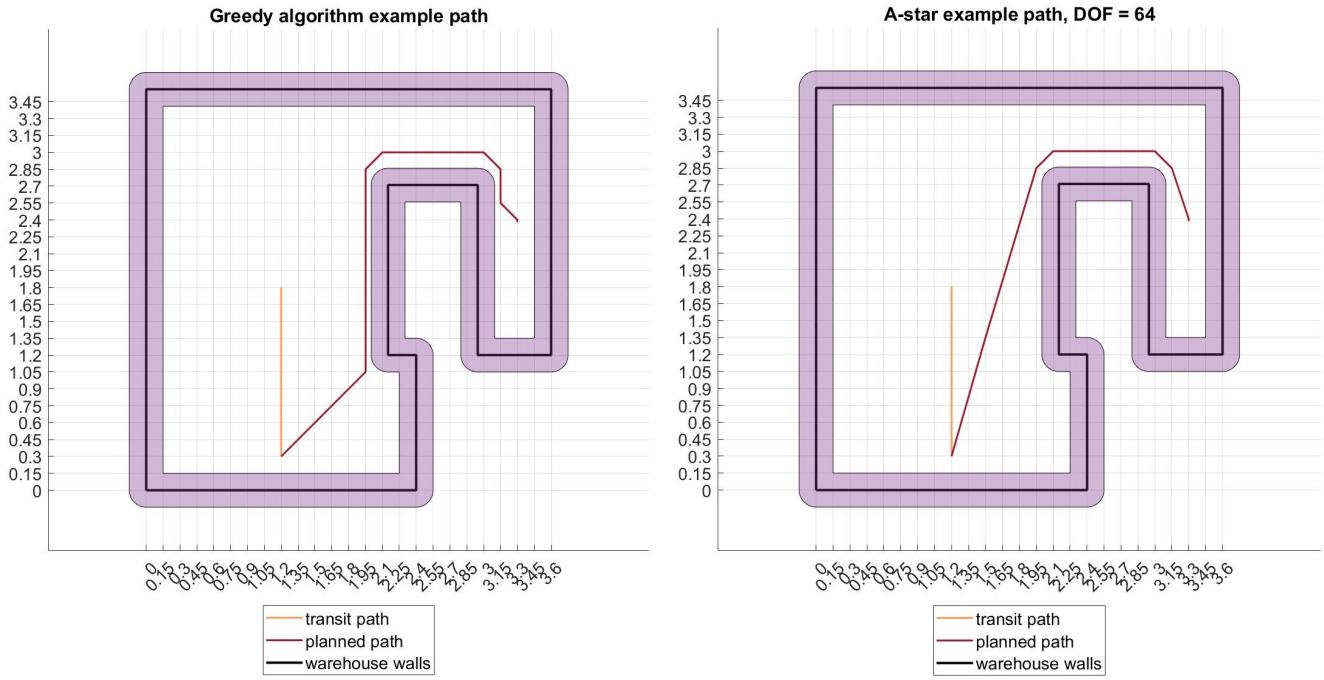
Figure 6: Stages in development of path-planning algorithm

At each starting point, the algorithm first attempts to navigate to the closest cell. It also checks whether the target location is valid, that is, inside the C-space. If valid, it also searches the cell

closest to the target location. Else, the target location will be redirected to the default starting location of the robot. Between the first and final cell, a graph-traversal algorithm is performed. The following section discusses two of such algorithms, namely Greedy and A* [7].

The Greedy algorithm searches for the shortest path based solely on least distance to the target location. At each cell, the program considers all valid surrounding cells (with a maximum of 8 closest cells), evaluating the cell closest to the target location. It also has memory, to recall the cells that already has been navigated in case of a dead-end. The Greedy algorithm is sub-optimal in that, it only selects the optimal choice at a specific point, and not the globally optimal choice (i.e. shortest path). In addition, for a $m * n$ graph, its time and space complexity are $O(m * n)$, which would grow substantially if the cell resolution is increased.

A* algorithm was chosen to replace Greedy to produce a smaller-distance path. A special feature of A* is that the user can assign a cost to each cell based on a certain heuristic. Thus, if the heuristic "cell is also close to starting point" is chosen, A* will generate the shortest distance path between any two locations. A* algorithm also allowed for more memory, allowing the algorithm to evaluate not just the closest 8 surrounding cells, but also their neighbours, and so on, therefore increasing the allowed degree of freedom. A* also produces a path that leads to fewer rotations. An example path between the starting location, the low dispenser and low receptacle is provided in Figure 7a and Figure 7b to compare Greedy and A*. This particular example allows A* to evaluate the 64 closest cells.



(a) Example path, using Greedy

(b) Example path, using A-star

Figure 7: Comparison of two path-planning algorithms

The A* algorithm is trialled to generate myriad different paths between any two points in the C-space, and most importantly, between the starting location, dispensers and receptacles. Simulation based tests seem to validate the algorithm design. Thus, A* is a very suitable algorithm to generate the path to be followed by the robot through trajectory tracking with the motor controllers.

However, it was discovered that a connection between ROS and Matlab could not established for the

robot. Specifically, the robot could not subscribe to the data published by MATLAB. This problem is suspected to be due to the inability to bypass the University's firewall to establish a connection. As a result, the path-planning algorithm could not be used to generate any real-time path between the current robot pose and a target location. Instead, the program was used to generated all relevant transit, delivery and return paths between all locations of interest. A path, which consists of arrays of x and y coordinates, would be saved as values in a comma-separated value (.csv) file, and passed to the robot in an offline process. The relevant paths would then be selected through user input.

3.2 Motor Control

This section covers the "Motor Controller" block in system level diagram 4. The input are the feedback and reference angular velocity and the output is the motor voltage. Firstly, the motor must be modelled to determine the internal parameters. Then, the controllers are designed based on the transfer function. In this project, both open loop and closed loop controller are designed. The relevant controller is later chosen for implementation.

3.2.1 Motor Modelling

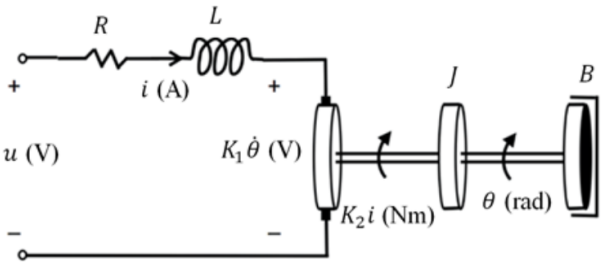


Figure 8: Simplified model of DC motor

The simplified motor model is shown in Figure 8, where the internal parameters are:

- R: motor's armature resistance
- L: motor's armature inductance
- K_1: electromotive force constant
- K_2: motor torque constant
- J: moment of inertia
- B: motor viscous friction constant

The transfer function from the input Voltage $V(s)$ to the angular velocity $\dot{\Theta}(s)$ is then can be derived [8]. By assuming that the contribution of inductance is negligible ($L = 0$), the transfer function becomes Equation (1) where $K_m = K_2/(BR + K_1K_2)$ and $T_m = RJ/(BR + K_1K_2)$.

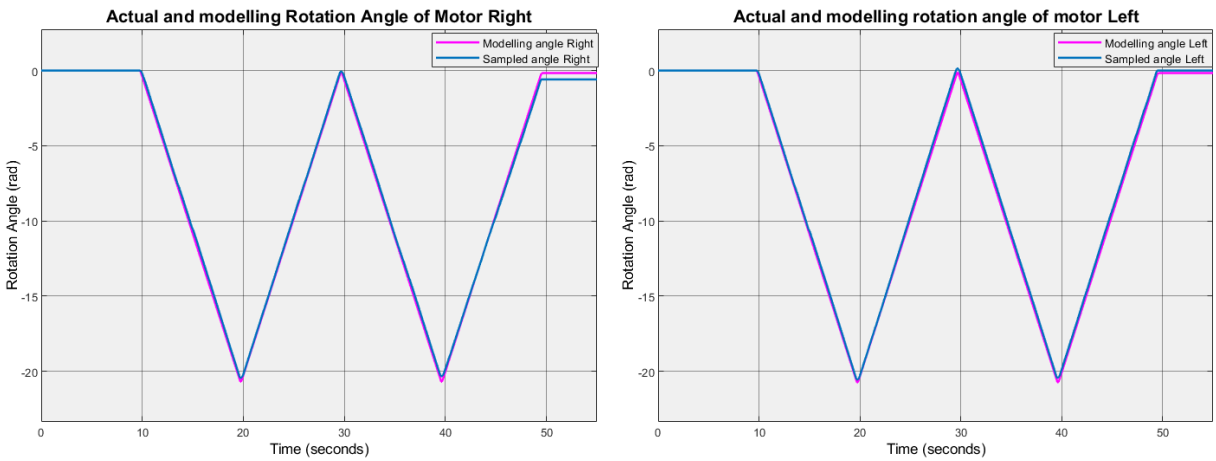
$$G(s) = \frac{\dot{\Theta}(s)}{U(s)} = \frac{K_2}{(R + Ls)(Js + B) + K_1K_2} = \frac{K_m}{T_m s + 1} \tag{1}$$

Parameters	Left motor	Right motor
K_m	1.745	0.06
T_m	1.74	0.05

Table 2: Estimated internal parameters

The internal parameters K_m and T_m are estimated, starting by measuring the rotation angles of two motors at $U(t) = 1.2/-1.2(V)$ and frequency 10 seconds. After that, these parameters of the two modelled motors are tuned in order to match the simulation and experimental results, as shown in Figure 9. The estimated internal parameters K_m and T_m of two motors are shown in Table 2.

3.2.2 Open loop controller



(a) Right motor

(b) Left motor

Figure 9: Actual and modelled rotation angle of two motors at |U(t)| = 1.2V

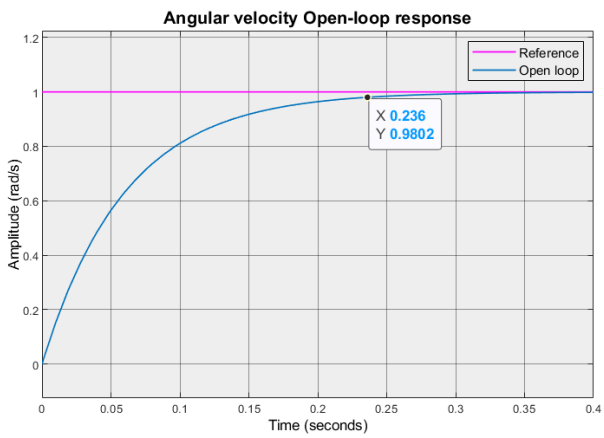


Figure 10: Open loop control response

Open loop control is the most simple technique by converting the desired angular velocity into pwm signal. The response can be seen in Figure 10, with the settling time $t_s \approx 0.24s$. However, this technique is not robust under the influence of the environment. For example, there is a difference in the performance of no-load and load mode. Moreover, the motor performance also changes under high load or friction.

3.2.3 Closed loop controller

Assume that the expectations of closed-loop response are: steady state error $e_{ss} = 0$, settling time $t_s^{in} \leq 0.1s$, overshoot $M_p^{in} \leq 5\%$. A continuous PI controller (PI_{con}) are then designed to meet the requirements. However, some issues can be found in this approach, such as discrete control (digital controller) and saturation ($U > 12V$). Therefore, based on the continuous PI controller, the "Anti-windup" scenario [9] is applied , which is then discretised by emulation technique [10]. The simulation results of these controllers are shown in Figure 11.

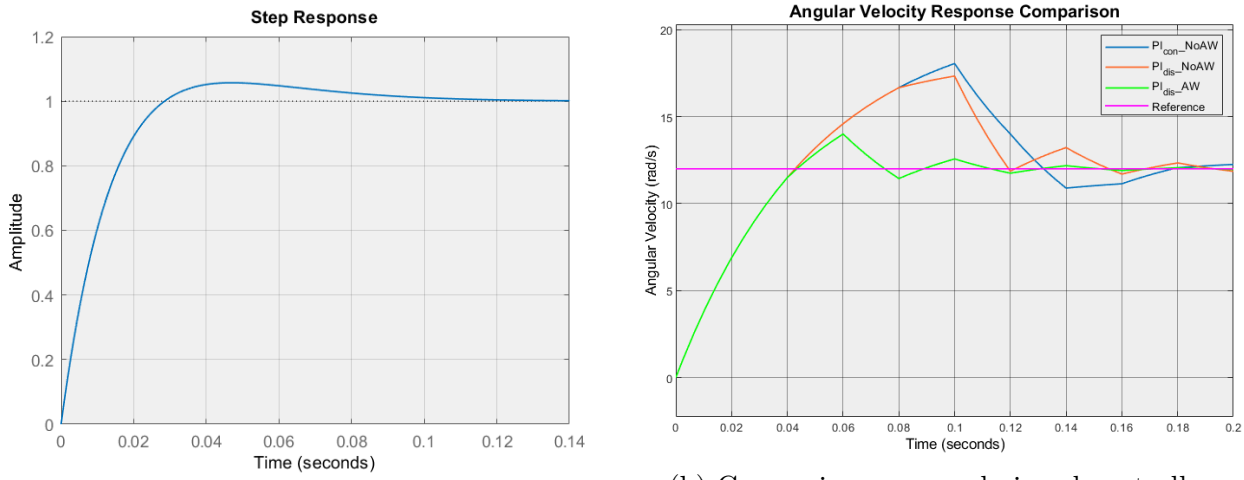
In the emulation step, the sampling time for inner loop is chosen $T_s^{in} = 0.02s$. It is decided by considering the resolution of encoder 800 ppr, when reading falling edge of 1 channel. When motor runs at lowest pwm = 5%, the encoder can detect 133 pps, leading to the lowest sampling time is 0.01s with high error.

As can be seen in Figure 11b, the discrete controller with anti-windup is the only controller that can make the system to meet the expectation with $t_s \approx 0.1s$. The traditional discrete controller could not deal with the saturation when $\dot{\theta}_{ref} > 3.5rad/s$. The PI digital controllers of two motor $U(z) = K_p + K_i z/(z-1)$ are shown in Table 3.

Controller	K_p^{right}	K_i^{right}	K_p^{left}	K_i^{left}
$C_{PI}(s)$	2.43	86.93	3.02	104.01
$C_{PI}(z)$	1.56	1.74	1.98	2.08

Table 3: Controller parameters

However, the expected response of discrete controller does not match that of continuous controller (Figure 11a vs Figure 11b). The reason is that at $\zeta = 0.95, \omega_n = 0.95$, the bandwidth is $\omega_{bw} = 38(rad/s)$. It leads to sampling time



(a) Step response of continuous controller

(b) Comparison among designed controllers

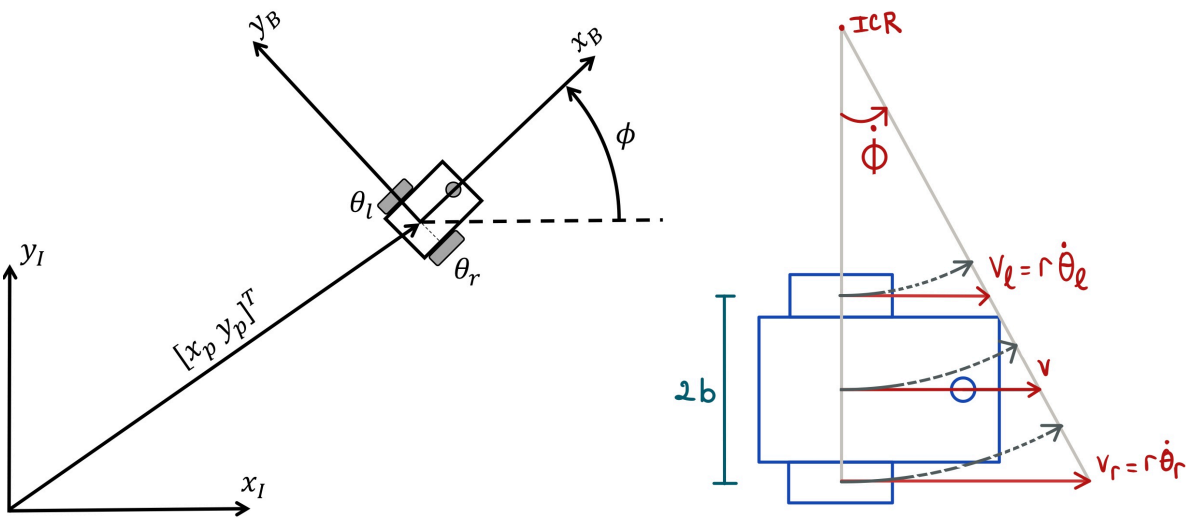
Figure 11: Simulation result in Angular Velocity Response

$T_s \leq 0.008s$ in order to ensure the discrete controller response similar to continuous controller response [10], which is roughly a third of designed sampling time $T_s^{in} = 0.02s$.

3.3 Pose Control

3.3.1 Kinematic Model

The drivetrain of a 2-wheel differential drive robot includes two coaxial motor-driven wheels with a passive wheel at the front and back for stablising. Motion control is mainly implemented on the two motor-driven wheels by sending PWM command to the motors. The straight path following can be performed on a flat surface when the two wheels can rotate at the same velocity, whereas purely rotation happens when they rotation at the same speed with opposite directions. The curvature of its path is determined based on the difference in the two wheels' speed (Figure 12b), so is ICR.



(a) (b)

Figure 12: Kinematics of the robot [11]

Refer to Figure 12a, the description of a 2-wheel differential drive robot is presented as follow:

- Wheel radius r
- Wheel base $2b$
- Pose of the robot
  - Inertial frame (x_I, y_I)
  - Body frame (x_B, y_B)
  - Position and heading $[x_p\ y_p\ \phi]^T$
- Input: Motor voltage (PWM)
- Measurements
  - Wheels angular velocity, $\dot{\theta}_l, \dot{\theta}_r$
  - Pose, $[x_p\ y_p\ \phi]^T$

Assuming that there is no slip on the drive wheels, the kinematic model of the 2-wheel differential drive robot in the body frame is shown as

$$\begin{bmatrix} \dot{x}_B \\ \dot{y}_B \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \frac{r}{2}(\dot{\theta}_l + \dot{\theta}_r) \\ 0 \\ \frac{r}{2b}(\dot{\theta}_r - \dot{\theta}_l) \end{bmatrix} = \begin{bmatrix} v \\ 0 \\ \omega \end{bmatrix} \implies \begin{matrix} \dot{\theta}_l = \frac{1}{r}(v - b\omega) \\ \\ \dot{\theta}_r = \frac{1}{r}(v + b\omega) \end{matrix} \tag{2}$$

where v and ω respectively are linear velocity along x_B axis and angular velocity about the center point of wheel base. Transforming from body frame to inertial frame gives

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v\cos\phi \\ v\sin\phi \\ \omega \end{bmatrix} \tag{3}$$

3.3.2 Trajectory Tracking Modelling

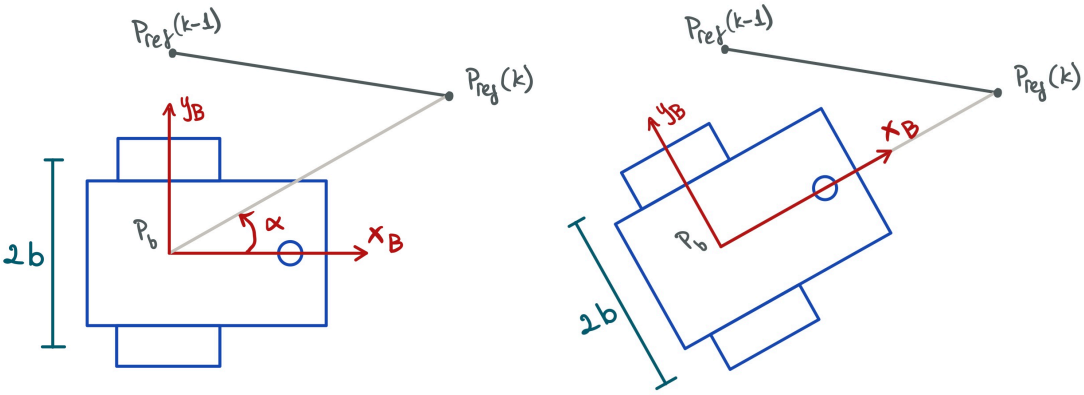
One challenge for controlling the robot is that this is a non-holonomic model. In other words, the rotation of two wheels at an instance of time will determine the next position of the robot and cannot be controlled separately. Referring to the kinematic model of the robot presented in the previous section, the trajectory tracking will be simplified to two different modes associated with two distinct movements of the robot: **(1) Pure rotation** and **(2) Follow a straight line with the given velocity profile**. The control architecture is built in accordance with following assumptions:

- There is no slip of the drive wheel
- For a reference straight path, the robot motion happens within a small period of time, hence the velocity is assumed to be constant within that time.
- The control architecture will neglect the inertia of the robot and the dynamics of the motors.

To have a better understanding about the approach to the design of pose controller, Figure 13 presents the current position of the robot $P_B = (x_C, y_C, \phi_C)$ with respect to the previous and current reference position, $P_{ref}(k-1)$ and $P_{ref}(k)$ in the inertial frame. With the previous reference position $P_{ref}(k-1)$, the robot has moved to the position P_B, with some errors in its pose. In order to move to the next reference position $P_{ref}(k)$

- Firstly, it needs to rotate by the angle α, calculated in the inertial frame (Figure 13a).
- Then, it will be switched to straight path following control, whose error is the Euclidean distance of the two positions (Figure 13b).

The control output which is the linear velocity along x_B, v will then be used to calculate the desired rotating speed on the wheel motors using Equation 2.



(a) Before (b) After

Figure 13: Pure rotation

Calculate α

To calculate the new heading reference at each time instance, referring to Figure 13a we firstly have to calculate the angle between $\overrightarrow{P_b P_{ref}}(k)$ and $\vec{x}_0$. Moreover, as we have the heading of the robot in the inertial frame at that time instance, output feedback from localisation, which is the angle between $\vec{x}_B$ and $\vec{x}_0$, α can simply be obtained from taking the difference of the two angles.

Model

From Equation 2, the kinematic model for **pure rotation** (mode 1) can be expressed as

$$\begin{aligned} \dot{\phi} = \omega \implies \mathscr{L}\{\dot{\phi}\} &= \Omega(s) \\ \therefore \frac{\Phi(s)}{\Omega(s)} &= \frac{1}{s} = G_1(s) \end{aligned} \tag{4}$$

and for **following a straight path with a given velocity profile** (mode 2)

$$\begin{aligned} \dot{x}_B = v \implies \mathscr{L}\{\dot{x}_B\} &= V(s) \\ \therefore \frac{X_B(s)}{V(s)} &= \frac{1}{s} = G_2(s) \end{aligned} \tag{5}$$

For each mode, the transfer function has one pole at the origin, hence they are type 1 systems. In unity-feedback control system, for a unit-step input, the steady-state error $e_{ss} = 0$[12].

3.3.3 Pose Estimation Using Odometry

Odometry or dead-reckoning measure the wheels' rotation as a function of time, and the location of the robot is extracted using geometric equations. In details, the optical encoders, which are mounted onto the motors, feed the pulse counters to the processor. The central processor then

estimates the pose of robot $\rho = [x_p, y_p, \phi_p]^T$ using Equation (6). Notate at each time t, the location of robot is $\rho_t = [x_{p,t}, y_{p,t}, \phi_{p,t}]^T$.

$$\rho_t = \begin{bmatrix} x_{p,t} \\ y_{p,t} \\ \phi_{p,t} \end{bmatrix} = \begin{bmatrix} x_{p,t-1} \\ y_{p,t-1} \\ \phi_{p,t-1} \end{bmatrix} + \begin{bmatrix} \Delta s_t \cos \phi_{t_1} + \Delta\phi_t/2 \\ \Delta s_t \sin \phi_{t_1} + \Delta\phi_t/2 \\ \Delta\phi_t \end{bmatrix} \tag{6}$$

where

$$\Delta s_t = \frac{r_r \Delta\theta_{r,t} + r_l \Delta\theta_{l,t}}{2}; \; \Delta\phi_t = \frac{r_r \Delta\theta_{r,t} - r_l \Delta\theta_{l,t}}{2b}$$

- $\Delta\theta_{r,t}$, $\Delta\theta_{l,t}$: Rotation angle of right and left wheel at time t (rad).
- r_r, r_l: Right and left wheel radius (m).
- b: Half the distance between two wheels (m).

Among the variables, r_r, r_l, b are constants related to robot chassis showed in Section, and $\Delta\theta_{r,t}$, $\Delta\theta_{l,t}$ are fed signals from the encoders.

However, the most significant limitation is that the localisation errors accumulate over time. The causes include systematic errors (unequal diameters, inaccurate wheel base) and non-systematic errors (wheel slippage, backlash, discrete wheel increments) [13].

3.3.4 Pose Estimation Using Camera

There are some notices in this section:

- There are 4 main frames: global frame $\vec{\mathcal{F}}_g$, marker frame (including 21 frames $\vec{\mathcal{F}_{mi}} \forall i \in [0; 20]$), camera frame $\vec{\mathcal{F}}_c$ and body frame $\vec{\mathcal{F}}_b$.
- The robot location $\vec{r}$ and the transformation matrices between frames are represented in homogeneous form.
- Notate T_b^g, T_m^g, T_c^m, T_b^c: the transformation matrix from $\vec{\mathcal{F}}_b$ to $\vec{\mathcal{F}}_g$, $\vec{\mathcal{F}}_m$ to $\vec{\mathcal{F}}_g$, $\vec{\mathcal{F}}_c$ to $\vec{\mathcal{F}}_m$ and $\vec{\mathcal{F}}_b$ to $\vec{\mathcal{F}}_c$ respectively.

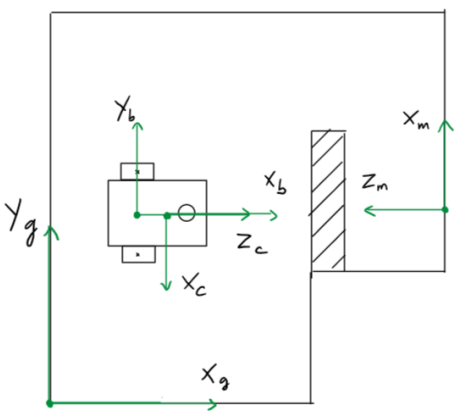


Figure 14: Coordinate system in relative with warehouse map

Firstly, the co-ordinate systems used for localization, as can be seen in Figure 14 include:

- $\vec{\mathcal{F}}_g$: global frame, whose origin is at the bottom right corner of the map.
- $\vec{\mathcal{F}}_b$: body frame, which attaches to the robot.
- $\vec{\mathcal{F}_{mi}}$: marker frame, which attaches to each marker. For simplification, $\vec{\mathcal{F}_{mi}}$ is notated as $\vec{\mathcal{F}}_m$ to represent marker frame in general.
- $\vec{\mathcal{F}}_c$: camera frame, which attaches to the camera.

Secondly, the robot location $\vec{r} = \vec{\mathcal{F}}_g^{\,T} r_g = \vec{\mathcal{F}}_b^{\,T} r_b$ in different frames is represented in homogeneous form,

such as $r_b = [0, 0, 0, 1]^T$. However, by using the transformation matrices to represent the relationship between different frames, it becomes Equation (7).

$$\vec{r} = \vec{\mathcal{F}}_g^{\,T} r_g = \vec{\mathcal{F}}_g^{\,T} T_b^g r_b = \vec{\mathcal{F}}_g^{\,T} T_m^g T_c^m T_b^c r_b \tag{7}$$

Thirdly, among the transformation matrices, while T_b^c and T_m^g are constants, T_c^m varies and is updated depending on the real location of the AMR. The constant T_b^c and T_m^g can be found in Appendix D. T_b^g therefore can be calculated using Equation (7). The general form of T_b^g is in Equation (8) and similarly for other transformation matrices.

$$T_b^g = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R_b^g & P_b^g \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{8}$$

Under the above assumptions, the location and heading of the AMRs in global frame can be estimated as:

- The location:

$$\vec{\mathcal{F}}_g^{\,T} r_g = \vec{\mathcal{F}}_g^{\,T} T_b^g r_b = \vec{\mathcal{F}}_g^{\,T} \begin{bmatrix} R_b^g & P_b^g \\ 0_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \vec{\mathcal{F}}_g^{\,T} \begin{bmatrix} P_b^g \\ 1 \end{bmatrix} \tag{9}$$

- The heading:

Consider the rotation matrix R_b^g from $\vec{\mathcal{F}}_g$ to $\vec{\mathcal{F}}_b$ using Euler angles with 3-2-1 sequence for rotational motion (rotate about z by ψ, followed by rotating about y by θ and then rotating about x by ϕ). The rotation matrix therefore can be presented as Equation (10) [14]. The detailed proof can be found in Appendix E.

$$R_b^g = R_z(\psi)R_y(\theta)R_x(\phi) = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{10}$$

Note that, under the assumptions that the robot maneuvers in a flat warehouse without slope, the rotation about x and y are $\phi = \theta = 0$. Moreover, with this Euler angle configuration, the heading of the robot coincides the rotation about z axis. The heading therefore can be estimated as:

$$\phi_p = \psi = \tan^{-1} \frac{r_{21}}{r_{11}} \tag{11}$$

3.3.5 Pose Controller

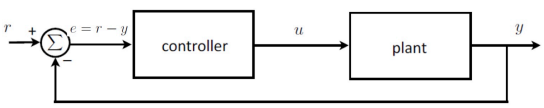


Figure 15: Closed loop system

Consider the transfer function $G_1(s)$ and $G_2(s)$ as shown in Equation 4 and 5, the transfer function for the closed loop system presented in Figure 15 using proportional controller $C(s) = K$ can be expressed as

$$G_{cl}(s) = \frac{Y(s)}{R(s)} = \frac{K}{s+K} \tag{12}$$

We want to find the bandwidth of the system at which the gain of $G_{cl}(s)$, equals to -3.010dB. In other words,

$$\begin{aligned} 20log_{10}(|G_{cl}(j\omega_{bw})|) &= 20log_{10}(\frac{1}{\sqrt{2}}) \\ |G_{cl}(jw_{bw})| = \left|\frac{K}{j\omega_{bw}+K}\right| &= \frac{1}{\sqrt{2}} \implies \left|j\frac{\omega_{bw}}{K}+1\right| = \sqrt{2} \\ \therefore \omega_{bw} &= K \end{aligned} \tag{13}$$

As the settling time of the inner loop is 0.25s, the discrete equivalent of the outer loop (pose control) needs to have the sampling time of at least 0.25s. According to [10], the discrete equivalent of this system can be used with confidence providing that the sampling rates is at 25 times the bandwidth or higher. means that

$$T_s \leq \frac{2\pi}{25\omega_{bw}} \implies \omega_{bw} = K \leq \frac{2\pi}{25T_s} = 1$$

The relation of settling time and bandwidth for the first order system can be expressed as

$$t_s = \frac{4}{\omega_{bw}} = \frac{4}{K} \tag{14}$$

Assuming that we want the robot to track reference point for every 5s, hence we choose the settling time for pose control $t_s = 5$ which equivalents to $\omega_{bw} = K = 0.8$.

The block diagram for controlling the two modes of movement is shown in Figure 16. The proportional controller $C(z) = K = 0.8$ is used for both heading and path tracking control, with the sampling time of 0.25s. The discrete step response is plotted in Figure 17, from which the settling time $y_{step} = 0.98y_{ss}$ is $t_s = 4.25s$ with no overshoot.

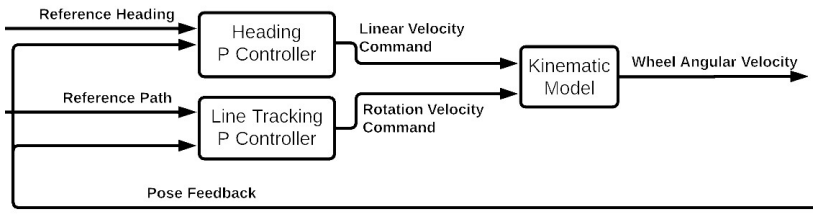


Figure 16: Pose Control block diagram

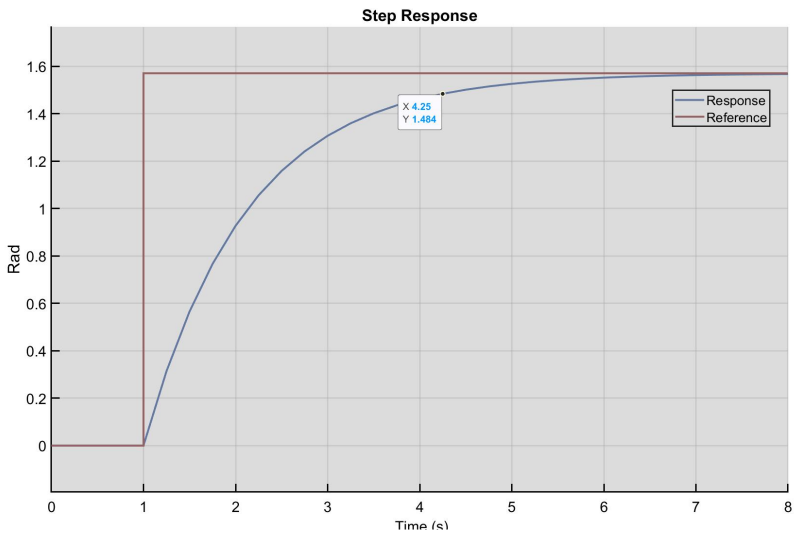


Figure 17: Step Response of the pose control

3.4 Finite State Machine

Having mentioned in the Project Plan, ***C++*** and ***Python*** are the two main programming languages will be used in this project. The FSM will also be constructed to present the high level predetermined operation sequence of the robot (Section 3.4) along with the system architecture developed from ROS framework (Appendix A).

Combining all the above design choices into one finite state machine we get:
Variables:

- AvailableCycle $\in \mathbb{N}$
- ReachPickUpPoint, ReachUnloadPoint, ReachStartPoint: pure
- CompletePickUp, CompleteUnload: pure
- PauseInTransit, PauseInDelivery, PauseInReturn: pure
- IsCameraTriggered: pure
- DonePureR: pure

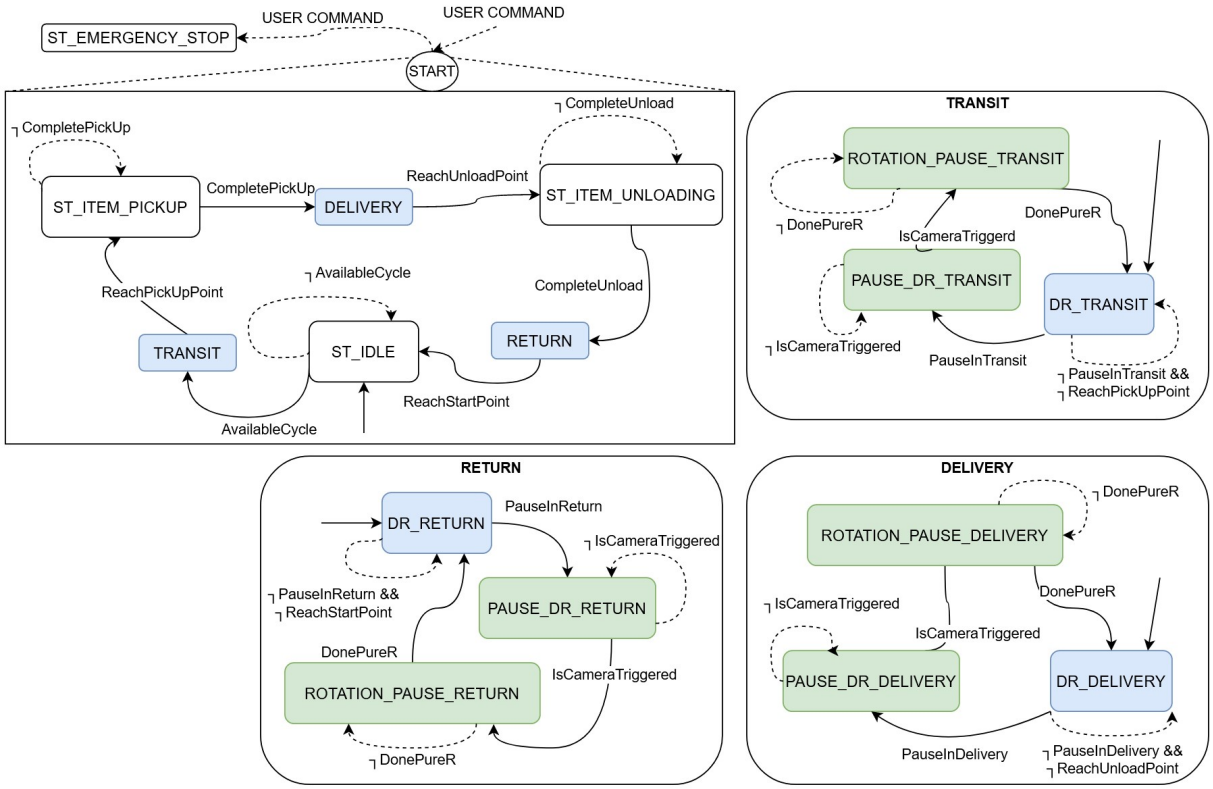


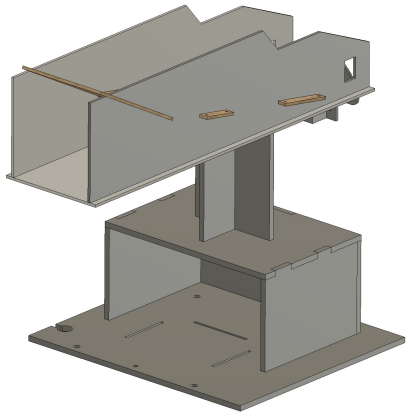
Figure 18: Finite State Machine

3.5 Mechanical Design

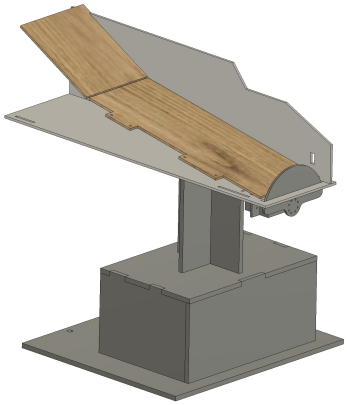
3.5.1 Loading Mechanism Design

For loading and unloading mechanism, a container is designed to be mounted on top of the robot. The design of the container features the use of 3-mm and 6-mm-thick MDF board with the laser cut technique for ease of manufacturing, serviceability and cost-efficient (Figure 19). Other features in the mechanical design are listed as follow

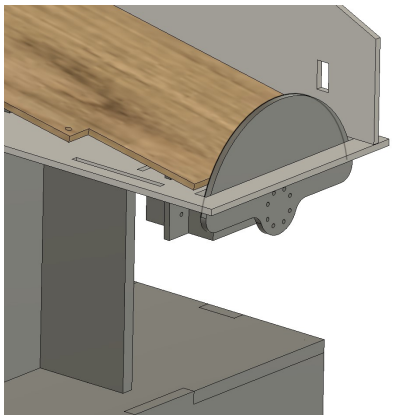
- The container can be easily mounted and removed from the robot, allowing for replacing different container designs for handling different types of items.
- A top slider with an inclination of 29° (Figure 19d) which is enough to generate momentum for the stress ball to roll and handle the elasticity of the bouncy balls.
- A dedicated slot for mounting the camera on top of the robot, providing extra protection and stability (Figure 19a).
- A dedicated slot for mounting the proximity sensor near the unload side for item detection (Figure 19c).
- Motor mounting mechanism with a rotating door on the unload side.



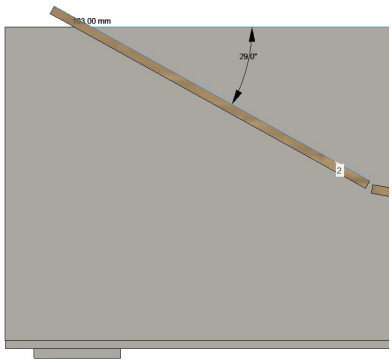
(a) Isometric view of the container



(b) Slider of the container



(c) Design of the unload side



(d) Top slider of the container

Figure 19: Container 3D model

3.5.2 Camera assembly

According to the datasheet, the camera diagonal fov is $dFoV = 78^o$, image frame ratio 16:9 (HD resolution). Therefore, the camera vertical and horizontal fov are $vFoV = 38.2^o$ and $hFoV = 68^o$.

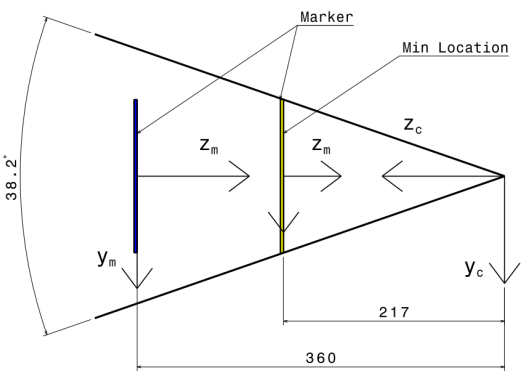


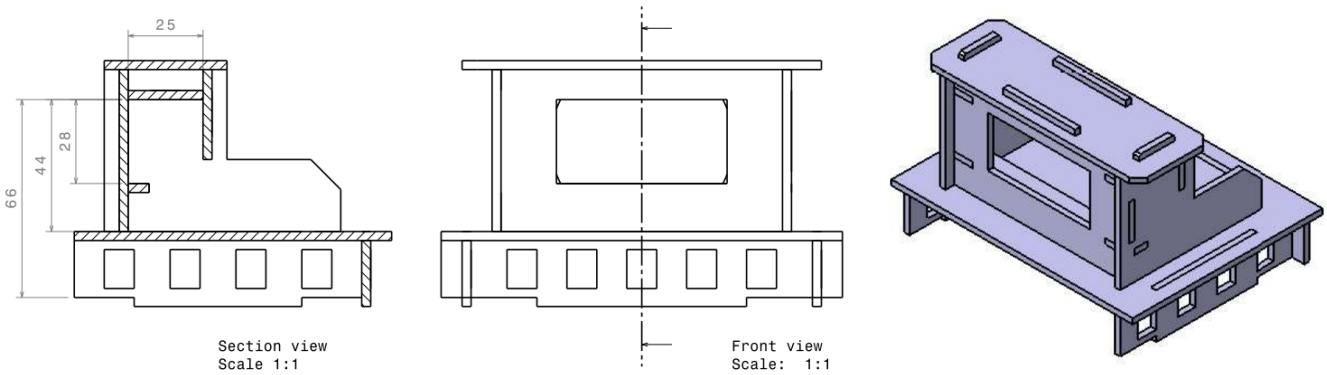
Figure 20: Marker position in camera fov

The objective is to determine the optimal height of the camera to minimise the distance robot - marker that the camera can still detect the marker. It ensures that the robot can approach the dispensers/ receptacles as close as possible. By the basic geometric analysis, the same height of camera frame origin and marker frame origin (height 0.185m) returns the optimal result. With respect to the marker size 0.15m, the minimum distance from camera origin to marker is 0.217m, as shown in Figure 20.

Note that the above analysis assumes that the z axis of $\vec{\mathcal{F}}_c$ is parallel with z axis of $\vec{\mathcal{F}}_m$. The final design is position that the distance camera - marker is 0.36m.

Moreover, according to the calculation in Section 3.3.4, the image plane is assumed to be perpendicular to the floor. It is essential to design a camera cover, that ensures this assumption relevant. Because this

project does not focus on mechanical design aspect, the design is simplified and only the most important dimensions are shown in the drawing 21.



(a) Camera cover drawing

(b) Isometric View

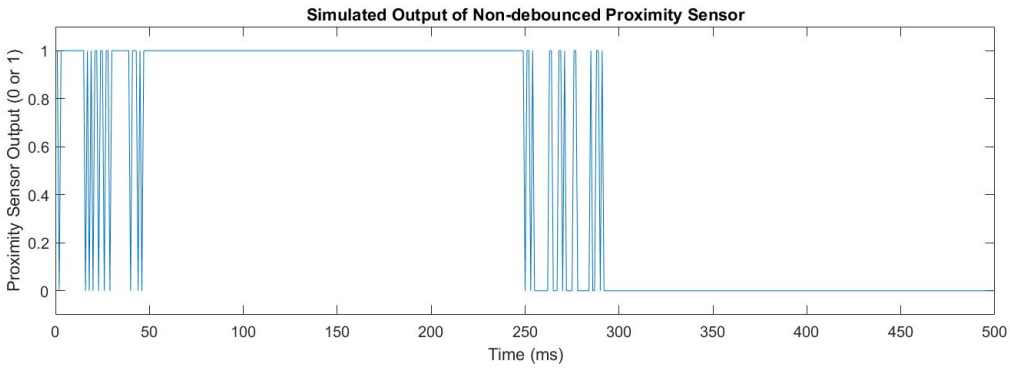
Figure 21: Camera cover

3.6 Signal Processing

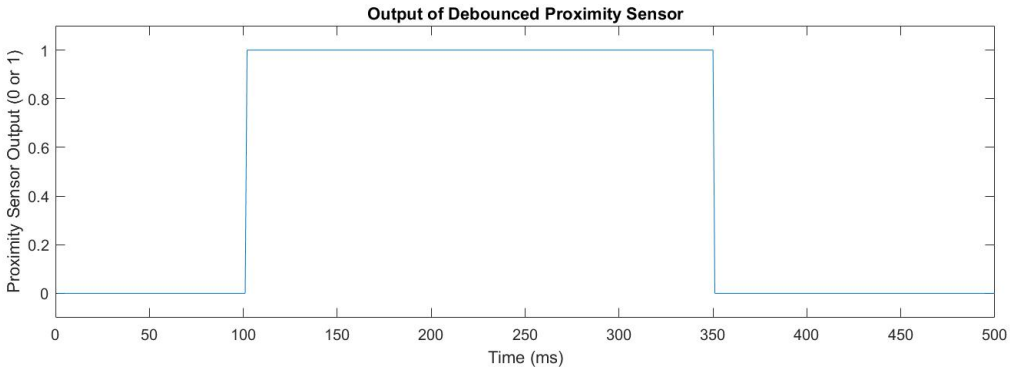
3.6.1 Debouncing the Proximity Sensor

The sensor supports a maximum polling rate of 400 Hz. However, output values for such high frequency are typically unstable and inaccurate. This is resolved by setting the period between each measurement to be longer than the maximum period required to sample an output specified by the datasheet [3] of 3.77ms. Thus, a 4ms period is selected corresponding to a polling rate of 250 Hz. To prevent false detection, the proximity sensor is digitally "debounced", requiring an object to be present for at least 100ms, or 25 samples to be registered as a detection. This time value was selected on the basis that it may reduce sensor noise, all the while obtaining an output quickly and correctly. The simulated outputs of a non-debounced sensor and a debounced proximity sensor is presented in Figure 22a and Figure 22b respectively, where an object is placed in the sensor detection range at t = 0ms, and subsequently removed at t = 250ms.

After debouncing the proximity sensor, it could then be mounted onto the container. During testing, it was discovered that while the sensor would reliably detect the stress ball, it would occasionally not detect the bouncy ball. This is the case for when the bouncy ball rolls onto the edge of the unload side of the container, touching the proximity sensor. As noted previously, its ranging capabilities are 2cm to 10cm. As a result the sensor is not able to detect the object. To resolve this issue, the wall portion of the container that holds the proximity sensor could further be redesigned to add a buffer, such that no object could fall into the 2cm range.



(a) Non-debounced sensor output



(b) Debounced sensor output

Figure 22: Simulated output logic of Proximity Sensor

4 Testing & Implementation

From the design procedure, each functional test has been conducted for each type of subsystems. Due to COVID-19 pandemic, there has been some limitations in obtaining the testing data on the physical robot.

4.1 Functional Test

This project was implemented starting with functional tests and final demonstration in the small-scaled warehouse.

4.1.1 Open-loop angular velocity control

The motor is control using Pololu library, which can be conducted via 7-bit/ 3200 cpm/ PWM signal. This project applied PWM signal. The main objective is to identify the rotational directions of two motors.

1. Identify rotational directions: send a positive (+) PWM signals to motor. They motor directions coincide with CW/CCW direction. They are then defined in CW_PWM_R and CW_PWM_L (+1/-1: PWM positive direction of robot is CW/CCW direction of motor)
2. Identify forward/backward direction: the forward direction of robot is equivalent to CW/CCW rotation of each motor. They are then defined in FW_CW_L and FW_CW_R (+1/-1: FW direction of robot is CW/CCW direction of motors).

3. As a result, the equivalent velocity direction is identified by $FW_CW_L * CW_PWM_L$ and $FW_CW_R * CW_PWM_R$.

4.1.2 Encoder monitors

The Encoders are monitors using single channel single edge detection. The resolution in this case is 800 cpm. In more details, when the trigger event occurs on the falling edge of channel A, the counter increases/ decreases, depending on the state of channel B. Assume that the CW rotation is equivalent with when falling event on channel A and channel B state is "OFF". The "encoder direction" must be adjusted to ensure that the sign of encoder counter is the same as the sign of forward direction. It is conducted by sending positive reference velocity. The sign of the encoder counters are then flipped by variable FW_ENC_R and FW_ENC_L (+1/-1: FW direction of robot is equivalent to +/- direction of encoders).

4.1.3 Pose estimator using Odometry

As can be seen in system level diagram 4 and odometry estimation equation (6), the odometry takes the rotation angles as the input signal, which can be calculated from the encoder counters as:

$$\Delta\theta = 2\pi * \frac{counter}{resolution}$$

The test is conducted by in 2 modes: pure rotation and linear translation. The constant angular velocities $v_{ref} = 2(rad/s)$ are sent in a period of $t = 10s$. Assume the wheel radius $r_r = r_l = 0.05m$, in linear translation, the localisation are calculated by-hand to be: $v_{ref} * t * r_l = 1(m)$. Similarly, in pure rotation mode, the heading are calculated as $2 * v_ref * t * r_l/(2b) = 8.92(rad)$

4.1.4 Closed loop angular velocity control

The feedback signals for inner loop controller are the angular velocities, which are measured by encoder counters in the sampling time t_s:

$$\Delta\dot{\theta} = 2\pi * \frac{counter}{resolution * t_s}$$

The test was conducted by replacing the "open-loop controller" by the new "closed-loop controller" as designed in Section. However, due to inaccurate encoder counters when velocity is higher than $PWM = 20$, the controller did not work. Therefore, this project uses the open loop controller.

4.1.5 Pose estimator using Camera

Identify coordinate systems: As discussed in Section 3.3.4, the algorithm for pose estimation using camera needs to identify all the coordinates. However, only global frame $\vec{\mathcal{F}}_g$ and body frame $\vec{\mathcal{F}}_b$ are known. According to the marker detection algorithm, the camera frame $\vec{\mathcal{F}}_c$ has z axis points along optical axis, x axis points to the right and y axis points to the down when looking along optical axis. By reading the rotation matrix R_m^c when projecting single marker, the marker frame $\vec{\mathcal{F}}_m$ is identified.

4.1.6 User command acknowledgement

This tested the robot on its ability to proceed the user command, input as publishing ROS message into a dedicated topic for switching between different states. In ROS, this functionality is introduced in "*Motion planning*" node (Appendix A) with "*/state_indicator*" topic. Having successfully tested this function will enable the robot to have not only an emergency stop feature without having to disengage the electrical connection, but also a feature to allow it to receive and proceed multiple pick-and-drop requests.

As a result, the robot has passed this test. It could perform a stop or start operation with the correct database of its path when receiving user commands without having any trouble.

4.2 Implementation

One of the most important parts of this design is localisation, as where the pose errors come from that can makes the system go unstable within a second. Therefore, we put more effort into testing the performance of the camera and encoders. We first implemented the robot without using pose feedback from the camera. The robot is placed at the the start zone, facing towards $D1$ and waiting for input command. The behaviour of the robot is then observed as it started to go to the dispensers. Then we implement the pose estimation by camera function on the robot and re-run the cycle, we want to compare and see if there is any improvement in the performance of the robot when using both camera and encoder for localisation. Lastly, we rely mostly on pose estimation by encoders and implement a dedicated state at which the robot standstill and activate the camera to scan code, this is to prevent inaccuracy in pose estimation by camera.

Due to unforeseen circumstances related to COVID-19 pandemic, the demonstration data cannot be recorded and analysed. Therefore, for the result of the end-to-end demonstration, we will state based on our observation. From the demonstration, we found out that the robot did not reach exactly to the indicated position to pickup the items. The error can be observed to be within the a circular range of 0.05m. However, with the mechanical design of the container, the robot was still able to handle both items released from both dispensers. As the robot moved from a dispenser to a receptacle, it could not finish the task as the system started to behave abnormally during half way. Further investigation needs to be conducted once we can get access to the demonstration data. Further improvement and some limitations in this project will be discussed in the next section of the report.

5 Validation

5.1 Reachability Analysis

Here we will find the set of reachable states R of our finite-state machine. This is a reachability analysis using symbolic search discussed in page 411412 [15].

First let v be a variable denoting the state of our FSM at the start of each reaction. Then the reachable state R is given by

$$
\begin{aligned}
& v = \text{START} \\
\vee & v = \text{ST_IDLE} \wedge \text{AvailableCycle} = 0 \\
\vee & v = \text{DR_TRANSIT} \wedge \text{PauseInTransit} = \text{false} \wedge \text{ReachPickUpPoint} = \text{false} \\
\vee & v = \text{PAUSE_DR_TRANSIT} \wedge \text{IsCameraTriggered} = \text{false} \\
\vee & v = \text{ROTATION_PAUSE_TRANSIT} \wedge \text{DonePureR} = \text{false} \\
\vee & v = \text{ST_ITEM_PICKUP} \wedge \text{CompletePickUp} = \text{false} \\
\vee & v = \text{DR_DELIVERY} \wedge \text{PauseInDelivery} = \text{false} \wedge \text{ReachUnloadPoint} = \text{false} \\
\vee & v = \text{PAUSE_DR_DELIVERY} \wedge \text{IsCameraTriggered} = \text{false} \\
\vee & v = \text{ROTATION_PAUSE_DELIVERY} \wedge \text{DonePureR} = \text{false} \\
\vee & v = \text{ST_ITEM_UNLOADING} \wedge \text{CompleteUnload} = \text{false} \\
\vee & v = \text{DR_RETURN} \wedge \text{PauseInReturn} = \text{false} \wedge \text{ReachStartPoint} = \text{false} \\
\vee & v = \text{PAUSE_DR_RETURN} \wedge \text{IsCameraTriggered} = \text{false} \\
\vee & v = \text{ROTATION_PAUSE_RETURN} \wedge \text{DonePureR} = \text{false}
\end{aligned}
$$

Given the set of reachable states, we found that there is no unreachable states, the robot can successfully switched back to "IDLE" state once it finished the cycle.

6 Discussion

6.1 Observation

6.1.1 Motor control

There are some essential observations in the motor's behaviors during inspections:

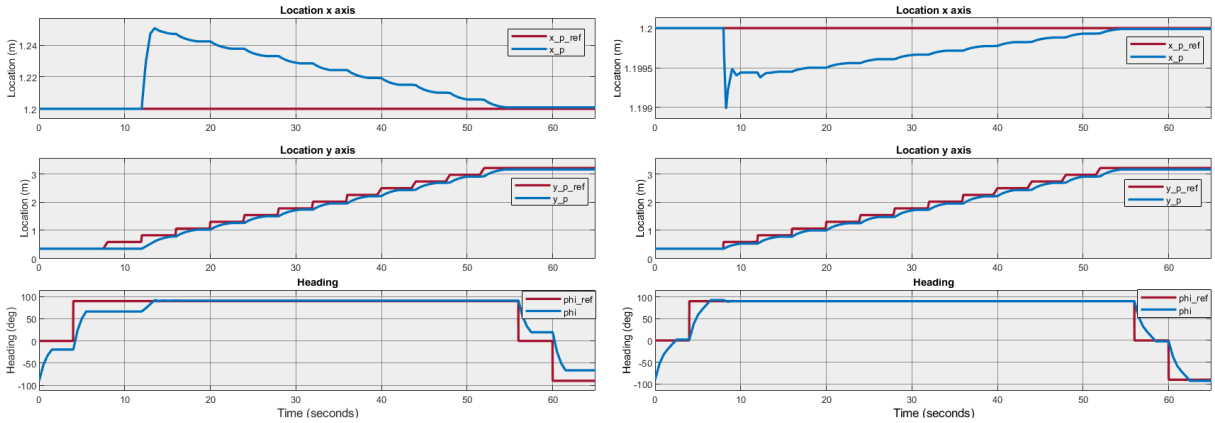
1. The encoder counters become inaccurate and imprecise when the the PWM signal is greater than 20% ($abs(PWM) > 20$) due to high frequency of encoder counter. For example, by reading single channel single edge, the frequency at PWM=50 is $f = resolution * \theta_{max} * PWM/(100 * 60) = 1,333(Hz)$.
2. The motors do not work when the PWM signal is less than 4% ($abs(PWM) < 4$) due to low generated torque.

The above analysis explain the reasons why the closed-loop control designed in Section 3.2 did not work.

6.1.2 Pose control

During experiment, the robot could not reach the desired end-points (S, D1, D2, R1, R2) in Figure 5b. The robot kept stopping when the distances are roughly 0.05-0.1(m). The reason is that the closed-loop controller (P controller) in Section 3.3 leads to low control effort when it reaches close to the reference. It matches the observation in Section 6.1.1, where the motors does not work at low reference velocity. This issue can also be simulated, as in Figure 23a.

It is then overcome by adjusting the motor controller when it meets lower bound. In stead of using the desired voltage of the CL pose controller, motor takes the lower bound as the reference until the steady state error is less than 2%. The "adjusted CL controller" response while travelling D1-R1 path can be seen in Figure 23b.



(a) Designed CL controller

(b) Fine-tune CL controller

Figure 23: System response comparison

6.1.3 Pose estimation using camera

In static test, the returned pose from camera is relevant. However, if the camera estimates the pose when the robot is running, the results are not precise. Therefore, the pose estimator using camera should be activated only if the robot stops. This solution is shown in FSM Section 3.4, as the states "PAUSE_DR_TRANSIT", "PAUSE_DR_DELIVERY" and "PAUSE_DR_RETURN".

6.2 Limitations

There is a wide range of limitations due to the mild assumptions in this project.

Firstly, the **inaccurate estimation** in constant wheel radius r_r, r_l and the wheel base b lead to inaccurate odometry estimation, as discussed in Section 3.3.3. This issue is especially serious in pure rotation mode because the wheel base is almost unable to be correctly measured.

Secondly, the **open-loop velocity controller** did not ensure the robot runs at desired velocity, leading to the difference in pose control between simulation and experimental results.

Thirdly, the **non-robust camera estimator** causes the wrong localisation. It is likely to occur when the distance between the camera and the markers dominates. One of the consequence can be seen in the final demonstration, when the camera failed to determine robot's real location. It caused the misbehaviors of the robot on the delivery path D1->R1.

Finally, the **inaccurate estimation of the transformation matrix** T_b^c. The reason is that it is impossible to determine the relative position of camera frame $\vec{\mathcal{F}}_c$ to body frame $\vec{\mathcal{F}}_b$. The design in Section 3.5.2 only mitigates the negative effects of this issue by ensuring the z axis of $\vec{\mathcal{F}}_c$ coincides with x axis of $\vec{\mathcal{F}}_b$.

6.3 Further Improvements

To overcome the above limitations, some improvement can be conducted.

- Measure and correct the systematic odometry errors [16] [13].
- Design the closed-loop controller for voltage saturation V=2.4V (equivalent to PWM = 20 and the lower boundary of motor). The simulation result can be found in Appendix B.

- Apply Kalman filter to better fuse the odometry signal and camera signal. However, it is challenging to determine the covariance matrix of the camera.
- Calibrate mechanical assembly of camera.

7 Conclusion

In conclusion, the potential for further development on this project is huge. During a few months, we have designed software and hardware for the AMR based on a given platform. We have drawn knowledge from different disciplines and worked on multiple engineering applications deployed on the robot to perform the task. Through vigorous testing, we have improved our design and made it more robust. However, there were some limitations that prevented us from delivering a fully working robot as planned.

Appendices

A ROS Architecture

Figure 24 presents the ROS system architecture implemented in the main computer of the robot.

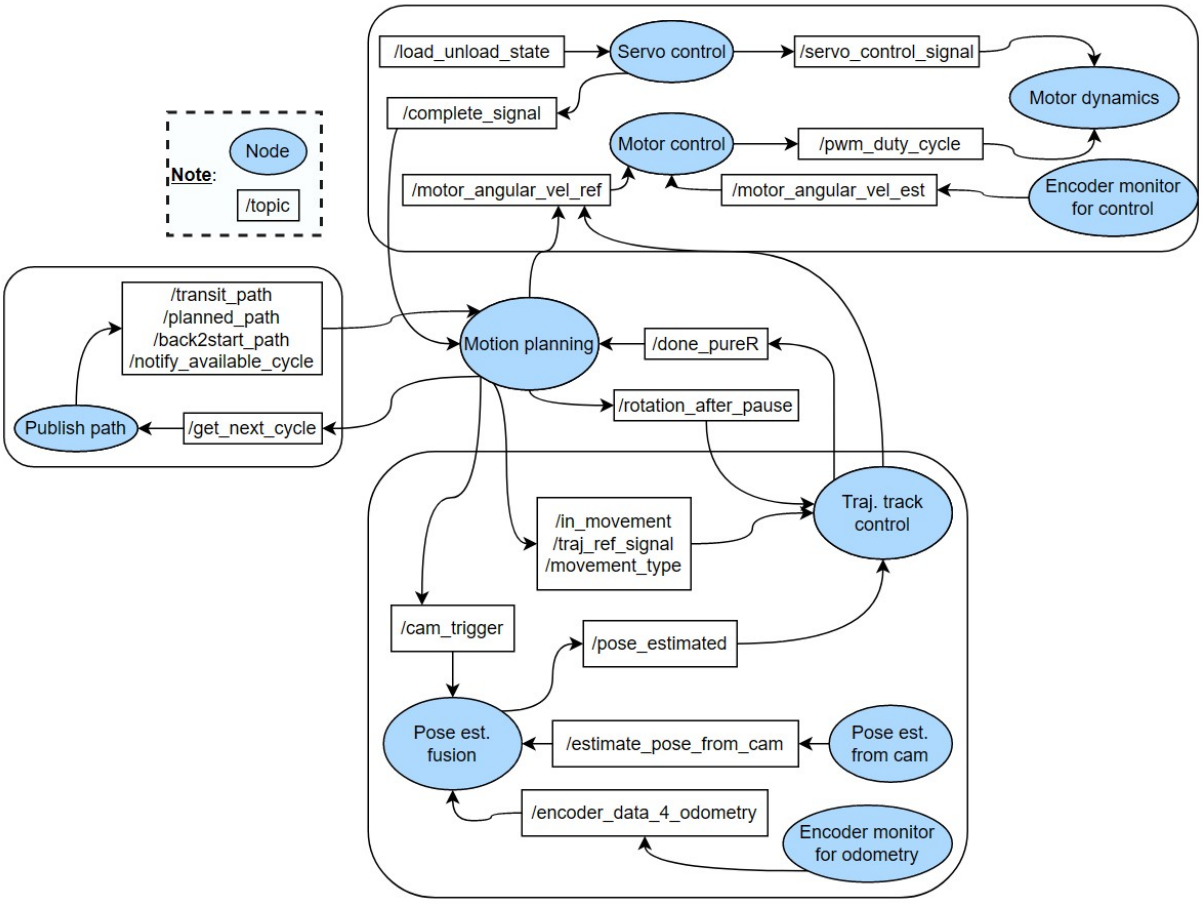


Figure 24: ROS architecture

- Motion planning: The node manages the state of the robot, including emergency stop case, importing the path database and transfering to the **Traj. track control** node for pose control purpose. It also publish a flag signal for activate the servo motor on the container for unloading the items.
- Traj. track control: The node manages the pose controller, receiving the reference pose from **Motion planning** node and calculate desired linear and angular velocity before sending to the **Motor control** node.
- Pose est. fusion: The node fuses the localization signals of **Pose est. from cam** and **Encoder monitor for odometry**.
- Pose est. from cam: The node monitors camera and estimates the pose of robot.
- Encoder monitor for odometry: The node takes the encoder counters as the input and estimates the pose.

- Publish path: The node manages to convert the type of data and send the path data to **Motion planning** node. It also takes a role as an outer loop keeping track of the cycle the robot needs to perform. It is also the node that receives the user command for starting a series of cycle.
- Motor control: The node manages the motor control, by taking the desired and the real angular velocity as the input and returning the PWM signals.
- Servo control: The node manages to open/ close the trap door of the loading mechanism.
- Motor dynamics: The node can read the PWM signal sent from motor controller before generating PWM duty cycle to the motor.
- Encoder monitor for control: The node takes the encoder counters as the input and estimates the angular velocity.

B Fine Tune Velocity Control

With respect to the first limitation in Section 6.1, the closed loop controller in Section 3.2.3 is "fined tuned" for the "adjusted saturated voltage" 2.4V (equivalent to PWM 20%). With this adjustment, the settling time for inner-loop control considerably increases, but it is still faster than open-loop control $t_s = 0.24s$. The comparison between open-loop control and closed-loop control can be seen in Figure 25.

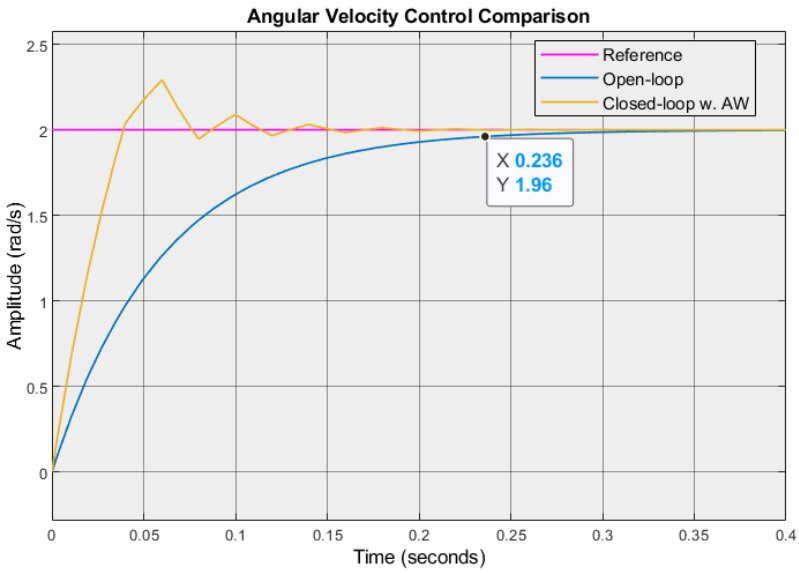


Figure 25: Angular velocity comparison

C Camera Calibration

In addition to the odometry estimate, a camera vision system is implemented to generate a separate pose estimate by detecting ArUco markers placed at known locations around the warehouse. Both odometry and camera estimates will subsequently be "fused" together to deduce the most accurate current robot pose. Thus, the camera module must output an exact location of the robot, or at least, one that is more correct than the odometry data.

Initial testing revealed that images captured during robot movement are blurry and therefore unusable for self-localisation. Hence, it was decided that images would only be captured when the robot is static. As a result, the faster rate of capture is not required, thus, 1080p at 30 frames per second was selected. In addition, using a higher resolution typically yields better accuracy for pose estimation, provided the processing time needs not be extremely fast. Intuitively, it can be understood that since 1080p has over twice the amount of pixels compared to 720p, there would consequently be better pixel quantisation performance and therefore better accuracy. Other considerations include the pixel format, with available choices being "YUV" and "MPEG". Tests are once again inconclusive regarding the advantages of either format. MPEG was chosen for its widespread use.

Camera Calibration

To obtain a precise reading, the camera must first be calibrated to extract its intrinsic parameters. These parameters include scaling factors, which relate the real-world dimensions to the camera image plane dimensions in pixels, and radial distortion parameters, which are linked to the imperfections of the camera lens, causing real-world straight lines to not be mapped to straight lines in the image. Before capturing pictures, the auto-focus feature of the camera must be turned off, and its focus level set to zero corresponding to a focusing at "infinity". Calibration was performed based on an OpenCV tutorial [17] using Python, by repeatedly capturing images of a chessboard, viewed from multiple perspectives. A final sample image bank of 60 was recorded, with an example shown in Figure 26.

The intrinsic parameters could then be extracted by running a Python script, which calculates and averages these parameters for the entire set of pictures, yielding the following camera matrix.

$$CameraMatrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 639.61935325 & 0 & 313.65162393 \\ 0 & 640.25521442 & 243.24068703 \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

Additionally, the distortion parameters are:

$$\begin{aligned} DistortionCoefficients &= \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \\ &= \begin{bmatrix} 0.05328607 & -0.16649145 & -0.00126354 & -0.00216118 & 0.06680599 \end{bmatrix} \end{aligned} \tag{16}$$

To estimate the exactness of the extracted parameters, the re-projection error can be calculated. A Python script is written such that it takes in the intrinsic parameters as found in 15 and 16 to re-project the chessboard corners onto each individual calibration image, then calculates the absolute norm of the difference between the projected corner locations and the actual corner locations of the

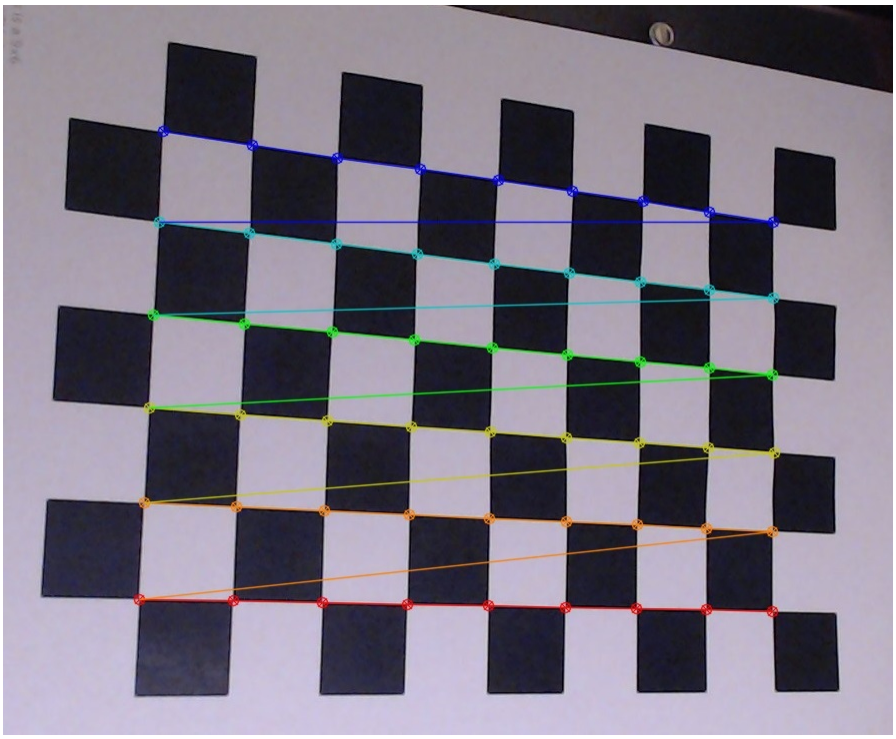


Figure 26: Example of output image from camera calibration step

chessboard. The re-projection error is then the arithmetical mean of the errors for all calibration images. The closer it is to zero, the more accurate the parameters are. The re-projection error is found to be $error = 0.0025634630326241$.

D Transformation Matrix for Markers

Assume that the height of the markers' center in comparison with the warehouse floor $h = 0.1845(m)$.

$$T^g_{m0} = \begin{bmatrix} 1 & 0 & 0 & 2.00 \\ 0 & 0 & 1 & 0.00 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m1} = \begin{bmatrix} 1 & 0 & 0 & 1.60 \\ 0 & 0 & 1 & 0.00 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m2} = \begin{bmatrix} 1 & 0 & 0 & 1.20 \\ 0 & 0 & 1 & 0.00 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$T^g_{m3} = \begin{bmatrix} 1 & 0 & 0 & 0.80 \\ 0 & 0 & 1 & 0.00 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m4} = \begin{bmatrix} 1 & 0 & 0 & 0.40 \\ 0 & 0 & 1 & 0.00 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m5} = \begin{bmatrix} 0 & 0 & 1 & 0.00 \\ -1 & 0 & 0 & 0.58 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$T^g_{m6} = \begin{bmatrix} 0 & 0 & 1 & 0.00 \\ -1 & 0 & 0 & 1.18 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m7} = \begin{bmatrix} 0 & 0 & 1 & 0.00 \\ -1 & 0 & 0 & 1.78 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m8} = \begin{bmatrix} 0 & 0 & 1 & 0.00 \\ -1 & 0 & 0 & 2.38 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$T^g_{m9} = \begin{bmatrix} 0 & 0 & 1 & 0.00 \\ -1 & 0 & 0 & 2.98 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m10} = \begin{bmatrix} 1 & 0 & 0 & 0.40 \\ 0 & 0 & 1 & 3.56 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m11} = \begin{bmatrix} 1 & 0 & 0 & 0.80 \\ 0 & 0 & 1 & 3.56 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$T^g_{m12} = \begin{bmatrix} 1 & 0 & 0 & 1.20 \\ 0 & 0 & 1 & 3.56 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m13} = \begin{bmatrix} 1 & 0 & 0 & 1.60 \\ 0 & 0 & 1 & 3.56 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m14} = \begin{bmatrix} 1 & 0 & 0 & 2.00 \\ 0 & 0 & 1 & 3.56 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$T^g_{m15} = \begin{bmatrix} 1 & 0 & 0 & 3.00 \\ 0 & 0 & 1 & 3.56 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m16} = \begin{bmatrix} 0 & 0 & -1 & 3.60 \\ 1 & 0 & 0 & 2.98 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m17} = \begin{bmatrix} 0 & 0 & -1 & 3.60 \\ 1 & 0 & 0 & 2.38 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$T^g_{m18} = \begin{bmatrix} 0 & 0 & -1 & 3.60 \\ 1 & 0 & 0 & 1.78 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m19} = \begin{bmatrix} 1 & 0 & 0 & 3.00 \\ 0 & 0 & 1 & 1.20 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}; T^g_{m20} = \begin{bmatrix} 0 & 0 & -1 & 2.40 \\ 1 & 0 & 0 & 0.58 \\ 0 & -1 & 0 & h \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

E Euler Angles for R_b^g

Notate $s_\theta \equiv \sin\theta$, $c_\theta \equiv \cos\theta$ and similarly for other angles. In this project, the rotational matrix uses Euler angles with 3-2-1 sequence for rotational motion. In other words, $\vec{\mathcal{F}}_b$ is created from $\vec{\mathcal{F}}_g$ by rotating about $\vec{z}$ by ψ, followed by rotating about $\vec{y}$ by θ and then rotating about $\vec{x}$ by ϕ.

$$
\begin{aligned}
R_b^g = R_z(\psi)R_y(\theta)R_x(\phi) &= \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix} \\
&= \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}
\end{aligned}
$$

Under the assumptions that the robot maneuvers in a flat warehouse without slope, the rotation about $\vec{x}$ and $\vec{y}$ are $\phi = \theta = 0$. The rotational matrix is then simplified as:

$$
R_b^g = \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

References

[1] Asclinic-wiki, "Asclinic wiki," 2021, accessed: 2021-05-31. [Online]. Available: https://people.eng.unimelb.edu.au/pbeuchat/asclinic/

[2] Pololu, *37D Metal Gearmotors*, 2021. [Online]. Available: https://www.pololu.com/file/0J1736/pololu-37d-metal-gearmotors-rev-1-2.pdf

[3] SHARP-Corporation, "Gp2y0d810z0f digital distance measuring sensor unit (100mm)," 2006, accessed: 2021-05-31. [Online]. Available: https://www.pololu.com/file/0J154/GP2Y0D810Z0F.pdf

[4] Logitech, "C922 pro hd stream webcam," 2021, accessed: 2021-05-31. [Online]. Available: https://www.logitech.com/en-au/products/webcams/c922-pro-stream-webcam.960-001090.html

[5] Polulu, "Power hd high-torque servo 1501mg," accessed: 2021-05-31. [Online]. Available: https://www.pololu.com/product/1057

[6] P. N. Stuart Russell, *Artificial intelligence : a modern approach*, 4th ed. Pearson, 2018.

[7] E. Ueland, "Astar-algorithm," accessed: 2021-05-31. [Online]. Available: https://github.com/EinarUeland/Astar-Algorithm

[8] W. P. Aung, "Analysis on modeling and simulink of dc motor and its driving system used for wheeled mobile robot," *World Academy of Science, Engineering and Technology*, vol. 32, pp. 299–306, 2007.

[9] M. E. S. Graham C. Goodwin, Stefan F. Graebe, *Control System Design*. Prentice Hall PTR, 2000.

[10] A. E.-N. Gene F. Franklin, J. David Powell, *Feedback Control of Dynamic Systems, Global Edition*, 8th ed. Pearson, 2019.

[11] P. Beuchat, "Asclinic lecture 5: Trajectory tracking," 2021, accessed: 2021-05-31.

[12] K. Ogata, *Modern control engineering*, 5th ed. Prentice Hall, 2010.

[13] K. S. Chong and L. Kleeman, "Accurate odometry and error modelling for a mobile robot," in *Proceedings of International Conference on Robotics and Automation*, vol. 4. IEEE, 1997, pp. 2783–2788.

[14] A. Tewari, *Atmospheric and space flight dynamics*. Springer, 2007.

[15] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*, 2nd ed. MIT Press, 2017, no. ISBN 978-0-262-53381-2.

[16] J. Borenstein and L. Feng, "Measurement and correction of systematic odometry errors in mobile robots," *IEEE Transactions on robotics and automation*, vol. 12, no. 6, pp. 869–880, 1996.

[17] OpenCV, "Camera calibration," 2021, accessed: 2021-05-31. [Online]. Available: https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html