



Bachelor of Engineering with Honours in Mechanical Design and Manufacturing Engineering

MME3191 Capstone Project AY2020/21

Final Report

Waiter! Campaign Please!

Date of Submission: 30 July 2021

Name	Lin Mei Ling @ Alis Lin
Admin No.	1801796

Blank Page

SINGAPORE INSTITUTE OF TECHNOLOGY – NEWCASTLE UNIVERSITY
MME3191 Capstone Project Report Submission Form
Declaration of Authorship

Name of Student: Lin Mei Ling @ Alis Lin	Student ID Number: 1801796
I am submitting the report/thesis as:	
<input checked="" type="checkbox"/> an individual work	
Degree: Mechanical Design and Manufacturing Engineering	
Capstone Project Title: Waiter! Campaign Please!	
Faculty Supervisor : A/Prof Michael Lau Dr Edwin Foo	Organization : SIT / NU
Industry Supervisor (if applicable) : -	Organization : -

I hereby confirm that:

1. this work was done wholly or mainly while in candidature for the SIT-NU joint degree programme;
2. where any part of this capstone project has been previously submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. I have acknowledged the use of all resources in the preparation of this thesis / report;
4. the thesis / report contains/does not contain* Company's proprietary and/or confidential information. I have sought approval from my Industry Supervisor for the use of the Company's information (if any) for my capstone project;
5. the work was conducted in accordance with the Research Integrity Policy and ethics standards of SIT and that the research data are presented honestly and without prejudice. The SIT Institutional Review Board (IRB) approval number is _____ (where applicable);
6. I have read and understood the University's definition of plagiarism as stated in the SIT Academic Policy Section 14: Academic Integrity

Plagiarism is the copying, using or passing off of another's work as one's own work without giving credit to the author or originator, and also includes self-plagiarism. For example, reusing, wholly or partially, one's previous work in another context without referencing its previous use.
7. the thesis / report has been checked by Turnitin. I attach the signed Turnitin Originality Report.

Signature of Student

30 July 2021

Date

Acknowledgement

I would like to express my utmost appreciation to Dr. Michael Lau and Dr. Edwin Foo for their constant guidance and advise throughout the project timespan. Their patience and dedication provided me with a valuable and eye-opening experience. Thank you for allowing me to take on this project and to grow as an engineer.

I would also like to express my gratitude towards Wan Saw Yang Ash as a friend and a mentor who provided me with insights to the world of robots and programming. Thank you for your valuable guidance.

Blank Page

Abstract

In the food and beverage industry, countless types of waiter robots were being implemented for one purpose, to only serve food to their customers. Liquids such as soups and beverages would be served by human waiters. To increase the versatility of the robot, it should be able to serve both food and beverages in a restaurant. The waiter robot should be able to move smoothly in a planar motion without spilling the liquids on the tray. Tapping on the versatility portion, robots can be implemented to serve beverages in cocktail settings as well. In this scenario, the robot should be able to manoeuvre and avoid obstacles throughout the open ballroom while serving beverages without spilling.

In the literature review, the development of velocity planning, smooth trajectory, autonomous planning, and the reactive autonomy of robots were reviewed. Various algorithms were elaborated on and its suitability for the robot was stated.

The methodology describes the equations and algorithms used for the smoothing velocity profile. The design of 2 different state machines: fixed profile state machine and autonomous cocktail robot state machine were elaborated on with their specific functions

In the experiment, it was proven that the velocity smoothing works as no spillage were observed from the 2 state machine designs. The autonomous cocktail robot was capable of moving autonomously with obstacle avoidance. In conclusion, the objective of the project was achieved.

Table of Contents

Acknowledgement	ii
Abstract	iv
1 Introduction.....	1
1.1 Current Implementation of Waiter Robot in F&B Industry	1
1.2 Current Development of Waiter Robot in F&B Industry.....	2
1.3 Waiter Robot Set-Up: Hardware and Software.....	3
1.4 Objective	4
2 Literature Review.....	5
2.1 Development of Velocity Planning	5
2.2 Development of Trajectory Planning	5
2.3 Reactive Autonomy.....	6
2.4 Development of Autonomous Planning	6
3 Proposed Methodology	7
3.1 Comparison between S-Velocity Profile and Ramp Velocity Profile	7
3.2 Modelling Constant Jerk	7
3.3 Manhattan Distance Vectors for Planar Implementation.....	8
3.4 Fixed Profile State Machine (Rhombus).....	9
3.5 State-Machine for Autonomous Cocktail Robot.....	11
3.6 Sensor Reading Classification	12
3.7 Vector Selection via Random Generator	12
3.8 Vector Confirmation via Range Detection	13
3.9 Obstacle Detection Behaviour	13
4 Results.....	14
4.1 Experimental Set-Up.....	14
4.1.1 Rhombus Experiment.....	15
4.1.2 Autonomous Cocktail Robot Experiment.....	15
4.2 Data Collection Method.....	16
4.2.1 Rhombus Experiment.....	16
4.2.2 Autonomous Cocktail Robot Experiment.....	16
4.3 Experiment Results	17
4.3.1 Rhombus Experiment.....	17
4.3.2 Autonomous Cocktail Robot Experiment.....	18
5 Discussion	21
5.1 Performance of smooth trajectory.....	21
5.2 Controlling S-Curve Parameters using programmable features	21
5.3 Tuning of Classification Range of Sensor	21

5.5 Advantages of Proposed Solution for Cocktail Robot Application	22
6 Conclusions.....	23
6.1 Summary	23
6.2 Future Work.....	23
7 References.....	i
Appendix.....	iii
Appendix A – Gantt Chart	iii
Appendix B – ROS Graph	iv
Appendix C – Navigation Launch Folder (XML file)	v
Appendix D – Rhombus State Machine.....	ix
Appendix E – Autonomous Cocktail Robot State Machine	xxiii
Appendix F – Work from Home Risk Assessment.....	xlii
Appendix G – Laboratory Risk Assessment.....	xlvii

List of Figures

Figure 1: Waiter Robot serving prepackaged dishes in Haidilao. [5]	1
Figure 2: Waiter robot serving drinks in Dadawan. [2]	2
Figure 3: Spot serving drinks in a bar in Seville, Spain.[7]	2
Figure 4: Current Set-up of the waiter robot with its specifications [8]	3
Figure 5: Graphical comparison between the S-velocity profile and ramp velocity profile.....	7
Figure 6: Equation 1 (left) and Equation 2 (right)	7
Figure 7: Representation of Manhattan Distance in red, blue, and yellow.....	9
Figure 8: Manhattan Distance algorithm implementation in State Machine	9
Figure 9: Fixed Rhombus Profile Motion	10
Figure 10: State Machine View for the Rhombus Profile Motion.....	10
Figure 11: State Machine Sequence.....	11
Figure 12: IR sensor placement on the base of the robot.....	12
Figure 13: Callback code for the IR sensor	13
Figure 14: Physical Robot Set-Up	14
Figure 15: Layout of Rhombus Experiment	15
Figure 16: Set-Up 1.....	15
Figure 17: Set-up 2.....	15
Figure 18: Velocity Plot of the Fixed Profile.....	17
Figure 19: Calculated Distance versus Measured Distance in meters	17
Figure 20: Time taken to complete motion and whether spillage was present.	18
Figure 21: Graphs exhibiting the estop and docking behaviour for Test 1 to 3.....	18
Figure 22: Graphs exhibiting the estop and docking behaviour for Test 4 to 6.....	19
Figure 23: Representation of the actual motion travelled in Test 6	19
Figure 24: Data showing the Actual travelled distance against AMCL travelled distance, and data indicating if water was spilled.....	20
Figure 25: Sensor data conversion code	21

1 Introduction

1.1 Current Implementation of Waiter Robot in F&B Industry

In this current day and age, industrial sectors have been integrating automation to their operations, such as robotic automation and the Industrial Internet of Things (IIOT). Robotics automation has been a relatively popular option in the manufacturing industry to facilitate and improve its operating processes for years.[1] In recent years, the food and beverage (F&B) industry has been tapping on the robotic automation technology to improve their services. In the initial stage, robots were seen as marketing gimmicks rather than a service tool to aid human waiters in their services. These led to a mix in success, where robots successfully replaced humans and being replaced themselves.[1] These waiter robots have been implemented across restaurants around the world, in countries such as China[1], Netherlands[2], United Kingdom[3], and Singapore[4].

In 2018, one of the biggest hotpot chain restaurants, Haidilao, opened a smart restaurant in Beijing, China. This service-oriented restaurant implemented robotics arms to retrieve pre-packaged food from the cold storage and waiter robots, in Figure 1, to convey food to the tables. Human waiters were able to focus on attending to the needs of their customers or only required for ushering services. It took approximately 3 minutes for the waiter robot to arrive with the orders upon its placement through the iPad. Additionally, instructions can be provided for the robot to convey emptied dishes from the tables to the kitchens.[5]



Figure 1: Waiter Robot serving prepackaged dishes in Haidilao. [5]

In 2017, Pin Xian Lou in Singapore introduced 2 humanoid robots to their restaurant which successfully boosted their business. Thus, the immense demand for their waiting services to serve food were inevitable. In addition, the constraint in space was a point of concern as it disrupts the service of the waiter robot. All these issues led the robot to become a form of entertainment for customers, and losing its purpose as a waiter.[4]

In light of the COVID-19 pandemic, waiter robots have been integrated into countless restaurants all over the world to stand out among their competitors.[3] Restaurants were required to take on extra tasks to curb the spread of the virus, such as temperature taking, disinfection of tables, the limitation of human-to-human interaction, and ensuring social-distancing between tables. In June 2020, Dadawan, a restaurant in Netherland, hired 3 robots to take on these additional service-related tasks together with their waiting services. This implementation successfully achieved its purpose by aiding human waiters and helping to curb the spread of the virus. Additionally, the waiter robots was capable of serving beverages to customers.[2]



Figure 2: Waiter robot serving drinks in Dadawan. [2]

In November 2020, a bar in Sevilla, Spain, employed a legged Boston Dynamics waiter robot, Spot, for contact-free serving. The motion of the robot was controlled via a controller and was able to serve beers without spilling as shown in Figure 3.[6] However, wheeled mobile robots taking on the role as cocktail robots to serve drinks are yet to be developed.



Figure 3: Spot serving drinks in a bar in Seville, Spain.[7]

1.2 Current Development of Waiter Robot in F&B Industry

In the F&B industry, countless types of waiter robots were being implemented primarily used to serve food to their customers. Liquids such as soups and beverages were often served

by humans. If the waiter robots do convey liquid, the cups were filled below the standard level or covers were used.

Waiter robots should be able to serve food and beverages like human waiters. Humans would modify their pace when carrying liquids. To improve productivity and increase the versatility of the robot, it should be able to serve both food and beverages in a restaurant. The waiter robot should be able to move smoothly in a planar motion without spilling the liquids on the tray. To accomplish this motion, the velocity profile in the robot operating system (ROS) was modified by limiting jerk to obtain a smooth S-curve velocity profile. Motion designs for the congested and space-constrained restaurants were often not continuous and could possibly be short zig-zag motions.

1.3 Waiter Robot Set-Up: Hardware and Software


	<table border="1"> <thead> <tr> <th>Items</th><th>Specifications</th></tr> </thead> <tbody> <tr> <td>Sensors</td><td>Array of Infra-red, TeraBee and a LiDAR</td></tr> <tr> <td>Power</td><td>24 V@5A</td></tr> <tr> <td>Battery</td><td>Lithium 24 V self-charging (duration min 40 mins)</td></tr> <tr> <td>Motor Com</td><td>TTL serial</td></tr> <tr> <td>Wireless</td><td>Wifi IEEE 802.11 n</td></tr> <tr> <td>Processor</td><td>Intel i7 NUC</td></tr> <tr> <td>Wheels</td><td>Omni-direction (3 wheels) - 3 Robotis MX106T</td></tr> <tr> <td>Speed</td><td>Max forward speed: 0.5 m/s</td></tr> <tr> <td>Payload</td><td>10 kg</td></tr> <tr> <td>Tray</td><td>3 Tier (maximum)</td></tr> </tbody> </table>	Items	Specifications	Sensors	Array of Infra-red, TeraBee and a LiDAR	Power	24 V@5A	Battery	Lithium 24 V self-charging (duration min 40 mins)	Motor Com	TTL serial	Wireless	Wifi IEEE 802.11 n	Processor	Intel i7 NUC	Wheels	Omni-direction (3 wheels) - 3 Robotis MX106T	Speed	Max forward speed: 0.5 m/s	Payload	10 kg	Tray	3 Tier (maximum)
Items	Specifications																						
Sensors	Array of Infra-red, TeraBee and a LiDAR																						
Power	24 V@5A																						
Battery	Lithium 24 V self-charging (duration min 40 mins)																						
Motor Com	TTL serial																						
Wireless	Wifi IEEE 802.11 n																						
Processor	Intel i7 NUC																						
Wheels	Omni-direction (3 wheels) - 3 Robotis MX106T																						
Speed	Max forward speed: 0.5 m/s																						
Payload	10 kg																						
Tray	3 Tier (maximum)																						

Figure 4: Current Set-up of the waiter robot with its specifications [8]

The set-up for hardware and software were extracted from a paper written by Wan et al. This robot with a steel frame was built as an improvement from the Beta-G to become lighter in weight and smaller in size. An open double-tiered tray design provides the ease of insertion and removal of the trays, as well as accessibility by users. The circular 3-Swedish 90° wheeled base provides omnidirectional movement during operation. Furthermore, the robot is capable of conveying loads of up to 10kg.[8] The electrical components have been specified in Figure 4.

The navigation portion of the waiter robot runs on the Robot Operating System (ROS) in Linux. The development of the ROS software modules: Hector SLAM to generate the map,

the navigation stack, “move base” node to publish waypoints, local planners, and finite state machine was completed by Ash et al.[8]

1.4 Objective

Simple short path segments and shapes with a smooth velocity profile to imitate movement from point A to point B while conveying liquids without spillage in a planar motion will be solved. In the motion design, obstacle avoidance must be implemented. Tapping on the versatility portion, robots can be implemented to serve beverages in cocktail settings before a formal dinner.

In this scenario, the cocktail waiter robot should be able to manoeuvre and avoid obstacles throughout the open ballroom smoothly while serving beverages without spilling. The speed should be appropriate enough for humans to lift off and place their glass onto the robot in motion.

2 Literature Review

2.1 Development of Velocity Planning

Within the robotics industry, the most common optimisations were the time-optimal trajectory for productivity and jerk-minimisation trajectory for velocity smoothing.[9] Guarino Lo Bianco et al. applied the minimum-jerk velocity planning to solve feasibility issues in wheeled mobile robots.[9] In another paper, Guarino Lo Bianco et al. used the inversion-based control scheme for velocity smoothing of autonomous vehicles.[10] Lini et al., applied the path-velocity decomposition to obtain a smooth velocity profile.[11]

2.2 Development of Trajectory Planning

Barghi Jond et al., applied the high-order polynomial trajectory for a two-wheeled differential drive (DD) robot for an optimal time-distance planning.[12] The optimisation for time-distance was achieved, but simulations were only conducted for DD robots. Lu et al. used the augmented Lagrange constrained particle swarm optimisation (ALCPSO) which considers jerk and time for optimisation.[13] However, experimentations were not done. Its feasibility in actual application remains unknown. Fang et al. applied a novel S-curve trajectory planning for multi-axis motion synchronisation which is based on a piecewise sigmoid jerk function. [14]

All the concepts stated were capable of minimising jerk, however, complicated calculations were involved. This makes it computationally expensive. Furthermore, the experimentations of simulations were conducted using robot arm manipulators that work in static and controlled environments. On the other hand, mobile robots work in dynamic environments and cannot afford the computationally expensive algorithms that affects its response during operation. Thus, the Wan et al. developed a liquid-conveying algorithm that was based on the S-curve. [1]

Wan et al. made a comparison between the ROS navigation move base and a ramp velocity profile.[1] The ROS navigation move base produced a step velocity, where instantaneous change in velocity occurs at the start and end of the point. While the ramp velocity has a gradual increment and decrement, exhibiting a smoother velocity profile. These 2 profiles were tested and compared. Spillage did not occur for the ramp velocity profile while it occurred for the ROS navigation move base. In the paper, Wan et al. developed a simple computation approach that will be explained in the methodology section.[1]

2.3 Reactive Autonomy

Reactive behaviours in a robot refers to an event that causes a reaction that changes the behaviour of the robot.[15] Mura et al. [16] and Sun et al. [17] utilised the zig-zag algorithm for their respective terrestrial mobile robots. Together with IR sensors, the obstacle avoidance function can be integrated together with this algorithm, allowing the robot to move in tight spaces.

Ismail et al. [18], Punetha et al.[19], and Ghani et al.[20] utilized the line following control algorithm for their respective mobile robots. Gavrilut et al.[21] and Ando et al.[22] utilized the wall following algorithm with the aid of IR sensors and sonar ring respectively. Laying out a line track for an unpredictable cocktail event would be time-consuming and a waste of material. If the event attendees stood right on top of the line, the robot will become stuck. In an open ballroom, the robot might get lost while trying to find a wall. Therefore, both the line tracking and wall tracking are unsuitable for cocktail robot.

Schillaci et al.[23] and Palmieri et al.[24] utilized the random walk approach for the purpose of robot navigation. This approach was implemented for a dynamic environment for robot navigation. Ultimately, the most suitable algorithm for the cocktail robot is still the zig-zag algorithm.

2.4 Development of Autonomous Planning

In autonomous path planning, different algorithms are available to solve shortest path problem. The greedy search algorithm is used to obtain the shortest path in the problem in any given path, where it can be used for routing. The A* algorithm employs the heuristic approach to obtain the estimated distance between the current position and the goal position. During the process, only the relevant nodes will be visited to obtain the shortest path.[25]

It can be deduced that these 2 algorithms are not suitable for the cocktail robot. In the ballroom, the robot must be able to move freely to serve people. However, to move towards the shortest path, a goal must be present, further proofing its unsuitability. As stated previously, the zig-zag algorithm is the most suitable method.

3 Proposed Methodology

3.1 Comparison between S-Velocity Profile and Ramp Velocity Profile

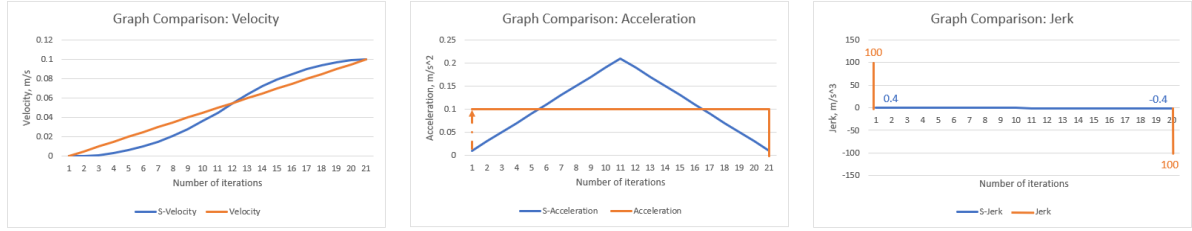


Figure 5: Graphical comparison between the S-velocity profile and ramp velocity profile

In Figure 5, the comparison graphs show a ramp velocity change of 0.1 m in 1 second, a step change in acceleration, leading jerk to exhibit a magnitude of 100 m/s³ at the iteration number, 1 and 21. In contrast, the S-velocity profile shows a gradual increase in velocity until it reaches the maximum acceleration at time interval 11. From this iteration, the velocity will continue to gradually increase to 0.1 m/s as acceleration decreases linearly. Therefore, jerk has a magnitude of 0.4 m/s³ then -0.4 m/s³ that spreads the change in acceleration over time. Thus, no spillage should be observed when utilising the S-curve.

3.2 Modelling Constant Jerk

This portion focuses on the theoretical calculation to obtain the actual displacement, acceleration, and jerk based on the velocity values. As mentioned by Wan et al., the S-curve has been hardcoded in Python with incremental operations in Figure 6.[1] The python codes were matched with its respective equations, where Eqn.(1) refers to the change in velocity per time step and Eqn.(2) refers to the change in acceleration per time step. [1]

<p>Python Code for Eqn.(1):</p> <pre>cmd_vel.linear.x = cmd_vel.linear.x + dvx</pre> <p>Equation (1):</p> $v_i = v_{i-1} + dv_i$ <p>where v_i refers to the current velocity command v_{i-1} refers to the previous velocity command dv_i refers to change in velocity command</p>	<p>Python Code for Eqn.(2):</p> <pre>dvx = dvx + j</pre> <p>Equation (2):</p> $dv_i = dv_{i-1} + j .sgn(j)$ <p>dv_{i-1} refers to change in velocity command where $sgn(j) = 1$ for the concave part of the S-Curve and $sgn(j) = -1$ for the convex part of the S-Curve</p>
---	--

Figure 6: Equation 1 (left) and Equation 2 (right)

Integrating of velocity to obtain displacement is represented by the trapezoidal rule. When an autonomous robot is travelling from the initial position to the goal position, it might be challenging to measure the actual distance travelled. Thus, the actual distance can be calculated. The trapezoidal rule for each time step, is given by,

$$I = \Delta t \frac{f(X_i) + f(X_{i-1})}{2}$$

The jerk coefficient in the Python code was calculated using the first derivative of command velocity, more commonly known as the central difference method. It was also used to calculate the actual acceleration based on the command velocity. The central difference method for each time step, is given by,

$$f'(X_i) = \frac{f(X_{i+1}) - f(X_{i-1})}{2\Delta t}$$

By applying and derivative approximation to the central difference method, the actual jerk value can be obtained. The second derivative of command velocity for each time step, is given by,

$$f''(X_i) = \frac{f(X_{i+1}) - 2f(X_i) + f(X_{i-1}))}{\Delta t^2}$$

Where Δt refers to the change in time.

$f(X_i)$ refers to the current command velocity.

$f(X_{i-1})$ refers to the previous command velocity.

$f(X_{i+1})$ refers to the next command velocity.

3.3 Manhattan Distance Vectors for Planar Implementation

The smooth velocity profile has been successfully implemented in the 1-Dimensional (1D) motion for the X and Y direction respectively. With the omnidirectional wheels and restaurant implementation, 1D motion would be insufficient. Thus, there was a need to create a smooth planar velocity profile. An attempt was made by the predecessor, where the twist command for the X-axis and Y-axis motion were split into two different nodes. Each 1D motion was computed separately with the nodes running in parallel. [26]The resulting motion was still jerky. However, it proves that planar motion smoothing is possible.

The main challenge was to maintain a constant jerk while having a smooth planar motion. To achieve this, the Manhattan Distance heuristic algorithm, also known as L1 norm was implemented. This algorithm is the summation of all distances between the starting position and the goal position.[27] It was stated that the algorithm works similarly to the Pac-Man game. In Figure 7, the starting position and goal are represented by black circles and the line in red, blue, and yellow represents the possible solutions using Manhattan distance

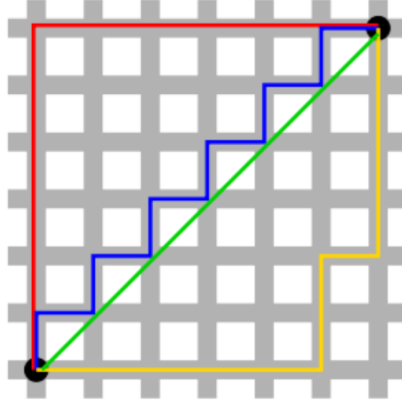


Figure 7: Representation of Manhattan Distance in red, blue, and yellow.

In this scenario, both acceleration and deceleration begin with the increment or decrement of its respective X-direction state, followed by the Y-direction. At each increment or decrement step, only one directional state is active at a time and jerk coefficient would be constant at 0.0001 m/s^3 . Resulting in a smooth planar motion to allow the transportation of liquids without spilling.

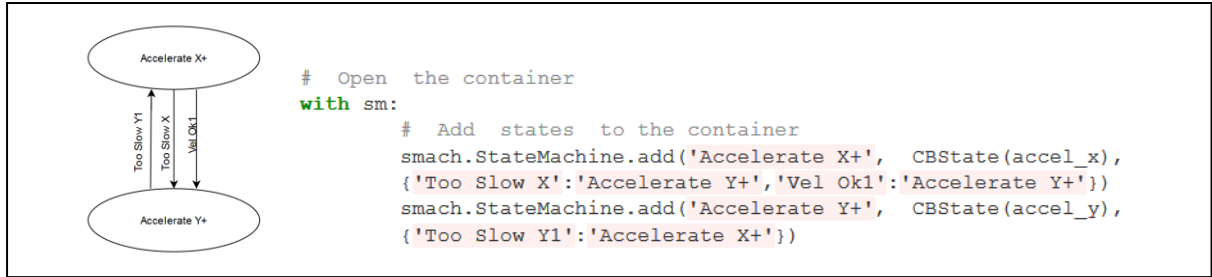


Figure 8: Manhattan Distance algorithm implementation in State Machine

To further elaborate on the explanation, Figure 8 clearly illustrates this implementation in a state machine. In the first program loop, Acceleration X+ state returns an outcome 'Too Slow X' which transitioned to the Acceleration Y+ state. In the subsequent loop, 'Too Slow Y1' would be returned from the Acceleration Y+ state, it would transition back to the Acceleration X+. This algorithm would be applied to the fixed profile and autonomous cocktail robot state machines.

3.4 Fixed Profile State Machine (Rhombus)

The Twist command for the x-axis and y-axis runs on a single node using the Manhattan Distance algorithm to generate smooth planar motion with a constant jerk. A fixed profile state machine for a rhombus shaped motion was generated in ROS to demonstrate this concept. The trajectory demonstration works in the manner illustrated in Figure 9, where it begins and ends at the starting point.

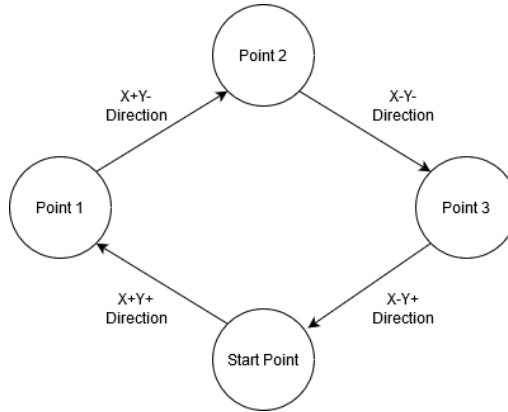


Figure 9: Fixed Rhombus Profile Motion

Within this state machine, a single planar motion direction consists of the acceleration and deceleration state for the x-axis and y-axis. At each point, the robot would start to accelerate towards its planar direction until it reaches 0.1 m/s. Then it will immediately start to decelerate in the same direction until command velocity is zero. At this instance, the robot would have arrived at its respective points. This cycle will repeat until the rhombus shaped trajectory is completed.

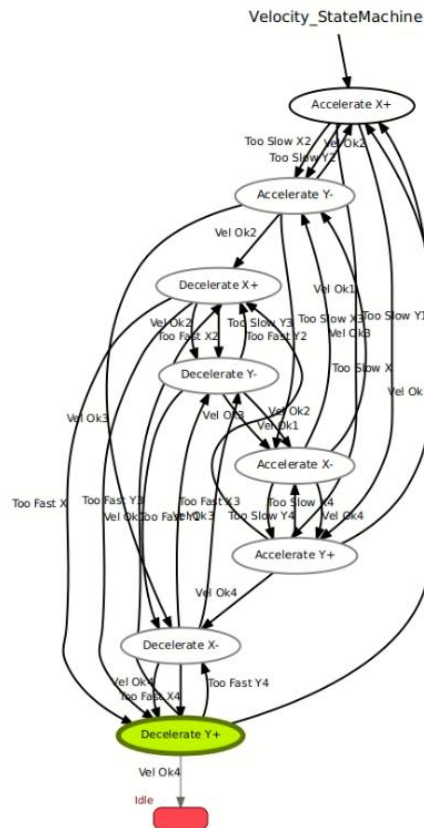


Figure 10: State Machine View for the Rhombus Profile Motion

This state machine transition states using the Manhattan Distance algorithm illustrates that it is possible for the robot to convey liquids in a planar motion without spilling. In addition, the states used in this state machine will be implemented in the planar directional motion for the autonomous cocktail robot.

3.5 State-Machine for Autonomous Cocktail Robot

The waiter robot conveying food and beverages in a restaurant can be converted into a cocktail robot that serves beverages before a formal dinner. In an open ballroom, the autonomous robot should be able to detect obstacles while conveying liquid without spillage. To enable robot autonomy and proximity sensing, infrared (IR) sensors were integrated into this solution. As a result, a state machine was generated.

When human waiters receive a beverage order, motion planning to their goal would be done. Upon receiving the beverage, the waiter would gradually increase their walking speed until they reach an optimal walking speed. Once the goal is in sight, they would start to decrease their speed, then stop at the goal to serve. Thus, the state machine sequence for the cocktail robot in Figure 11.

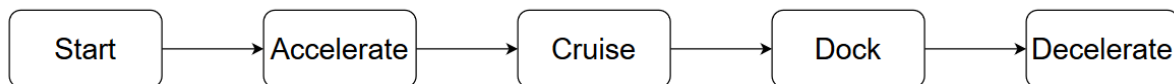


Figure 11: State Machine Sequence

All the decisions for the direction of motion were made in the Start state. Approximately 50 program loops were required for the IR to start up. A random generator was embedded in the first loop to generate a single integer, n , from 1 to 8. Each ' n '-value has been assigned a direction of motion. The reason for the chosen algorithm will be further elaborated in the discussion segment. From loop 51 to 53, the average of the sensor reading that corresponds to the ' n '-value will be obtained. If the average is more than or equal the current sensor reading, it will then transition to its respective acceleration states based on the ' n '-value. Else, the robot will go to the X+ acceleration state, the forward direction.

The robot will remain in the acceleration state until command velocity for its corresponding direction reaches ± 0.1 m/s. At the command velocity of approximately ± 0.1 m/s, it will transition to the cruise state. If an obstacle appears within the set range of the IR, the emergency-stop (estop) will be activated. Command velocity will become zero and return to the Start state. Furthermore, estop were implemented in the cruise and deceleration state as well.

The robot will remain at cruise state with a fixed velocity obtained from the acceleration state. If a stationary obstacle within the secondary set range was detected or the docking radius is lesser than 0.2m, the robot will transition to the dock state.

The dock state would instruct the robot to transition to its respective deceleration states. If the current position is equivalent to the goal position, the robot will remain docked. Else if the average of the sensor reading is lesser than 0.2m and time is 200, it returns to the start state.

In the deceleration state, the robot will remain in this state until command velocity is approximately zero. At zero, the state machine ends. If the robot is still required, an operator must start it up again.

3.6 Sensor Reading Classification

Sensor reading classification is defined as the categorisation of the data obtained by the sensor. The classification stems from the desired thresholds for estop and deceleration to occur. When a black object was detected or an obstacle is out of the detectable range, the IR sensor tends to read a value of 'inf'. Therefore, string 'inf' was converted into the maximum integer available in the system using the sys.maxint function in Python. To prevent collision during the experimentation phase, black-coloured items were removed from the test area.

3.7 Vector Selection via Random Generator

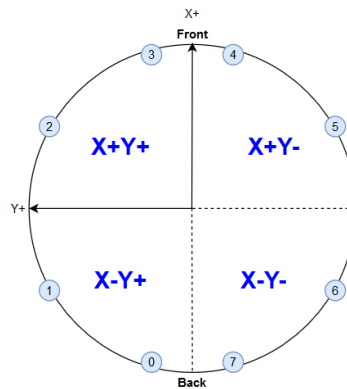


Figure 12: IR sensor placement on the base of the robot

Vector selection in the Start state was achieved via the random generator in the first loop. The random generator only generates an integer from 1 to 8. Figure 12 displays the IR sensor placement, aided in the assignment of the distinct direction of motion for the 'n'-values. Once the random 'n'-value has been obtained, only the readings obtained from the corresponding IR will be used, as illustrated in Figure 13.

The 'n'-values are as follows:

1. If $n = 1$, then the robot moves in the X-Y+ direction.
2. If $n = 2$, then robot moves in the X+Y+ direction.
3. If $n = 3$, then robot moves forward, X+ direction.
4. If $n = 4$, then robot move backwards, X- direction.
5. If $n = 5$, then robot moves in X+Y- direction.
6. If $n = 6$, then robot moves in X-Y- direction.
7. If $n = 7$, then robot moves left, Y+ direction.
8. If $n = 8$, then robot moves right, Y- direction.

```
## Callback Section
def callback1(msg):
    global i, min_int, ProxMax, sensor, n, IRmax, IRinf
    for i in range(8):
        if msg.ranges[i].range == float('inf') or msg.ranges[i].range == float('-inf'):
            msg.ranges[i].range = IRinf
    if n == 1:#X-Y+
        if msg.ranges[1].range == IRinf or msg.ranges[1].range > 0:
            ProxMax = msg.ranges[1].range
    if n == 2:#X+Y+
        if msg.ranges[2].range == IRinf or msg.ranges[2].range > 0:
            ProxMax = msg.ranges[2].range
    if n == 3:#forward
        if msg.ranges[3].range == IRinf or msg.ranges[3].range > 0 or msg.ranges[4].range == IRinf or msg.ranges[4].range > 0:
            ProxMax = msg.ranges[3].range
    if n == 4:#backward
        if msg.ranges[0].range == IRinf or msg.ranges[0].range > 0 or msg.ranges[7].range == IRinf or msg.ranges[7].range > 0:
            ProxMax = msg.ranges[0].range
    if n == 5:#X+Y-
        if msg.ranges[5].range == IRinf or msg.ranges[5].range > 0:
            ProxMax = msg.ranges[5].range
    if n == 6:#X-Y-
        if msg.ranges[6].range == IRinf or msg.ranges[6].range > 0:
            ProxMax = msg.ranges[6].range
    if n == 7:#left
        if msg.ranges[1].range > 0 and msg.ranges[2].range > 0:
            ProxMax = (msg.ranges[1].range + msg.ranges[2].range)/2
    if n == 8:#right
        if msg.ranges[5].range > 0 and msg.ranges[6].range > 0:
            ProxMax = (msg.ranges[5].range + msg.ranges[6].range)/2
```

Figure 13: Callback code for the IR sensor

3.8 Vector Confirmation via Range Detection

The vector confirmation via range detection by the IR sensor would be explained with an example. Assuming that the robot is enclosed such that it can only move forward, where 'n'-value is 3. In this case, if the 'n'-value is random, the robot will continue to complete the Start state. As it transitions to the respective acceleration, estop should be activated to transition back to Start state. This cycle will repeat until 'n'-value is 3, then acceleration will begin.

3.9 Obstacle Detection Behaviour

After vector confirmation, the room bounded robot would bounce about walls and obstacles indefinitely. This behaviour is the act of the estop. If the robot is still accelerating or decelerating and an obstacle is less than 0.3m away from the robot, estop will be activated. Once activated, the robot comes to a stop while being transitioned back to the start state. Repetition will occur until the cruise state.

In the cruise state, there is an estop and a docking condition. The estop condition remains the same where it would be triggered if an obstacle is less than 0.3m away from the

direction of motion. Docking is bounded by two conditions: if IR detects less than 0.8m from the stationary obstacle or the docking radius is less than 0.2m. If one of the two docking conditions is true, a transition to the deceleration state will occur.

In summary, the estop will be triggered if obstacle is less than 0.3m and 0.8m for docking. The estop distance factors in the reaction time of mechanical wheels before the robot comes to a complete halt. Docking requires a sufficient distance to allow the robot to decelerate fully. In this case, reaction time, wheel slippage, and decelerating distance were factored into the selected distance. Through trial and error, it was determined that 0.3m and 0.8m were suitable distances.

4 Results

This portion exhibits the experimental set-up, collected data, and results of two distinct experiments - Rhombus Experiment, and Autonomous cocktail robot experiment. Both experiments share a common aim of delivering liquids without spilling while travelling in all directions. Furthermore, the autonomous cocktail robot experiment aims to exhibit its autonomy based on the IR sensor detection. The data collected for both experiments vary, they would be evaluated accordingly.

4.1 Experimental Set-Up

Both experiments share a common physical set-up of the robot, this is demonstrated in Figure 14. The nodes involved in both experiments were placed in Appendix B. The following assumptions were made for both experiments: water level in the cup, starting position, and battery voltage of the robot remains constant throughout.

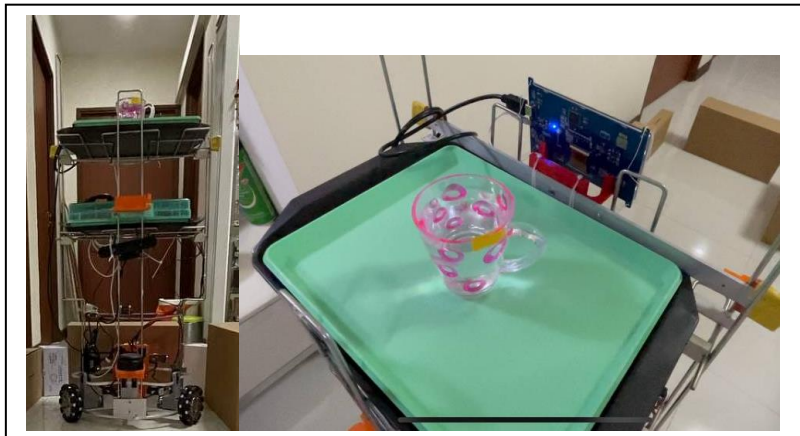


Figure 14: Physical Robot Set-Up

4.1.1 Rhombus Experiment

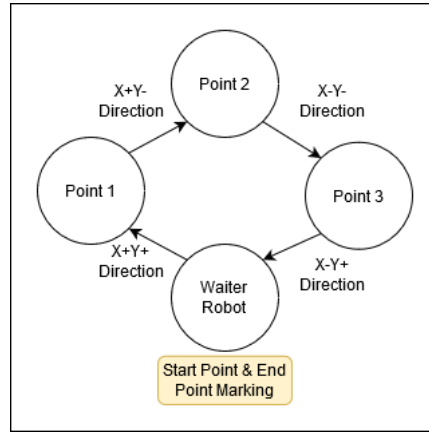


Figure 15: Layout of Rhombus Experiment

To ensure a smooth experiment execution, the 2x2 metre test area was kept obstacle-free, shown in Figure 13. A marking for the starting point was made on the ground where the robot will be positioned at. Then it will move according to the fixed profile. Additionally, the terminals for the launch file, state machine file, and the rosbag must be ready.

4.1.2 Autonomous Cocktail Robot Experiment

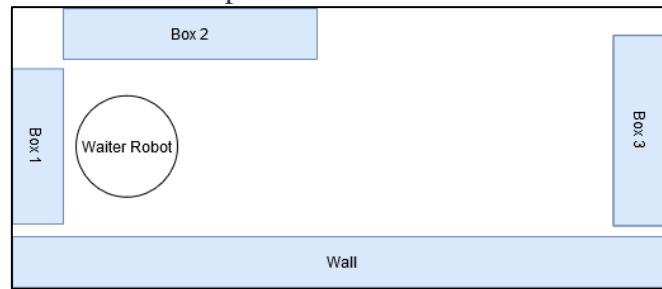


Figure 16: Set-Up 1

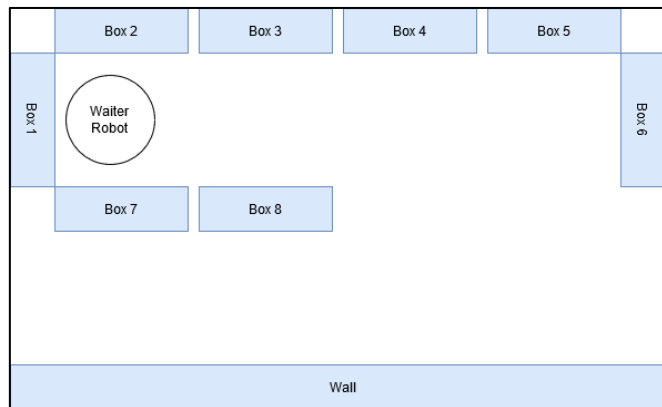


Figure 17: Set-up 2

The terminals for the launch file, state machine file, and rosbag must be ready before the experiment. Boxes were strategically placed within the test area to act as obstacles. Two different set-ups were created for this test as shown in Figure 16 and 17. The robot will be

positioned at the starting point, situated 10cm from Box 1. This placement provides constraints to force the robot to move forward upon start up.

In set-up 1, 5 tests took place in a hallway and the robot was constrained to move forward until it detects Box 3. The 4 tests had varying distances: 4m, 2.4m, 1.8m and 1.2m, to exhibit the functionality of the estop, docking functions, and facilitate in data collection for the actual distance travelled by the robot.

In the other set-up, the robot had a larger area to run in a random and autonomous manner. A single test was conducted with this set-up, as the remaining functions were exhibited by the tests in set-up 1. The robot was constrained to move forward until it detects Box 6. After this, a new random direction will be generated and executed by the robot accordingly. This repeat until it comes to a stop at a wall.

4.2 Data Collection Method

4.2.1 Rhombus Experiment

In the state machine, a topic was created to record the command velocity and position of the robot for x-axis and y-axis. Rosbag was executed to record these values. Command velocity for the x-axis and y-axis were plotted to observe the S-curve plot. During the experiment, markings were made on the ground from Point 1 to the ending point. These markings were manually measured to obtain the actual distance travelled. These were then compared to the distance recorded in rosbag. Lastly, the qualitative data for water spillage will be recorded as yes for spillage and no for no spillage.

4.2.2 Autonomous Cocktail Robot Experiment

In the state machine, a topic was created to record the command velocity, position, 'n'-value, and its corresponding sensor reading. Rosbag was executed to record these values. Recorded distance must be compared with the actual distance. Actual distance was calculated using the trapezoidal rule. Spillage of liquid during the movement phase, time taken to generate the integer for the forward motion, and whether the robot executes estop or docking according to the IR sensor reading were collected.

The data recorded in the rosbag will be plotted against time. The first graph plot would display the command velocity for x-axis and y-axis with IR reading as the secondary axis. The second plot includes the IR reading and the 'n'-value as the secondary axis. Since the IR sensor data has been converted to the largest system value, it will be replaced with 1.5m to exaggerate the plot for analysis.

4.3 Experiment Results

4.3.1 Rhombus Experiment

Test 1 was conducted to obtain the space required in the test area. The dataset obtained would then act as a benchmark for the remaining tests. Due to the capacity of the battery and the minimum operating voltage of the robot, only a maximum of 6 tests could be completed. This applies to the autonomous cocktail robot experiment as well.



Figure 18: Velocity Plot of the Fixed Profile

As shown in Figure 18, where the command velocity graphs for the x-axis and y-axis were plotted. It was observed that all 6 tests provided the same S-velocity curves. These results were as expected, as it proves that the robot was moving in the correct direction.

The data in the calculated segment in Figure 19 was numerically calculated by the Pythagoras theorem formula, using the values of the x-axis and y-axis position recorded in rosbag. On the other hand, the measured portion was done by using a measuring tape to measure the points marked out for the individual tests. The difference between the calculated distance and measured distance was expected. Since the experiment was executed in a location containing smooth flooring, slippage of the wheels tends to occur during movement. This could also result in the distance observed between the starting point and ending point, the ‘End Point – Start Point’ columns as shown in Figure 19.

Data Table 1	Calculated					Measured				
	X+Y+	X+Y-	X-Y-	X-Y+	End Point - Start Point	X+Y+	X+Y-	X-Y-	X-Y+	End Point - Start Point
Test 1	1.12	1.27	1.43	1.35	0.24	1.49	1.47	1.47	1.48	0.03
Test 2	1.12	1.27	1.69	1.52	0.40	1.48	1.46	1.47	1.48	0.04
Test 3	1.14	1.03	1.30	1.21	0.06	1.49	1.47	1.47	1.48	0.03
Test 4	1.22	1.18	1.40	1.26	0.04	1.48	1.46	1.46	1.48	0.03
Test 5	1.21	1.06	1.22	1.51	0.30	1.48	1.47	1.48	1.48	0.04
Test 6	1.24	1.32	1.85	1.26	0.02	1.48	1.47	1.47	1.48	0.02

Figure 19: Calculated Distance versus Measured Distance in meters

The time was taken to verify if the robot completes the fixed profile within the same timeframe and if water will spill during the movement phase for each test. It can be observed that robot is capable of completing this profile within a range of 1.26 min to 1.28 min. While the robot was completing the fixed profile, no spillage of water occurred. This proves that the Manhattan Distance algorithm works, and the robot is able to convey liquids without spilling.

	Time Taken (min)	Water Spillage?
Test 1	1.26	No
Test 2	1.28	No
Test 3	1.27	No
Test 4	1.27	No
Test 5	1.26	No
Test 6	1.26	No

Figure 20: Time taken to complete motion and whether spillage was present.

4.3.2 Autonomous Cocktail Robot Experiment

5 Tests were conducted for Set-Up 1 to distinctly display the difference between docking and estop. The command velocity for x-axis and y-axis, and the distance reading of the sensor was plotted together. In the graphs for Test 1 and 3, the smooth trapezoidal velocity graph is observed. When the sensor senses less than 0.8m, deceleration occurs until velocity reaches zero. In the graph for Test 2, a distinction between the estop and docking function is observed. When the sensor senses less than 0.3m, the estop occurs and the robot continues to operate. Once the robot enters the cruising state, and sensor senses less than 0.8m, deceleration occurs. This exhibits the behaviour of an autonomous robot.

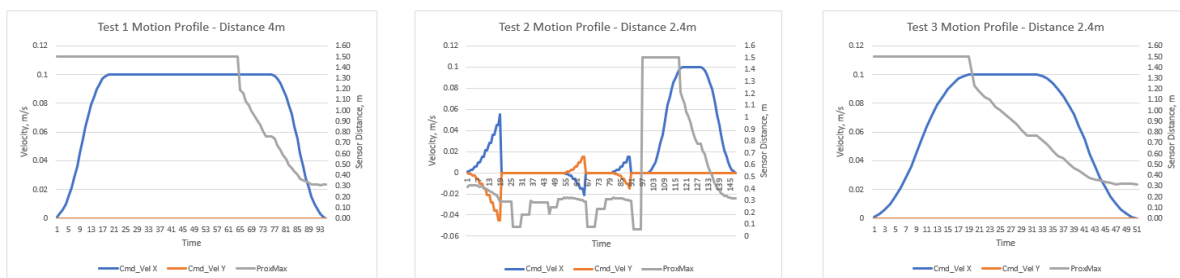


Figure 21: Graphs exhibiting the estop and docking behaviour for Test 1 to 3

In the initial portion of Test 4 and 5, it can be seen that the direction generated by the random generator was blocked by the boxes shown in Set-Up 1. The robot will remain stationary until the 'n'-value becomes 3. The only difference is that Test 4 docks successfully, while Test 5 activates an estop. Test 6 was conducted in Set-Up 2 to exhibit the autonomous robot motion. Both the estop and docking function works according to the set sensor reading.

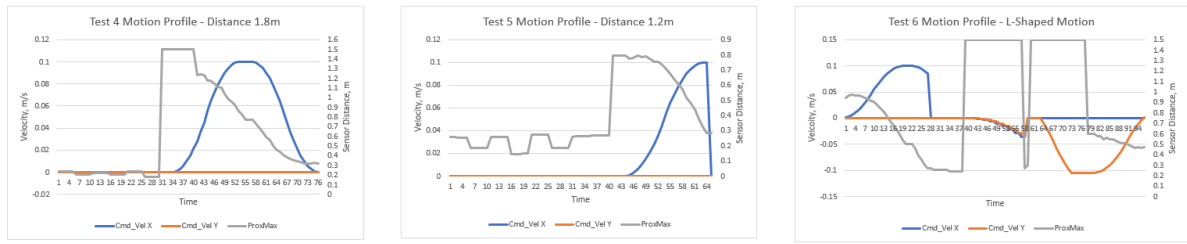


Figure 22: Graphs exhibiting the estop and docking behaviour for Test 4 to 6

The data for the command velocity compared against the sensor readings show that the estop and docking function works with clear distinction. Furthermore, the graph for Test 6 in Figure 22 can aid in the explanation of the trajectory shown in Figure 23.

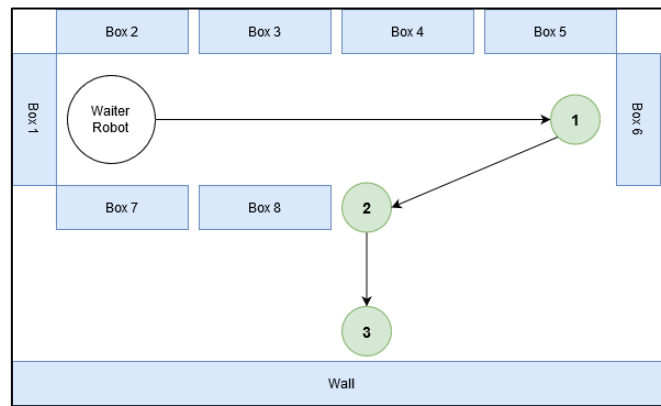


Figure 23: Representation of the actual motion travelled in Test 6

As the robot was travelling to point 1, the IR sensor detected less than 0.8m and decelerated. Before it could decelerate completely, less than 0.3m was detected and estop was activated. A new randomly generated number was then determined, instructing the robot to move towards point 2. Before the acceleration phase was complete, the IR sensor detected less than 0.3m, the estop was activated, and a new random number in the direction of point 3 was generated. During the movement phase, the robot entered the cruise state and less than 0.8m was detected for docking to occur. This time round, there was sufficient space for the robot to fully decelerate and dock.

Due to the random generator, the fully autonomous test for set-up 2 is only repeatable until point 1. After point 1, the motion travelled would differ. Therefore, only one such test was executed to prove that the robot works according to the objective. Furthermore, the current algorithm used for the experiment is only for the proof of concept. In actuality, the robot sequence will not end after the deceleration state.

Since this robot travels randomly, obtaining the total distance travelled was challenging. Therefore, the distance travelled by the robot was calculated. Actual distance was

calculated using the trapezoidal rule, and the adaptive Monte Carlo localisation (AMCL) distance was calculated using the Pythagoras theorem based on data from the amcl node.

A significant difference in distance can be observed. For the Tests 1 to 5, the test area distance was determined and measured physically. Comparing the test area distance to the data obtained, the actual distance travelled has a better representation of the distance travelled. As for the AMCL distance, the data obtained from the amcl node was determined by probability. Thus, it is better to use the trapezoidal rule to calculate determine the actual distance travelled by a robot. Since the set-up for Test 6 was less constrained, it was allowed to move randomly, measuring physically was a challenge. According to the findings, the actual distance travelled by the robot is an accurate representation of the actual distance.

	water spillage?	Actual Distance	AMCL Distance
Test 1	No	3.80	3.00
Test 2	No	1.79	1.45
Test 3	No	1.60	0.75
Test 4	No	1.15	0.30
Test 5	No	0.55	0.31
Test 6	No	0.76	0.67

Figure 24: Data showing the Actual travelled distance against AMCL travelled distance, and data indicating if water was spilled.

This autonomous test was used to verify if the robot can travel autonomously without spilling water to fulfill its role as an autonomous cocktail robot. The data in Figure 24, shows that water did not spill in any of the tests. This proves that the robot is able to fulfill its objective.

5 Discussion

5.1 Performance of smooth trajectory

As observed from the experimentation segment, the performance of the robot meets the requirement as no spillage was observed. The previous implementation of 1D smoothing by the predecessor has no spillage, however, it underutilises the function of the omnidirectional wheel. To move the robot diagonally, rotation was required to change its heading. This motion wastes time and induce risk for spillage. Thus, the planar implementation with no spillage fully utilises the omnidirectional wheel and is considered as an improvement from before.

The robot was capable of moving smoothly and randomly on the ground. With omnidirectional wheels and sensors sensing in 8 directions, rotation is not required. Furthermore, the 20 iterations of the code to create a jerk of 0.001 m/s^3 was sufficient for this set-up. The code iterations were fine enough, and lagging does not occur in the response of the robot.

5.2 Controlling S-Curve Parameters using programmable features

The key control of the S-curve parameters has to do with the change in time of a single program loops or the number of iterations of the program. If the delay function in the code is doubled to tune the jerk coefficient, either the jerk coefficient gets doubled or the number of iterations become double. If the change in time and jerk coefficient are not balanced, the distance travelled might become doubled.

If there is a constraint in space, travelling distance and braking distance based on the limitations of the sensor must be factored in. A tighter space, might increase the jerk coefficient or disallow the increase in time or iterations, leading to spillage. All these considerations has to be made to be able to obtain a properly controlled S-curve.

5.3 Tuning of Classification Range of Sensor

According to the datasheet, the sensor is capable of sensing up to 2m indoor with an accuracy of $\pm 3\text{cm}$. [28] Through data analysis of the sensor readings, it was discovered that the sensor is capable of detecting up to 1.2m. Any detection beyond 1.2m and sensor threshold of 0.1m is defined as 'inf' by the system. To facilitate the range classification process, the 'inf' was converted to the maximum system integer defined as IRinf, shown in Figure 25.

```
for i in range(8):  
    if msg.ranges[i].range == float('inf') or msg.ranges[i].range == float('-inf'):  
        msg.ranges[i].range = IRinf
```

Figure 25: Sensor data conversion code

In the program, the estop is classified as a sensor reading of less than 0.3m and less than 0.8m for the docking to occur. Any detection beyond the threshold of the sensor becomes system maximum integer.

5.5 Advantages of Proposed Solution for Cocktail Robot Application

If the waiter robots available in the F&B industry do convey liquid, the cups were filled below the standard level or covers were used. However, this robot was able to deliver liquids smoothly without the need to of a cover. As observed in the experimentation, there was no spillage of liquids in all tests.

The behaviour of the autonomous robot allows it to roam freely about a ballroom while serving drinks before a formal dinner. The random generator is unbiased, enabling the robot to move in any obstacle-free direction.

The speed of the robot is kept at 0.1 m/s, to allow humans to lift off and place their glass onto the robot in motion. If an obstacle appears within the estop activation range, it will be activated. This could cause disturbances to the liquid on the robot. However, 0.1 m/s is not large enough to create a large jerk for spillage to occur. If a speed of 0.2, more considerations need to be made.

6 Conclusions

6.1 Summary

Previously, the robot was only capable of moving smoothing is a 1D motion. The algorithm used for smoothing was a simple computational approach designed by Wan et al. This was then implemented together with the Manhattan Distance algorithm to combine the x-axis and y-axis motion. Thus, a smooth velocity profile was obtained for the planar motion. An experimentation for the rhombus-shaped motion profile was generated, no spillage of water was observed. Therefore, the smooth planar velocity profile was valid.

To increase the versatility of the waiter robot, a ROS state machine design for the cocktail robot was created. The states transition in the following sequence: Start, Acceleration, Cruise, Dock, Deceleration. A random generator was embedded to determine a random direction of motion. The robot will only move when the random generator produces a direction that is obstacle-free. To allow obstacle avoidance with the use of the IR sensors, an estop function was integrated. If less than 0.3m was detected, estop would activate. For docking to occur, the robot has to be in the cruise state, and the IR must detect less than 0.8m for deceleration to occur. Thus, allowing the robot to come to a stop upon detecting a stationary obstacle. An experiment for the autonomous cocktail robot was carried out, the robot only moves when the obstacle-free direction was randomly generated, the estop and docking function works as programmed, and no spillage occurred. Therefore, the first step to an autonomous cocktail robot was accomplished.

6.2 Future Work

The autonomous waiter robot can be integrated with more features to increase its versatility to function. The following features are as follows:

1. Integration with machine vision to recognise different objects.
2. Human-Robot interaction in a restaurant setting.
3. Integration with systems to implement the robot in both cocktail and restaurant at the same time, where it can be new product.
4. Add more IOT features to integrate the robot into a smart restaurant, such as being able to take orders.

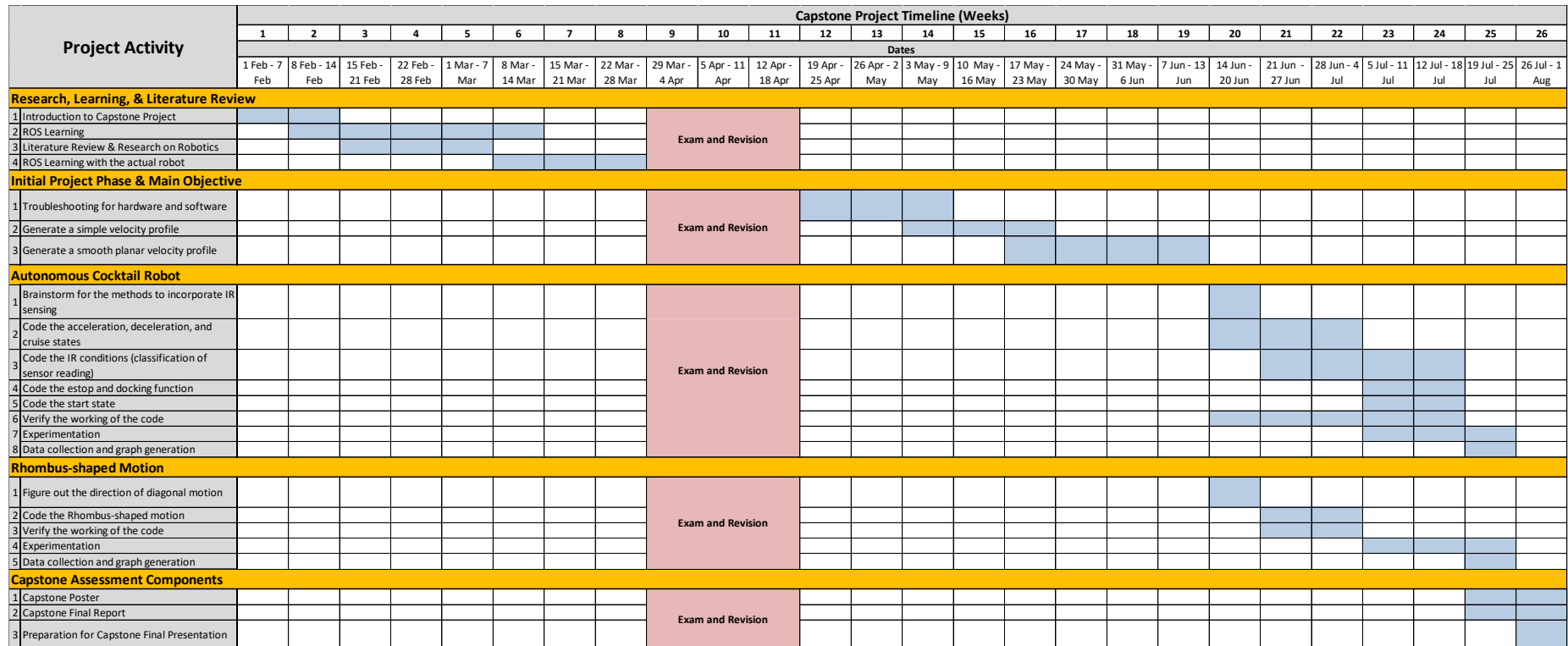
7 References

- [1] A. Y. S. Wan, Y. De Soong, E. Foo, W. L. E. Wong, and W. S. M. Lau, “Waiter Robots Conveying Drinks,” *Technologies*, vol. 8, no. 3, p. 44, 2020, doi: 10.3390/technologies8030044.
- [2] S. Brady, “Robot waiters serve drinks and take temperatures at this Dutch restaurant - Lonely Planet,” *Lonely Planet*, 2020. <https://www.lonelyplanet.com/articles/robot-waiters-netherlands> (accessed Mar. 02, 2021).
- [3] L. Wu, “The Rise Of Restaurant Robots Amidst Pandemic Measures,” *Forbes*, 2020. <https://www.forbes.com/sites/lesliewu/2020/09/27/the-rise-of-restaurant-robots-amidst-pandemic-measures/?sh=31c426c54ea3> (accessed Mar. 02, 2021).
- [4] K. Lim, “Meet robot waiters Le Le and Xiao Jin at hot pot restaurant Pin Xian Lou, Food News & Top Stories - The Straits Times,” *The Straits Times*, 2017. <https://www.straitstimes.com/lifestyle/food/adding-byte-to-your-appetite> (accessed Mar. 02, 2021).
- [5] “Beijing Haidilao hotpot: Good reasons to dine in on your China tour,” *ChinaTours.com*, 2021. <https://www.chinatours.com/beijing-haidilao-hotpot/> (accessed Mar. 02, 2021).
- [6] Chris Young, “Spot the Robot Spotted Serve Beers in Sevilla, Spain ,” *Interesting Engineering*, Nov. 02, 2020. <https://interestingengineering.com/spot-the-robot-spotted-serving-beers-in-sevilla-spain> (accessed Jul. 29, 2021).
- [7] KHOU 11, *Robots replacing waiters at restaurants, bars to help prevent spread of COVID-19 - YouTube*. Youtube, 2021.
- [8] A. Wan, E. Foo, Z. Lai, H. Chen, and W. Lau, “Waiter Bots for Casual Restaurants,” *Int. J. Robot. Eng.*, vol. 4, no. 1, 2019, doi: 10.35840/2631-5106/4118.
- [9] C. Guarino Lo Bianco, “Minimum-jerk velocity planning for mobile robot applications,” *IEEE Trans. Robot.*, vol. 29, no. 5, pp. 1317–1326, 2013, doi: 10.1109/TRO.2013.2262744.
- [10] C. Guarino Lo Bianco and M. Romano, “Bounded velocity planning for autonomous vehicles,” *2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS*, no. 1, pp. 685–690, 2005, doi: 10.1109/IROS.2005.1545604.
- [11] G. Lini, L. Consolini, and A. Piazzzi, “Minimum-time constrained velocity planning,” *2009 17th Mediterr. Conf. Control Autom.*, no. 6, pp. 748–753, 2009, doi: 10.1109/med.2009.5164633.
- [12] H. Barghi Jond, V. V. Nabyev, and A. Akbarimajd, “Planning of mobile robots under limited velocity and acceleration,” *2014 22nd Signal Process. Commun. Appl. Conf. SIU 2014 - Proc.*, no. Siu, pp. 1579–1582, 2014, doi: 10.1109/SIU.2014.6830545.
- [13] S. Lu, J. Zhao, L. Jiang, and H. Liu, “Solving the Time-Jerk Optimal Trajectory Planning Problem of a Robot Using Augmented Lagrange Constrained Particle Swarm Optimization,” *Math. Probl. Eng.*, vol. 2017, 2017, doi: 10.1155/2017/1921479.
- [14] Y. Fang, J. Hu, W. Liu, Q. Shao, J. Qi, and Y. Peng, “Smooth and time-optimal S-curve trajectory planning for automated robots and machines,” *Mech. Mach. Theory*, vol. 137, pp. 127–153, 2019, doi: 10.1016/j.mechmachtheory.2019.03.019.
- [15] M. Ben-Ari and F. Mondada, *Elements of Robotics NN ML in robotics. Fundamentals*. 2017.
- [16] F. Mura, N. Franceschini, L. De Neurobiologie, and C. J. Aiguiier, “Obstacle avoidance in a terrestrial mobile robot provided with a scanning retina,” pp. 0–5.
- [17] Z. Sun and J. Reif, “On energy-minimizing paths on terrains for a mobile robot,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 3, pp. 3782–3788, 2003, doi: 10.1109/robot.2003.1242177.

- [18] A. H. Ismail, H. R. Ramli, M. H. Ahmad, and M. H. Marhaban, "Vision-based system for line following mobile robot," *2009 IEEE Symp. Ind. Electron. Appl. ISIEA 2009 - Proc.*, vol. 2, no. Isiea, pp. 642–645, 2009, doi: 10.1109/ISIEA.2009.5356366.
- [19] D. Punetha, N. Kumar, and V. Mehta, "Development and Applications of Line Following Robot Based Health Care Management System," vol. 7, no. 27, pp. 14785–14792, 2015.
- [20] N. M. A. Ghani, F. Naim, and T. P. Yon, "Two wheels balancing robot with line following capability," *World Acad. Sci. Eng. Technol.*, vol. 79, pp. 634–638, 2011, doi: 10.5281/zenodo.1063160.
- [21] I. Gavrilut, V. Tiponut, A. Gacsadi, and L. Tepelea, "Obstacles avoidance method for an autonomous mobile robot using two IR sensors," *J. Electr. Electron. Eng.*, vol. 1, no. 1, pp. 194–197, 2008.
- [22] Y. Ando and S. Yuta, "Following a wall by an autonomous mobile robot with a sonar-ring," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 3, pp. 2599–2606, 1995, doi: 10.1109/robot.1995.525649.
- [23] G. Schillaci and V. V. Hafner, "Random movement strategies in self-exploration for a humanoid robot," *HRI 2011 - Proc. 6th ACM/IEEE Int. Conf. Human-Robot Interact.*, pp. 245–246, 2011, doi: 10.1145/1957656.1957753.
- [24] L. Palmieri, A. Rudenko, and K. O. Arras, "A fast random walk approach to find diverse paths for robot navigation," *IEEE Robot. Autom. Lett.*, vol. 2, no. 1, pp. 269–276, 2017, doi: 10.1109/LRA.2016.2602240.
- [25] D. Rachmawati and L. Gustin, "Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem," *J. Phys. Conf. Ser.*, vol. 1566, no. 1, 2020, doi: 10.1088/1742-6596/1566/1/012061.
- [26] Y. De Soong, "Capstone Project Report," *Capstone Proj. Rep. Wait. Robot Causal Restaur.*, no. June, 2020.
- [27] S. Karki and H. S. Ranjitkar, "Comparison of A*, Euclidean and Manhattan distance using Influence Map in Ms. Pac-Man," 2016.
- [28] "Installation and Operation Manual," no. April. TERABEE, pp. 1–26, 2010, doi: 294-00000-000-02-201502.

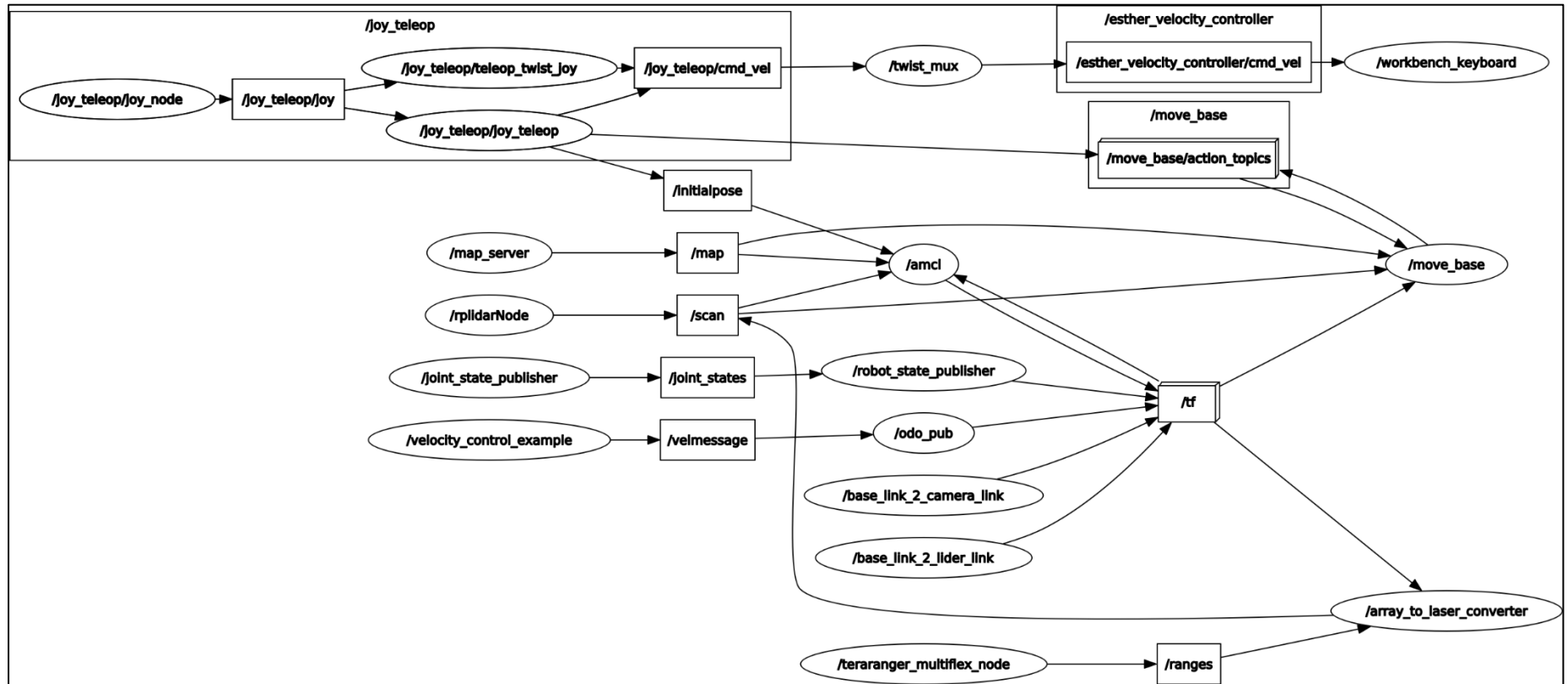
Appendix

Appendix A – Gantt Chart



Exam and Revision	
Completed	

Appendix B – ROS Graph



Appendix C – Navigation Launch Folder (XML file)

The backed-up catkin_ws can be downloaded from this link:

https://drive.google.com/file/d/1z5_fCGQBODY-noY8CntkmdGrOKe5bBI8/view?usp=sharing

It can be found on catkin_ws>src>nnaavvii>launch.

```
<?xml version="1.0"?>

<launch>

  <param name="/use_sim_time" value="false" />
  <param name="pub_map_odom_transform" value="true" />
  <param name="map_frame" value="map" />
  <param name="base_frame" value="base_link" />
  <param name="odom_frame" value="odom" />

  <include file="$(find my_dynamixel_workbench_tutorial)/launch/omniwheel_new.launch" />
  <!--<include file="$(find bringup)/launch/depthtolaser.launch" />-->
  <!--<node name="robot_state_publisher" pkg="robot_state_publisher"
type="state_publisher" />-->

  <!--<node pkg="tf" type="static_transform_publisher" name="map_2_odom"
args="0.0 0.0 0.0 0 0 0 /map /odom 100"/> -->

  <node pkg="tf" type="static_transform_publisher"
name="base_link_2_camera_link" args="0.16 -0.0125 0.2850 0 0 0 /base_link
/camera_link 100"/>
  <node pkg="tf" type="static_transform_publisher"
name="base_link_2_lider_link" args="0 0 0.0500 0 0 0 /camera_link /laser
100"/>

  <!-- <param name="pub_map_odom_transform" value="true" />-->

  <node pkg="odom_tf_package" type="odo_pub" name="odo_pub"
output="screen"/>
  <!--<arg name="odom_topic"
default="/dynamixel_workbench_velocity_conversion/odom" />-->

  <arg name="map_file" default="$(find nnaavvii)/map/nyp23.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg
map_file)" />

  <include file="$(find rplidar_ros)/launch/rplidar.launch" />
```

```

<!--include file="$ (find astra_launch)/launch/astra.launch" />-->
<!--<include file="$ (find nnaavvii)/launch/base_laser_filter.launch" />-->
  <include file="$ (find nnaavvii)/launch/aammccll.launch">
    <arg name="use_map_topic" value ="true"/>
  </include>

  <include file="$ (find nnaavvii)/launch/move_base.launch" />

<node pkg="teraranger_array" type="teraranger_multiflex"
name="teraranger_multiflex_node">
  <param name="portname" value="/dev/IR"/>/-->
  <!--remap from="ranges" to="ranges_raw" /-->
</node>

  <node pkg="teraranger_array_converter"
type="teraranger_array_converter.py" name="array_to_laser_converter">
  <param name="converter_mode" value="laser_scan"/>
  <param name="conversion_frame" value="base_hub"/>
  <param name="force_tf_refresh" value="true"/>
  <rosparam>
    sensor_mask: [true,true,true,true,true,true,true,true]
  </rosparam>
</node>

<!--
  <include file="$ (find navigation)/amcl/examples/aammccll.launch">
    <arg name="use_map_topic" value ="true"/>
  </include>

  <include file="$ (find navigation)/move_base/include/move_base.launch" />
-->

<!--

  <node pkg="amcl" type="amcl" name="amcl">
    <param name="use_map_topic" value="$ (arg
use_map_topic)"/>

    <param name="odom_model_type" value="omni"/>
    <param name="odom_alpha5" value="0.1"/>
    <param name="gui_publish_rate" value="1.0"/>
    <param name="laser_max_beams" value="100"/>
    <param name="min_particles" value="500"/>

```

```

    <param name="max_particles" value="50000"/>
    <param name="kld_err" value="0.05"/>
    <param name="kld_z" value="1.00"/>
    <param name="odom_alpha1" value="0.2"/>
    <param name="odom_alpha2" value="0.2"/>

    <param name="odom_alpha3" value="0.2"/>
    <param name="odom_alpha4" value="0.2"/>
    <param name="laser_z_hit" value="0.9"/>
    <param name="laser_z_short" value="0.05"/>
    <param name="laser_z_max" value="0.05"/>
    <param name="laser_z_rand" value="0.5"/>
    <param name="laser_sigma_hit" value="0.2"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_model_type" value="likelihood_field"/>

    <param name="laser_min_range" value="1"/>
    <param name="laser_max_range" value="5"/>
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d" value="0.2"/>
    <param name="update_min_a" value="0.5"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="base_frame_id" value="base_link"/>
    <param name="global_frame_id" value="map"/>
    <param name="resample_interval" value="1"/>
    <param name="transform_tolerance" value="0.5"/>
    <param name="recovery_alpha_slow" value="0.001"/>
    <param name="recovery_alpha_fast" value="0.1"/>

    <param name="initial_pose_x" value="0.0"/>
    <param name="initial_pose_y" value="0.0"/>
    <param name="initial_pose_a" value="0.0"/>
    <remap from="scan" to="/scan"/>

</node>
-->

<!--
  <node pkg="move_base" type="move_base" respawn="false" name="move_base"
  output="screen">

    <roscppparam file="$(find nnaavvii)/parafiles/costmap_common_params.yaml"
    command="load" ns="global_costmap" />

```

```

    <rosparam file="$(find nnaavvii)/parafiles/costmap_common_params.yaml"
command="load" ns="local_costmap" />
    <rosparam file="$(find nnaavvii)/parafiles/local_costmap_params.yaml"
command="load" />
    <rosparam file="$(find nnaavvii)/parafiles/global_costmap_params.yaml"
command="load" />
    <rosparam file="$(find
nnaavvii)/parafiles/base_local_planner_params.yaml" command="load" />

    <remap from="cmd_vel" to="/esther_velocity_controller/cmd_vel1"/>
    <remap from="odom" to="$(arg odom_topic)"/>

</node>
-->

```

```

</launch>

```


Appendix D – Rhombus State Machine

It can be found in Catkin_ws > src > executive_smach > smach > src > smach

[rhombus_final.py](#)

```
#!/usr/bin/env python
## Alis Notes
## Library Section

import rospy
import smach
import smach_ros
from smach import CBState
import sensor_msgs.point_cloud2

# From simple_navigation_goal
import roslib
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, Point, Quaternion, Twist,
PoseStamped, PoseWithCovarianceStamped
from nav_msgs.msg import Odometry
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from random import sample
from math import pow, sqrt
from std_msgs.msg import String
#Tera Converter
from sensor_msgs.msg import LaserScan, Range, PointCloud2
from sensor_msgs import point_cloud2
from std_msgs.msg import Header
from teraranger_array.msg import RangeArray, sensor_msgs
import math

## Global Declaration

## Var and const Assignment Section
Prox = Range ()
TwistNow = Twist()
cmd_vel = Twist()
j = 0.001
dv = 0.0
dvx = 0.0
dvy = 0.0
PoseNow = Pose()
count = 0

## Callback Section
def callback1(msg):
    Prox.range = msg.ranges[0].range
```

```

Prox.range = msg.ranges[1].range
Prox.range = msg.ranges[2].range
Prox.range = msg.ranges[3].range
Prox.range = msg.ranges[4].range
Prox.range = msg.ranges[5].range
Prox.range = msg.ranges[6].range
Prox.range = msg.ranges[7].range

def callback2(msg):
    Vel_1.linear.x= msg.linear.x
    Vel_1.linear.y= msg.linear.y
    Vel_1.angular.z= msg.angular.z

def callback3(amcl):
    PoseNow.position.x = amcl.pose.pose.position.x
    PoseNow.position.y = amcl.pose.pose.position.y
    PoseNow.position.z = amcl.pose.pose.position.z
    PoseNow.orientation = amcl.pose.pose.orientation

def callback4(msg):
    GoalPose.position.x = (msg.pose.position.x)
    GoalPose.position.y = (msg.pose.position.y)
    GoalPose.position.z = (msg.pose.position.z)
    #Goal.orientation.z = goal.pose.pose.orientation
    #rospy.loginfo(GoalPose.position.x)

def callback5(odom):
    TwistNow.linear.x = odom.twist.twist.linear.x
    TwistNow.linear.y = odom.twist.twist.linear.y
    TwistNow.angular.z = odom.twist.twist.angular.z

#State 1: Acceleration X+ - Adjust Command Velocity (Diagonal Only)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Slow
X', 'Vel Ok1', 'Too Slow X2', 'Vel Ok2'])
def accel_x(msg):
    global dvx, dvy, cmd_vel, count
    rospy.loginfo('acceleration x+')
    rospy.loginfo(count)
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    if count != 2 and count != 3 and count != 4:
        rospy.loginfo('X+Y+')
        if cmd_vel.linear.x <= 0.050:
            dvx = dvx + j
            rospy.loginfo(dvx)
            cmd_vel.linear.x = cmd_vel.linear.x + dvx
            vel_pub.publish(cmd_vel)

```

```

        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Too Slow X'
    if cmd_vel.linear.x > 0.05 and cmd_vel.linear.x <= 0.1:
        dvx = dvx - j
        rospy.loginfo(dvx)
        cmd_vel.linear.x = cmd_vel.linear.x + dvx
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Too Slow X'
    if cmd_vel.linear.x >= 0.1:
        dvx = 0.00
        cmd_vel.linear.x = cmd_vel.linear.x
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvx)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Vel Ok1'

if count == 2:
    rospy.loginfo('X+Y-')
    if cmd_vel.linear.x <= 0.050:
        dvx = dvx + j
        rospy.loginfo(dvx)
        cmd_vel.linear.x = cmd_vel.linear.x + dvx
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Too Slow X2'
    if cmd_vel.linear.x > 0.05 and cmd_vel.linear.x <= 0.1:
        dvx = dvx - j
        rospy.loginfo(dvx)
        cmd_vel.linear.x = cmd_vel.linear.x + dvx
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Too Slow X2'
    if cmd_vel.linear.x >= 0.1:
        dvx = 0.00
        cmd_vel.linear.x = cmd_vel.linear.x
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvx)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Vel Ok2'

```

#State 2: Acceleration X- - Adjust Command Velocity (Diagonal Only)

```

@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Slow
X3', 'Vel Ok3', 'Too Slow X4', 'Vel Ok4'])
def accel_xminus(msg):
    global dvx, dvy, cmd_vel, count
    rospy.loginfo('acceleration x-')
    rospy.loginfo(count)
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    if count == 3:
        rospy.loginfo('X-Y-')
        if cmd_vel.linear.x >= -0.050:
            dvx = dvx - j
            rospy.loginfo(dvx)
            cmd_vel.linear.x = cmd_vel.linear.x + dvx
            vel_pub.publish(cmd_vel)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.05)
            return 'Too Slow X3'
        if cmd_vel.linear.x < -0.05 and cmd_vel.linear.x >= -0.1:
            dvx = dvx + j
            rospy.loginfo(dvx)
            cmd_vel.linear.x = cmd_vel.linear.x + dvx
            vel_pub.publish(cmd_vel)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.05)
            return 'Too Slow X3'
        if cmd_vel.linear.x <= -0.1:
            dvx = 0.00
            cmd_vel.linear.x = cmd_vel.linear.x
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvx)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.05)
            return 'Vel Ok3'
    if count == 4:
        rospy.loginfo('X-Y+')
        if cmd_vel.linear.x >= -0.050:
            dvx = dvx - j
            rospy.loginfo(dvx)
            cmd_vel.linear.x = cmd_vel.linear.x + dvx
            vel_pub.publish(cmd_vel)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.05)
            return 'Too Slow X4'
        if cmd_vel.linear.x < -0.05 and cmd_vel.linear.x >= -0.1:
            dvx = dvx + j
            rospy.loginfo(dvx)
            cmd_vel.linear.x = cmd_vel.linear.x + dvx
            vel_pub.publish(cmd_vel)

```

```

        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Too Slow X4'
    if cmd_vel.linear.x <= -0.1:
        dvx = 0.00
        cmd_vel.linear.x = cmd_vel.linear.x
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvx)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Vel Ok4'

#State 3: Acceleration Y+ - Adjust Command Velocity (Diagonal Only)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Slow
Y1', 'Too Slow Y4', 'Vel Ok1', 'Vel Ok4'])
def accel_y(msg):
    global dvx, dvy, cmd_vel, count
    rospy.loginfo('acceleration y+')
    rospy.loginfo(count)
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    if count != 2 and count != 3 and count != 4:
        rospy.loginfo('X+Y+')
        if cmd_vel.linear.y <= 0.05:
            dvy = dvy + j
            rospy.loginfo(dvy)
            cmd_vel.linear.y = cmd_vel.linear.y + dvy
            vel_pub.publish(cmd_vel)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.05)
            return 'Too Slow Y1'
        if cmd_vel.linear.y > 0.05 and cmd_vel.linear.y <= 0.1:
            dvy = dvy - j
            rospy.loginfo(dvy)
            cmd_vel.linear.y = cmd_vel.linear.y + dvy
            vel_pub.publish(cmd_vel)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.05)
            return 'Too Slow Y1'
        if cmd_vel.linear.y >= 0.1 and cmd_vel.linear.x >= 0.1 or
cmd_vel.linear.y >= 0.1 and cmd_vel.linear.x <= -0.1:
            dvy = 0.00
            cmd_vel.linear.x = cmd_vel.linear.x
            cmd_vel.linear.y = cmd_vel.linear.y
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvy)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.05)
            return 'Vel Ok1'

```

```

if count == 4:
    rospy.loginfo('X-Y+')
    if cmd_vel.linear.y <= 0.05:
        dvy = dvy + j
        rospy.loginfo(dvy)
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Too Slow Y4'
    if cmd_vel.linear.y > 0.05 and cmd_vel.linear.y <= 0.1:
        dvy = dvy - j
        rospy.loginfo(dvy)
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Too Slow Y4'
    if cmd_vel.linear.y >= 0.1 and cmd_vel.linear.x >= 0.1 or
cmd_vel.linear.y >= 0.1 and cmd_vel.linear.x <= -0.1:
        dvy = 0.00
        cmd_vel.linear.x = cmd_vel.linear.x
        cmd_vel.linear.y = cmd_vel.linear.y
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Vel Ok4'

#State 4: Acceleration Y- - Adjust Command Velocity (Diagonal Only)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=[ 'Too Slow
Y2', 'Vel Ok2', 'Too Slow Y3', 'Vel Ok3'])
def accel_yminus(msg):
    global dvx, dvy, cmd_vel, count
    rospy.loginfo('acceleration y-')
    rospy.loginfo(count)
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    if count == 2:
        rospy.loginfo('X+Y-')
        if cmd_vel.linear.y >= -0.05:
            dvy = dvy - j
            rospy.loginfo(dvy)
            cmd_vel.linear.y = cmd_vel.linear.y + dvy
            vel_pub.publish(cmd_vel)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.05)
            return 'Too Slow Y2'

```

```

    if cmd_vel.linear.y < -0.05 and cmd_vel.linear.y >= -0.1:
        dvy = dvy + j
        rospy.loginfo(dvy)
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Too Slow Y2'

    if cmd_vel.linear.y <= -0.1 and cmd_vel.linear.x >= 0.1 or
cmd_vel.linear.y <= -0.1 and cmd_vel.linear.x <= -0.1:
        dvy = 0.00
        cmd_vel.linear.x = cmd_vel.linear.x
        cmd_vel.linear.y = cmd_vel.linear.y
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Vel Ok2'

if count == 3:
    rospy.loginfo('X-Y-')
    if cmd_vel.linear.y >= -0.05:
        dvy = dvy - j
        rospy.loginfo(dvy)
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Too Slow Y3'

    if cmd_vel.linear.y < -0.05 and cmd_vel.linear.y >= -0.1:
        dvy = dvy + j
        rospy.loginfo(dvy)
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Too Slow Y3'

    if cmd_vel.linear.y <= -0.1 and cmd_vel.linear.x >= 0.1 or
cmd_vel.linear.y <= -0.1 and cmd_vel.linear.x <= -0.1:
        dvy = 0.00
        cmd_vel.linear.x = cmd_vel.linear.x
        cmd_vel.linear.y = cmd_vel.linear.y
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.05)
        return 'Vel Ok3'

```

#State 5: Deceleration X+ - Adjust Command Velocity (Diagonal Only)

```

@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Fast
X', 'Too Fast X2', 'Vel Ok1', 'Vel Ok2'])
def decel_x(msg):
    global dvx, cmd_vel, count
    rospy.loginfo('deceleration X+')
    rospy.loginfo(count)
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    if count != 2 and count != 3 and count != 4:
        rospy.loginfo('X+Y+')
        if cmd_vel.linear.x >= 0.05 :
            dvx = dvx - j
            cmd_vel.linear.x = cmd_vel.linear.x + dvx
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvx)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Too Fast X'
        if cmd_vel.linear.x < 0.050 and cmd_vel.linear.x >= 0.0:
            dvx = dvx + j
            cmd_vel.linear.x = cmd_vel.linear.x + dvx
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvx)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Too Fast X'
        if cmd_vel.linear.x <= 0.0:
            dvx = 0.00
            cmd_vel.linear.x = cmd_vel.linear.x
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvx)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Vel Ok1'
    if count == 2:
        rospy.loginfo('X+Y-')
        if cmd_vel.linear.x >= 0.05 :
            dvx = dvx - j
            cmd_vel.linear.x = cmd_vel.linear.x + dvx
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvx)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Too Fast X2'
        if cmd_vel.linear.x < 0.050 and cmd_vel.linear.x >= 0.0:
            dvx = dvx + j
            cmd_vel.linear.x = cmd_vel.linear.x + dvx
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvx)

```



```

        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Too Fast X2'
    if cmd_vel.linear.x <= 0.0:
        dvx = 0.00
        cmd_vel.linear.x = cmd_vel.linear.x
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvx)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Vel Ok2'

#State 5: Deceleration X- - Adjust Command Velocity (Diagonal Only)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Fast
X3', 'Vel Ok3', 'Too Fast X4', 'Vel Ok4'])
def decel_xminus(msg):
    global dvx, cmd_vel, count
    rospy.loginfo('deceleration X-')
    rospy.loginfo(count)
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    if count == 3:
        rospy.loginfo('X-Y-')
        if cmd_vel.linear.x <= -0.05 :
            dvx = dvx + j
            cmd_vel.linear.x = cmd_vel.linear.x + dvx
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvx)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Too Fast X3'
        if cmd_vel.linear.x > -0.050 and cmd_vel.linear.x <= 0.0:
            dvx = dvx - j
            cmd_vel.linear.x = cmd_vel.linear.x + dvx
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvx)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Too Fast X3'
        if cmd_vel.linear.x >= 0.0:
            dvx = 0.00
            cmd_vel.linear.x = cmd_vel.linear.x
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvx)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Vel Ok3'
    if count == 4:
        rospy.loginfo('X-Y+')

```

```

    if cmd_vel.linear.x <= -0.05 :
        dvx = dvx + j
        cmd_vel.linear.x = cmd_vel.linear.x + dvx
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvx)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Too Fast X4'
    if cmd_vel.linear.x > -0.050 and cmd_vel.linear.x <= 0.0:
        dvx = dvx - j
        cmd_vel.linear.x = cmd_vel.linear.x + dvx
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvx)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Too Fast X4'
    if cmd_vel.linear.x >= 0.0:
        dvx = 0.00
        cmd_vel.linear.x = cmd_vel.linear.x
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvx)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Vel Ok4'

#State 6: Deceleration Y+ - Adjust Command Velocity (Diagonal Only)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Fast
Y1', 'Vel Ok1', 'Too Fast Y4', 'Vel Ok4'])
def decel_y(msg):
    global dvy, cmd_vel, count
    rospy.loginfo('deceleration Y+')
    rospy.loginfo(count)
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    if count != 2 and count != 3 and count != 4:
        rospy.loginfo('X+Y+')
        if cmd_vel.linear.y >= 0.05:
            dvy = dvy - j
            cmd_vel.linear.y = cmd_vel.linear.y + dvy
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvy)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Too Fast Y1'
        if cmd_vel.linear.y < 0.050 and cmd_vel.linear.y >= 0.0:
            dvy = dvy + j
            cmd_vel.linear.y = cmd_vel.linear.y + dvy
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvy)

```

```

        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Too Fast Y1'
    if cmd_vel.linear.y <= 0.0:
        dvy = 0.00
        count = 2
        cmd_vel.linear.x = 0.0
        cmd_vel.linear.y = 0.0
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Vel Ok1'
if count == 4:
    rospy.loginfo('X-Y+')
    if cmd_vel.linear.y >= 0.05:
        dvy = dvy - j
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Too Fast Y4'
    if cmd_vel.linear.y < 0.050 and cmd_vel.linear.y >= 0.0:
        dvy = dvy + j
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Too Fast Y4'
    if cmd_vel.linear.y <= 0.0:
        dvy = 0.00
        cmd_vel.linear.x = 0.0
        cmd_vel.linear.y = 0.0
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Vel Ok4'

#State 6: Deceleration Y- - Adjust Command Velocity (Diagonal Only)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Fast
Y2', 'Vel Ok2', 'Too Fast Y3', 'Vel Ok3'])
def decel_yminus(msg):
    global dvy, cmd_vel, count
    rospy.loginfo('deceleration Y-')
    rospy.loginfo(count)

```

```

    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    if count == 2:
        rospy.loginfo('X+Y-')
        if cmd_vel.linear.y <= -0.05:
            dvy = dvy + j
            cmd_vel.linear.y = cmd_vel.linear.y + dvy
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvy)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Too Fast Y2'
        if cmd_vel.linear.y > -0.050 and cmd_vel.linear.y <= 0.0:
            dvy = dvy - j
            cmd_vel.linear.y = cmd_vel.linear.y + dvy
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvy)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Too Fast Y2'
        if cmd_vel.linear.y >= 0.0:
            dvy = 0.0
            count = 3
            cmd_vel.linear.x = 0.0
            cmd_vel.linear.y = 0.0
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvy)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Vel Ok2'
    if count == 3:
        rospy.loginfo('X-Y-')
        if cmd_vel.linear.y <= -0.05:
            dvy = dvy + j
            cmd_vel.linear.y = cmd_vel.linear.y + dvy
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvy)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Too Fast Y3'
        if cmd_vel.linear.y > -0.050 and cmd_vel.linear.y <= 0.0:
            dvy = dvy - j
            cmd_vel.linear.y = cmd_vel.linear.y + dvy
            vel_pub.publish(cmd_vel)
            rospy.loginfo(dvy)
            rospy.loginfo(cmd_vel)
            rospy.sleep(0.5)
            return 'Too Fast Y3'
        if cmd_vel.linear.y >= 0.0:

```

```

        dvy = 0.0
        count = 4
        cmd_vel.linear.x = 0.0
        cmd_vel.linear.y = 0.0
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Vel Ok3'

#State Machine: Main Function - Handles States Transitions
if __name__=='__main__':
    rospy.init_node ('Velocity_StateMachine')
    sm=smach.StateMachine(outcomes=['Idle'])
    rospy.Subscriber("ranges",RangeArray, callback1)
    rospy.Subscriber("esther_velocity_controller/cmd_vel1", Twist,
callback2)
    rospy.Subscriber("amcl_pose", PoseWithCovarianceStamped,
callback3)
    rospy.Subscriber("move_base_simple/goal", PoseStamped, callback4)
    rospy.Subscriber("dynamixel_workbench_velocity_conversion/odom",
Odometry, callback5)

    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('Accelerate X+', CBState(accel_x),
        {'Too Slow X': 'Accelerate Y+', 'Vel Ok1': 'Accelerate
Y+', 'Too Slow X2': 'Accelerate Y-',
        'Vel Ok2': 'Accelerate Y-'})
        smach.StateMachine.add('Accelerate X-',
CBState(accel_xminus),
        {'Too Slow X3': 'Accelerate Y-', 'Vel Ok3': 'Accelerate Y-',
'Too Slow X4': 'Accelerate Y+',
        'Vel Ok4': 'Accelerate Y+'})
        smach.StateMachine.add('Accelerate Y+', CBState(accel_y),
        {'Too Slow Y1': 'Accelerate X+', 'Too Slow Y4': 'Accelerate
X-', 'Vel Ok1': 'Decelerate X+',
        'Vel Ok4': 'Decelerate X-'})
        smach.StateMachine.add('Accelerate Y-',
CBState(accel_yminus),
        {'Too Slow Y2': 'Accelerate X+', 'Vel Ok2': 'Decelerate X+',
'Too Slow Y3': 'Accelerate X-',
        'Vel Ok3': 'Decelerate X-'})

        smach.StateMachine.add('Decelerate X+', CBState(decel_x),
        {'Too Fast X': 'Decelerate Y+', 'Too Fast X2': 'Decelerate Y-
', 'Vel Ok1': 'Decelerate Y+',
        'Vel Ok2': 'Decelerate Y-'})

```

```

        smach.StateMachine.add('Decelerate X-',
CBState(decel_xminus),
        {'Too Fast X3':'Decelerate Y-', 'Vel Ok3':'Decelerate Y-
', 'Too Fast X4':'Decelerate Y+',
        'Vel Ok4':'Decelerate Y+'})
        smach.StateMachine.add('Decelerate Y+', CBState(decel_y),
        {'Too Fast Y1':'Decelerate X+', 'Vel Ok1':'Accelerate
X+', 'Too Fast Y4':'Decelerate X-',
        'Vel Ok4':'Idle'})
        smach.StateMachine.add('Decelerate Y-',
CBState(decel_yminus),
        {'Too Fast Y2':'Decelerate X+', 'Vel Ok2':'Accelerate X-
', 'Too Fast Y3':'Decelerate X-',
        'Vel Ok3':'Accelerate X-'})

# Execute SMACH Plan
outcome = sm.execute()

```

Appendix E – Autonomous Cocktail Robot State Machine

It can be found in Catkin_ws > src > executive_smach > smach > src > smach

[teraflex_final.py](#)

```
#!/usr/bin/env python
## Alis Notes
## Library Section

import rospy
import smach
import smach_ros
from smach import CBState
import sensor_msgs.point_cloud2
import sys
from random import randint

# From simple_navigation_goal
import roslib
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, Point, Quaternion, Twist,
PoseStamped, PoseWithCovarianceStamped
from nav_msgs.msg import Odometry
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from random import sample
from math import pow, sqrt
from std_msgs.msg import String
#Tera Converter
from sensor_msgs.msg import LaserScan, Range, PointCloud2
from sensor_msgs import point_cloud2
from std_msgs.msg import Header
from teraranger_array.msg import RangeArray, sensor_msgs
import math

## Global Declaration

## Var and const Assignment Section
Prox = Range()
TwistNow = Twist()
cmd_vel = Twist()
GoalPose = Pose()
PoseNow = Pose()
dvx = 0.0
dvy = 0.0
cond_straight = 0
cruise_cond = 0
diagy_cond = 0
cond_XpYp = 0
```

```

cond_XpYm = 0
cond_XmYp = 0
cond_XmYm = 0
ProxMax = 0.0
t = 0
X_Now = 0.0
Y_Now = 0.0
Dock_Rad = 0.0
i = 0.0
min_int = 0
sensor = 0
avg = 0.0
r1 = 0.0
r2 = 0.0
r3 = 0.0
n = 0
r = 0.0
GoalPose.position.x = -5.63897813308 #-5.63897813308 + 1meter
GoalPose.position.y = 1.31322831629 # 1.31322831629 + 1meter
j = 0.001
IRmax = 0.7
IRinf = sys.maxint
estop = 0.3

## Callback Section
def callback1(msg):
    global i, min_int, ProxMax, sensor, n, IRmax, IRinf
    for i in range(8):
        if msg.ranges[i].range == float('inf') or
msg.ranges[i].range == float('-inf'):
            msg.ranges[i].range = IRinf
        if n == 1:#X-Y+
            if msg.ranges[1].range == IRinf or msg.ranges[1].range > 0:
                ProxMax = msg.ranges[1].range
        if n == 2:#X+Y+
            if msg.ranges[2].range == IRinf or msg.ranges[2].range > 0:
                ProxMax = msg.ranges[2].range
        if n == 3:#forward
            if msg.ranges[3].range == IRinf or msg.ranges[3].range > 0
or msg.ranges[4].range == IRinf or msg.ranges[4].range > 0:
                ProxMax = msg.ranges[3].range
        if n == 4:#backward
            if msg.ranges[0].range == IRinf or msg.ranges[0].range > 0
or msg.ranges[7].range == IRinf or msg.ranges[7].range > 0:
                ProxMax = msg.ranges[0].range
        if n == 5:#X+Y-
            if msg.ranges[5].range == IRinf or msg.ranges[5].range > 0:
                ProxMax = msg.ranges[5].range
        if n == 6:#X-Y-

```



```

        if msg.ranges[6].range == IRinf or msg.ranges[6].range > 0:
            ProxMax = msg.ranges[6].range

    if n == 7:#left
        if msg.ranges[1].range > 0 and msg.ranges[2].range > 0:
            ProxMax = (msg.ranges[1].range +
msg.ranges[2].range)/2
        if n == 8:#right
            if msg.ranges[5].range > 0 and msg.ranges[6].range > 0:
                ProxMax = (msg.ranges[5].range +
msg.ranges[6].range)/2

def callback2(msg):
    cmd_vel.linear.x= msg.linear.x
    cmd_vel.linear.y= msg.linear.y
    cmd_vel.angular.z= msg.angular.z

def callback3(amcl):
    PoseNow.position.x = amcl.pose.pose.position.x
    PoseNow.position.y = amcl.pose.pose.position.y
    PoseNow.position.z = amcl.pose.pose.position.z
    #PoseNow.orientation = amcl.pose.pose.orientation

def callback4(msg):
    GoalPose.position.x = (msg.pose.position.x)
    GoalPose.position.y = (msg.pose.position.y)
    GoalPose.position.z = (msg.pose.position.z)
    #Goal.orientation.z = goal.pose.pose.orientation
    #rospy.loginfo(GoalPose.position.x)

def callback5(odom):
    TwistNow.linear.x = odom.twist.twist.linear.x
    TwistNow.linear.y = odom.twist.twist.linear.y
    TwistNow.angular.z = odom.twist.twist.angular.z

def average(a,t, b, c, d, x):
    if t < 3 :
        d = c
        c = b
        b = a
        return 0
    else:
        d = c
        c = b
        b = a
        return (b + c + d)/3

```

```

#State 1: Start Up
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Starting
Up', 'Go X+', 'Go X-', 'Go Y+', 'Go Y-'])
def start(msg):
    global dvx, dvy, cmd_vel, t, ProxMax, avg, r1, r2, r3, n
    rospy.loginfo('Start')
    rospy.loginfo(n)
    rospy.loginfo(ProxMax)
    t = t + 1
    rospy.loginfo(t)
    if t <= 50:
        if t == 1:
            n = randint(1, 8)
            return 'Starting Up'
    if t > 50 and t <= 53 :
        if t < 53:
            r3 = r2
            r2 = r1
            r1 = ProxMax
            avg = average(ProxMax, t, r1, r2, r3, 1)
            rospy.loginfo(avg)
            return 'Starting Up'
        if t == 53:
            rospy.loginfo(avg)
            if avg >= ProxMax:
                t = 0
                if n == 3 or n == 2 or n == 5:
                    return 'Go X+'
                if n == 4 or n == 1 or n == 6:
                    return 'Go X-'
                if n == 7:
                    return 'Go Y+'
                if n == 8:
                    return 'Go Y-'
            else:
                t = 0
                n = 3
                return 'Go X+'

#State 3: Cruise
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Go
Cruise', 'Go Dock', 'Estop Cruise'])
def cruise(msg):
    global cmd_vel, cruise_cond, GoalPose, PoseNow, X_Now, Y_Now,
cond_XpYp, cond_XpYm, cond_XmYp, cond_XmYm, cond_straight
    global Dock_Rad, avg, t, ProxMax, r1, r2, r3, estop, IRmax

```

```

    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    rospy.loginfo('Cruise')
    rospy.loginfo(n)
    rospy.loginfo(ProxMax)
    cond_XpYp = cmd_vel.linear.x >= 0.1 and cmd_vel.linear.y >= 0.1
    cond_XpYm = cmd_vel.linear.x >= 0.1 and cmd_vel.linear.y <= -0.1
    cond_XmYp = cmd_vel.linear.x <= -0.1 and cmd_vel.linear.y >= 0.1

    cond_XmYm = cmd_vel.linear.x <= -0.1 and cmd_vel.linear.y <= -0.1
    cond_straight = cmd_vel.linear.x >= 0.1 or cmd_vel.linear.x <= -0.1
or cmd_vel.linear.y >= 0.1 or cmd_vel.linear.y <= -0.1
    cruise_cond = cond_XpYp or cond_XpYm or cond_XmYp or cond_XmYm or
cond_straight
    X_Now = GoalPose.position.x - PoseNow.position.x
    Y_Now = GoalPose.position.y - PoseNow.position.y
    rospy.loginfo(X_Now)
    rospy.loginfo(Y_Now)
    Dock_Rad = abs(sqrt((X_Now)**2+(Y_Now)**2))
    t = t + 1
    avg = average(ProxMax,t,r1,r2,r3,0)
    if ProxMax <= estop:
        t = 0
        cmd_vel.linear.x = 0
        cmd_vel.linear.y = 0
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        return 'Estop Cruise'
    if ProxMax < IRmax or Dock_Rad < 0.2:
        t = 0
        return 'Go Dock'
    if cruise_cond and ProxMax > estop:
        cmd_vel.linear.x = cmd_vel.linear.x
        cmd_vel.linear.y = cmd_vel.linear.y
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.loginfo(X_Now)
        rospy.loginfo(Y_Now)
        rospy.sleep(0.5)
        return 'Go Cruise'

#State 4: Docking
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Go X+', 'Go
X-', 'Go Y+', 'Go Y-', 'Stay', 'Restart'])
def Dock(msg):
    global cmd_vel, X_Now, Y_Now, Dock_Rad, PoseNow, GoalPose, t, n,
avg, ProxMax, IRmax

```

```

    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    rospy.loginfo('Docked')
    t = t + 1
    X_Now = GoalPose.position.x - PoseNow.position.x
    Y_Now = GoalPose.position.y - PoseNow.position.y
    Dock_Rad = sqrt((X_Now)**2+(Y_Now)**2)
    rospy.loginfo(X_Now)
    rospy.loginfo(Y_Now)
    if avg < ProxMax or ProxMax < IRmax:
        if n == 3 or n == 2 or n == 5:
            rospy.loginfo(Dock_Rad)
            rospy.loginfo(X_Now)
            rospy.loginfo(Y_Now)
            return 'Go X+'
        if n == 4 or n == 1 or n == 6:
            rospy.loginfo(Dock_Rad)
            rospy.loginfo(X_Now)
            rospy.loginfo(Y_Now)
            return 'Go X-'
        if n == 7:
            rospy.loginfo(Dock_Rad)
            rospy.loginfo(X_Now)
            rospy.loginfo(Y_Now)
            return 'Go Y+'
        if n == 8:
            rospy.loginfo(Dock_Rad)
            rospy.loginfo(X_Now)
            rospy.loginfo(Y_Now)
            return 'Go Y-'
    if PoseNow.position.x==5.63897813308 and
PoseNow.position.x==0.0183932259212:
        return 'Stay'
    #if GoalPose.position.x==0.0137 and GoalPose.position.y==0.975
    elif avg < estop and t == 200:
        t = 0
        return 'Restart'

#State 5: Acceleration X+ - Adjust Command Velocity (+ve X)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Slow
X3+', 'Vel OkX3+', 'Too Slow X2+', 'Vel OkX2+', 'Too Slow X5+', 'Vel OkX5+',
'Estop dX+'])
def accelx_plus(msg):
    global dvx, dvy, cmd_vel, n, avg, ProxMax, t, r1, r2, r3, estop
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    rospy.loginfo('acceleration diag x+')

```

```

rospy.loginfo(n)
rospy.loginfo(ProxMax)
X_Now = GoalPose.position.x - PoseNow.position.x
Y_Now = GoalPose.position.y - PoseNow.position.y
rospy.loginfo(X_Now)
rospy.loginfo(Y_Now)
t = t + 1
avg = average(ProxMax,t,r1,r2,r3,0)
if ProxMax < estop:
    t = 0
    dvx = 0.0
    cmd_vel.linear.x = 0
    cmd_vel.linear.y = 0
    vel_pub.publish(cmd_vel)
    rospy.loginfo(cmd_vel)
    return 'Estop dX+'
if cmd_vel.linear.x <= 0.050:
    dvx = dvx + j
    rospy.loginfo(dvx)
    cmd_vel.linear.x = cmd_vel.linear.x + dvx
    vel_pub.publish(cmd_vel)
    rospy.loginfo(cmd_vel)
    rospy.sleep(0.5)
    if n == 3:
        return 'Too Slow X3+'
    if n == 2:
        return 'Too Slow X2+'
    if n == 5:
        return 'Too Slow X5+'
if cmd_vel.linear.x > 0.05 and cmd_vel.linear.x <= 0.1:
    dvx = dvx - j
    rospy.loginfo(dvx)
    cmd_vel.linear.x = cmd_vel.linear.x + dvx
    vel_pub.publish(cmd_vel)
    rospy.loginfo(cmd_vel)
    rospy.sleep(0.5)
    if n == 3:
        return 'Too Slow X3+'
    if n == 2:
        return 'Too Slow X2+'
    if n == 5:
        return 'Too Slow X5+'
if cmd_vel.linear.x >= 0.1:
    t = 0
    dvx = 0.00
    cmd_vel.linear.x = cmd_vel.linear.x
    vel_pub.publish(cmd_vel)
    rospy.loginfo(dvx)
    rospy.loginfo(cmd_vel)

```

```

        rospy.sleep(0.5)
    if n == 3:
        return 'Vel OkX3+'
    if n == 2:
        return 'Vel OkX2+'
    if n == 5:
        return 'Vel OkX5+'

#State 7: Acceleration X- - Adjust Command Velocity (-ve X)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Slow
X4-', 'Vel OkX4-', 'Too Slow X1-', 'Vel OkX1-', 'Too Slow X6-', 'Vel OkX6-
', 'Estop dX-'])
def accelx_minus(msg):
    global dvx, dvy, cmd_vel, n, avg, ProxMax, t, r1, r2, r3, estop
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    rospy.loginfo('acceleration diag x-')
    rospy.loginfo(n)
    rospy.loginfo(ProxMax)
    X_Now = GoalPose.position.x - PoseNow.position.x
    Y_Now = GoalPose.position.y - PoseNow.position.y
    rospy.loginfo(X_Now)
    rospy.loginfo(Y_Now)
    t = t + 1
    avg = average(ProxMax,t,r1,r2,r3,0)
    if ProxMax < estop:
        t = 0
        dvx = 0.0
        cmd_vel.linear.x = 0
        cmd_vel.linear.y = 0
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        return 'Estop dX-'
    if cmd_vel.linear.x >= -0.050:
        rospy.loginfo('Accel X-')
        dvx = dvx - j
        rospy.loginfo(dvx)
        cmd_vel.linear.x = cmd_vel.linear.x + dvx
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        if n == 4:
            return 'Too Slow X4-'
        if n == 1:
            return 'Too Slow X1-'
        if n == 6:
            return 'Too Slow X6-'
    if cmd_vel.linear.x < -0.05 and cmd_vel.linear.x >= -0.1:

```

```

        dvx = dvx + j
        rospy.loginfo(dvx)
        cmd_vel.linear.x = cmd_vel.linear.x + dvx
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        if n == 4:
            return 'Too Slow X4-'
        if n == 1:
            return 'Too Slow X1-'
        if n == 6:
            return 'Too Slow X6-'
    if cmd_vel.linear.x <= -0.1:
        t = 0
        dvx = 0.00
        cmd_vel.linear.x = cmd_vel.linear.x
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvx)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        if n == 4:
            return 'Vel OkX4-'
        if n == 1:
            return 'Vel OkX1-'
        if n == 6:
            return 'Vel OkX6-'

#State 9: Acceleration Y+ - Adjust Command Velocity (Diagonal Only) (+ve Y)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=[ 'Too Slow
Y7+', 'Too Slow Y1+', 'Too Slow Y2+', 'Vel OkY+', 'Estop dY+'])
def accely_plus(msg):
    global dvx, dvy, cmd_vel, n, diagy_cond, avg, ProxMax, t, r1, r2,
r3, estop
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    diagy_cond = cmd_vel.linear.y >= 0.1 or cmd_vel.linear.y >= 0.1 and
cmd_vel.linear.x >= -0.1 or cmd_vel.linear.y >= 0.1 and cmd_vel.linear.x >=
0.1

    rospy.loginfo('acceleration diag y+')
    rospy.loginfo(n)
    rospy.loginfo(ProxMax)
    X_Now = GoalPose.position.x - PoseNow.position.x
    Y_Now = GoalPose.position.y - PoseNow.position.y
    rospy.loginfo(X_Now)
    rospy.loginfo(Y_Now)
    t = t + 1
    avg = average(ProxMax,t,r1,r2,r3,0)

```

```

if ProxMax < estop:
    t = 0
    dvy = 0.0
    cmd_vel.linear.x = 0
    cmd_vel.linear.y = 0
    vel_pub.publish(cmd_vel)
    rospy.loginfo(cmd_vel)
    return 'Estop dY+'
if cmd_vel.linear.y <= 0.05:
    dvy = dvy + j
    rospy.loginfo(dvy)
    cmd_vel.linear.y = cmd_vel.linear.y + dvy
    vel_pub.publish(cmd_vel)
    rospy.loginfo(cmd_vel)
    rospy.sleep(0.5)
    if n == 7:
        return 'Too Slow Y7+'
    if n == 1:
        return 'Too Slow Y1+'
    if n == 2:
        return 'Too Slow Y2+'
if cmd_vel.linear.y > 0.05 and cmd_vel.linear.y <= 0.1:
    dvy = dvy - j
    rospy.loginfo(dvy)
    cmd_vel.linear.y = cmd_vel.linear.y + dvy
    vel_pub.publish(cmd_vel)
    rospy.loginfo(cmd_vel)
    rospy.sleep(0.5)
    if n == 7:
        return 'Too Slow Y7+'
    if n == 1:
        return 'Too Slow Y1+'
    if n == 2:
        return 'Too Slow Y2+'
if diagy_cond:
    t = 0
    dvy = 0.00
    cmd_vel.linear.x = cmd_vel.linear.x
    cmd_vel.linear.y = cmd_vel.linear.y
    vel_pub.publish(cmd_vel)
    rospy.loginfo(dvy)
    rospy.loginfo(cmd_vel)
    rospy.sleep(0.5)
    return 'Vel OkY+'

```

#State 11: Acceleration Y- - Adjust Command Velocity (-ve Y)


```

@smach.cb_interface(input_keys=[], output_keys=[], outcomes=[ 'Too Slow
Y8-', 'Too Slow Y5-', 'Too Slow Y6-', 'Vel OkY-', 'Estop dY-'])
def accely_minus(msg):
    global dvx, dvy, cmd_vel, n, diagy_cond, avg, ProxMax, t, r1, r2,
r3, estop
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    diagy_cond = cmd_vel.linear.y <= -0.1 or cmd_vel.linear.y <= -0.1
and cmd_vel.linear.x <= -0.1 or cmd_vel.linear.y <= -0.1 and
cmd_vel.linear.x >= 0.1
    rospy.loginfo('acceleration diag y-')
    rospy.loginfo(n)
    rospy.loginfo(ProxMax)
    X_Now = GoalPose.position.x - PoseNow.position.x
    Y_Now = GoalPose.position.y - PoseNow.position.y
    rospy.loginfo(X_Now)
    rospy.loginfo(Y_Now)
    t = t + 1
    avg = average(ProxMax,t,r1,r2,r3,0)
    if ProxMax < estop:
        t = 0
        dvy = 0.0
        cmd_vel.linear.x = 0
        cmd_vel.linear.y = 0
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        return 'Estop dY-'
    if cmd_vel.linear.y >= -0.05 :
        dvy = dvy - j
        rospy.loginfo(dvy)
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        if n == 8:
            return 'Too Slow Y8-'
        if n == 5:
            return 'Too Slow Y5-'
        if n == 6:
            return 'Too Slow Y6-'
    if cmd_vel.linear.y < -0.05 and cmd_vel.linear.y >= -0.1:
        dvy = dvy + j
        rospy.loginfo(dvy)
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        if n == 8:
            return 'Too Slow Y8-'

```

```

        if n == 5:
            return 'Too Slow Y5-'
        if n == 6:
            return 'Too Slow Y6-'
    if diagy_cond:
        t = 0
        dvy = 0.00
        cmd_vel.linear.x = cmd_vel.linear.x
        cmd_vel.linear.y = cmd_vel.linear.y
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Vel OkY-'

#State 13: Deceleration X+ - Adjust Command Velocity (Diagonal Only) (+ve
X)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=[ 'Too Fast
X3+', 'Vel OkX3+', 'Too Fast X2+', 'Vel OkX2+', 'Too Fast X5+', 'Vel OkX5+',
'Estop dX+' ])
def decelx_plus(msg):
    global dvx, cmd_vel, n, avg, ProxMax, t, r1, r2, r3, estop
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    rospy.loginfo('deceleration diag X+')
    rospy.loginfo(n)
    rospy.loginfo(ProxMax)
    X_Now = GoalPose.position.x - PoseNow.position.x
    Y_Now = GoalPose.position.y - PoseNow.position.y
    rospy.loginfo(X_Now)
    rospy.loginfo(Y_Now)
    t = t + 1
    avg = average(ProxMax,t,r1,r2,r3,0)
    if ProxMax < estop:
        t = 0
        dvx = 0.0
        cmd_vel.linear.x = 0
        cmd_vel.linear.y = 0
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        return 'Estop dX+'
    if cmd_vel.linear.x >= 0.05 :
        dvx = dvx - j
        cmd_vel.linear.x = cmd_vel.linear.x + dvx
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvx)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)

```

```

        if n == 3:
            return 'Too Fast X3+'
        if n == 2:
            return 'Too Fast X2+'
        if n == 5:
            return 'Too Fast X5+'
    if cmd_vel.linear.x < 0.050 and cmd_vel.linear.x >= 0.0:
        dvx = dvx + j
        cmd_vel.linear.x = cmd_vel.linear.x + dvx
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvx)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        if n == 3:
            return 'Too Fast X3+'
        if n == 2:
            return 'Too Fast X2+'
        if n == 5:
            return 'Too Fast X5+'
    if cmd_vel.linear.x <= 0.0:
        t = 0
        dvx = 0.00
        cmd_vel.linear.x = cmd_vel.linear.x
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvx)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        if n == 3:
            return 'Vel OkX3+'
        if n == 2:
            return 'Vel OkX2+'
        if n == 5:
            return 'Vel OkX5+'

#State 15: Deceleration X- - Adjust Command Velocity (Diagonal Only) (-ve
X)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Fast
X4-', 'Vel OkX4-', 'Too Fast X1-', 'Vel OkX1-', 'Too Fast X6-', 'Vel OkX6-',
'Estop dX-'])
def decelx_minus(msg):
    global dvx, cmd_vel, n, avg, ProxMax, t, r1, r2, r3, estop
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    rospy.loginfo('deceleration diag X-')
    rospy.loginfo(n)
    rospy.loginfo(ProxMax)
    X_Now = GoalPose.position.x - PoseNow.position.x
    Y_Now = GoalPose.position.y - PoseNow.position.y

```

```

rospy.loginfo(X_Now)
rospy.loginfo(Y_Now)
t = t + 1
avg = average(ProxMax,t,r1,r2,r3,0)
if ProxMax < estop:
    t = 0
    dvx = 0.0
    cmd_vel.linear.x = 0
    cmd_vel.linear.y = 0
    vel_pub.publish(cmd_vel)
    rospy.loginfo(cmd_vel)
    return 'Estop dX-'
if cmd_vel.linear.x <= -0.05 :
    dvx = dvx + j
    cmd_vel.linear.x = cmd_vel.linear.x + dvx
    vel_pub.publish(cmd_vel)
    rospy.loginfo(dvx)
    rospy.loginfo(cmd_vel)
    rospy.sleep(0.5)
    if n == 4:
        return 'Too Fast X4-'
    if n == 1:
        return 'Too Fast X1-'
    if n == 6:
        return 'Too Fast X6-'
if cmd_vel.linear.x > -0.050 and cmd_vel.linear.x <= 0.0:
    dvx = dvx - j
    cmd_vel.linear.x = cmd_vel.linear.x + dvx
    vel_pub.publish(cmd_vel)
    rospy.loginfo(dvx)
    rospy.loginfo(cmd_vel)
    rospy.sleep(0.5)
    if n == 4:
        return 'Too Fast X4-'
    if n == 1:
        return 'Too Fast X1-'
    if n == 6:
        return 'Too Fast X6-'
if cmd_vel.linear.x >= 0.0:
    t = 0
    dvx = 0.00
    cmd_vel.linear.x = cmd_vel.linear.x
    vel_pub.publish(cmd_vel)
    rospy.loginfo(dvx)
    rospy.loginfo(cmd_vel)
    rospy.sleep(0.5)
    if n == 4:
        return 'Vel OkX4-'
    if n == 1:

```

```

        return 'Vel OkX1-'
    if n == 6:
        return 'Vel OkX6-'

#State 17: Deceleration Y+ - Adjust Command Velocity (Diagonal Only) (+ve
Y)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Fast
Y7+', 'Too Fast Y1+', 'Too Fast Y2+', 'Vel OkY+', 'Estop dY+'])
def decely_plus(msg):
    global dvy, cmd_vel, n, diagy_cond, avg, ProxMax, t, r1, r2, r3,
estop
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    diagy_cond = cmd_vel.linear.y <= 0.0 or cmd_vel.linear.y <= 0.0 and
cmd_vel.linear.x <= 0.0 or cmd_vel.linear.y <= 0.0 and cmd_vel.linear.x >=
0.0

    rospy.loginfo('deceleration diag Y+')
    rospy.loginfo(n)
    rospy.loginfo(ProxMax)
    X_Now = GoalPose.position.x - PoseNow.position.x
    Y_Now = GoalPose.position.y - PoseNow.position.y
    rospy.loginfo(X_Now)
    rospy.loginfo(Y_Now)
    t = t + 1
    avg = average(ProxMax,t,r1,r2,r3,0)
    if ProxMax < estop:
        t = 0
        dvy = 0.0
        cmd_vel.linear.x = 0
        cmd_vel.linear.y = 0
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        return 'Estop dY+'
    if cmd_vel.linear.y >= 0.05:
        dvy = dvy - j
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        if n == 7:
            return 'Too Fast Y7+'
        if n == 1:
            return 'Too Fast Y1+'
        if n == 2:
            return 'Too Fast Y2+'
    if cmd_vel.linear.y < 0.050 and cmd_vel.linear.y >= 0.0:

```

```

        dvy = dvy + j
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        if n == 7:
            return 'Too Fast Y7+'
        if n == 1:
            return 'Too Fast Y1+'
        if n == 2:
            return 'Too Fast Y2+'
    if diagy_cond:
        t = 0
        dvy = 0.00
        cmd_vel.linear.x = 0.0
        cmd_vel.linear.y = 0.0
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Vel OkY+'

#State 19: Deceleration Y- - Adjust Command Velocity (Diagonal Only) (-ve
Y)
@smach.cb_interface(input_keys=[], output_keys=[], outcomes=['Too Fast
Y8-', 'Too Fast Y5-', 'Too Fast Y6-', 'Vel OkY-', 'Estop dY-'])
def decely_minus(msg):
    global dvy, cmd_vel, n, diagy_cond, avg, ProxMax, t, r1, r2, r3,
estop
    vel_pub = rospy.Publisher('esther_velocity_controller/cmd_vel',
Twist, queue_size=1)
    rospy.loginfo('deceleration diag Y-')
    rospy.loginfo(n)
    rospy.loginfo(ProxMax)
    X_Now = GoalPose.position.x - PoseNow.position.x
    Y_Now = GoalPose.position.y - PoseNow.position.y
    rospy.loginfo(X_Now)
    rospy.loginfo(Y_Now)
    diagy_cond = cmd_vel.linear.y >= 0.0 or cmd_vel.linear.y >= 0.0 and
cmd_vel.linear.x >= 0.0 or cmd_vel.linear.y >= 0.0 and cmd_vel.linear.x <=
0.0

    t = t + 1
    avg = average(ProxMax,t,r1,r2,r3,0)
    bag(cmd_vel, PoseNow, n, ProxMax)
    if ProxMax < estop:
        t = 0

```

```

        dvy = 0.0
        cmd_vel.linear.x = 0
        cmd_vel.linear.y = 0
        vel_pub.publish(cmd_vel)
        rospy.loginfo(cmd_vel)
        return 'Estop dY-'
    if cmd_vel.linear.y <= -0.05:
        dvy = dvy + j
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        if n == 8:
            return 'Too Fast Y8-'
        if n == 5:
            return 'Too Fast Y5-'
        if n == 6:
            return 'Too Fast Y6-'
    if cmd_vel.linear.y > -0.050 and cmd_vel.linear.y <= 0.0:
        dvy = dvy - j
        cmd_vel.linear.y = cmd_vel.linear.y + dvy
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        if n == 8:
            return 'Too Fast Y8-'
        if n == 5:
            return 'Too Fast Y5-'
        if n == 6:
            return 'Too Fast Y6-'
    if diag_cond:
        t = 0
        dvy = 0.00
        cmd_vel.linear.x = 0.0
        cmd_vel.linear.y = 0.0
        vel_pub.publish(cmd_vel)
        rospy.loginfo(dvy)
        rospy.loginfo(cmd_vel)
        rospy.sleep(0.5)
        return 'Vel OkY-'

```

#State Machine: Main Function - Handles States Transitions

```

if __name__ == '__main__':
    rospy.init_node ('Velocity_StateMachine')
    sm=smach.StateMachine(outcomes=['Idle'])
    rospy.Subscriber("ranges",RangeArray, callback1)

```

```

    rospy.Subscriber("esther_velocity_controller/cmd_vel1", Twist,
callback2)
    rospy.Subscriber("amcl_pose", PoseWithCovarianceStamped,
callback3)
    rospy.Subscriber("move_base_simple/goal", PoseStamped, callback4)
    rospy.Subscriber("dynamixel_workbench_velocity_conversion/odom",
Odometry, callback5)

    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('START', CBState(start),
            {'Starting Up': 'START', 'Go X+': 'Accelerate X+', 'Go X-': 'Accelerate X-', 'Go Y+': 'Accelerate Y+', 'Go Y-': 'Accelerate Y-'})
        smach.StateMachine.add('Cruise', CBState(cruise),
            {'Go Cruise': 'Cruise', 'Go Dock': 'Dock', 'Estop Cruise': 'START'})
        smach.StateMachine.add('Dock', CBState(Dock),
            {'Go X+': 'Decelerate X+', 'Go X-': 'Decelerate X-', 'Go Y+': 'Decelerate Y+', 'Go Y-': 'Decelerate Y-', 'Stay': 'Dock', 'Restart': 'START'})

        smach.StateMachine.add('Accelerate X+', CBState(accelx_plus),
            {'Too Slow X3+': 'Accelerate X+', 'Vel OkX3+': 'Cruise', 'Too Slow X2+': 'Accelerate Y+', 'Vel OkX2+': 'Accelerate Y+', 'Too Slow X5+': 'Accelerate Y-', 'Vel OkX5+': 'Accelerate Y-', 'Estop dX+': 'START'})
        smach.StateMachine.add('Accelerate X-', CBState(accelx_minus),
            {'Too Slow X4-': 'Accelerate X-', 'Vel OkX4-': 'Cruise', 'Too Slow X1-': 'Accelerate Y+', 'Vel OkX1-': 'Accelerate Y+', 'Too Slow X6-': 'Accelerate Y-', 'Vel OkX6-': 'Accelerate Y-', 'Estop dX-': 'START'})
        smach.StateMachine.add('Accelerate Y+', CBState(accel_y_plus),
            {'Too Slow Y7+': 'Accelerate Y+', 'Too Slow Y1+': 'Accelerate X-', 'Too Slow Y2+': 'Accelerate X+', 'Vel OkY+': 'Cruise', 'Estop dY+': 'START'})
        smach.StateMachine.add('Accelerate Y-', CBState(accel_y_minus),
            {'Too Slow Y8-': 'Accelerate Y-', 'Too Slow Y5-': 'Accelerate X+', 'Too Slow Y6-': 'Accelerate X-', 'Vel OkY-': 'Cruise', 'Estop dY-': 'START'})

```



```

        smach.StateMachine.add('Decelerate X+',
CBState(decelx_plus),
        {'Too Fast X3+': 'Decelerate X+', 'Vel OkX3+': 'START', 'Too
Fast X2+': 'Decelerate Y+',
        'Vel OkX2+': 'Decelerate Y+', 'Too Fast X5+': 'Decelerate Y-
', 'Vel OkX5+': 'Decelerate Y-',
        'Estop dX+': 'START'})
        smach.StateMachine.add('Decelerate X-',
CBState(decelx_minus),
        {'Too Fast X4-': 'Decelerate X-', 'Vel OkX4-': 'START', 'Too
Fast X1-': 'Decelerate Y+',
        'Vel OkX1-': 'Decelerate Y+', 'Too Fast X6-': 'Decelerate Y-
', 'Vel OkX6-': 'Decelerate Y-',
        'Estop dX-': 'START'})
        smach.StateMachine.add('Decelerate Y+',
CBState(decely_plus),
        {'Too Fast Y7+': 'Decelerate Y+', 'Too Fast Y1+': 'Decelerate
X-', 'Too Fast Y2+': 'Decelerate X+',
        'Vel OkY+': 'Idle', 'Estop dY+': 'START'})
        smach.StateMachine.add('Decelerate Y-',
CBState(decely_minus),
        {'Too Fast Y8-': 'Decelerate Y-', 'Too Fast Y5-': 'Decelerate
X+', 'Too Fast Y6-': 'Decelerate X-',
        'Vel OkY-': 'START', 'Estop dY-': 'START'})

# Execute SMACH Plan
outcome = sm.execute()

```

Appendix F – Work from Home Risk Assessment

Inventory of Work Activities				
Reference Number: (please refer to S&H RA Repository for next running number)			Division	MDME
Title	Waiter			
Ref	Location	Process	Work Activity	Remarks
1	Alis' Home	Robot Code Development	Programming of Robot	
2			Simulation of written code	
3		Robot Maintenance	Rewire or beautify the existing electrical circuitry	
4		Robot's Battery Maintenance	Charging of Robot's Battery	
5			Battery Storage	
6		Robot Mobility Testing	Testing for detouring	
7			Testing for docking	
8			Testing robot's response time	
9		Speed Control Testing	Testing robot's braking distance	
10			Testing robot's load-carrying stability	
11			Testing robot with varying velocity profiles	
12		Robot Testing	Lifting of Robot	

*add more rows if necessary.

** Examples: Contents will be automatically deleted when new content is typed into the rows.

RISK ASSESSMENT														
<u>Reference Number</u>				RA Leader:	Lin Mei Ling @Alis Lin			Approved by:	Dr Michael Lau					
Title:	Waiter! Campaign Please!			RA Team:				Signature:						
Division:	MDME	Location:	SIT@NYP SR6C; NYP R407					Designation:						
Last Review Date:		Next Review Date:						Date						
	Hazard Identification			Risk Evaluation			Risk Control							
Ref	<u>Activity</u>	<u>Hazard</u>	<u>Possible injury / ill-health</u>	<u>Existing risk controls</u>	<u>S</u>	<u>L</u>	<u>RPN</u>	<u>Additional controls</u>	<u>S</u>	<u>L</u>	<u>RPN</u>	<u>Implementation Person</u>	<u>Due date</u>	<u>Remarks</u>
1) Robot Code Development														
1	Programming of Robot	Sitting in front of computer screen for long hours	Neck Aches	Take a short break every 1-2 hours to stretch muscles	2	3	6	N.A.	-	-	-	N.A.	-	-
2	Simulation of written code	Sitting in front of computer screen for long hours	Neck Aches	Take a short break every 1-2 hours to stretch muscles	2	3	6	N.A.	-	-	-	N.A.	-	-
2) Robot Maintenance														
3	Rewire or beautify the existing electrical circuitry	Exposed Wires	Short Circuit	Use heat shrinking tubes or wire cutters as corrective measures	3	3	9	N.A.	-	-	-	N.A.	-	-
3) Robot's Battery Maintenance														
4	Charging of Robot's battery	Overloading	Fire	Charge battery in a Li-Po Guard Bag	4	1	4	Keep a look out on the battery's charge	4	1	4	Alis Lin	-	-
5	Battery Storage	Fire	Burns	Keep the battery in the Li-Po Guard Bag when not in use. Store battery in an area that does not have direct sunlight.	4	1	4	Check for bloating every 3 days. If it is bloated, do not charge	4	1	4	Alis Lin	-	-
4) Robot Mobility Testing														


Ref	Hazard Identification			Risk Evaluation			Risk Control							
	<u>Activity</u>	<u>Hazard</u>	<u>Possible injury / ill-health</u>	<u>Existing risk controls</u>	<u>S</u>	<u>L</u>	<u>RPN</u>	<u>Additional controls</u>	<u>S</u>	<u>L</u>	<u>RPN</u>	<u>Implementation Person</u>	<u>Due date</u>	<u>Remarks</u>
6	Testing for detouring	Obstructed Vision	Collision	Monitor robot closely during operation. Keep a lookout for obstacles.	2	3	6	Look for assistance to look out for blind spots and unexpected robot behaviours	2	1	2	Alis Lin	-	-
7	Testing for docking	Obstructed Vision	Collision	Monitor robot closely during operation. Keep a lookout for obstacles.	2	3	6	Look for assistance to look out for blind spots and unexpected robot behaviours	2	1	2	Alis Lin	-	-
8	Testing robot's response time	Obstructed Vision	Collision	Monitor robot closely during operation. Keep a lookout for obstacles.	2	3	6	Look for assistance to look out for blind spots and unexpected robot behaviours	2	1	2	Alis Lin	-	-
5) Speed Control Testing														
9	Testing robot's braking distance	Obstructed Vision	Collision	Monitor robot closely during operation. Keep a lookout for obstacles.	2	3	6	Look for assistance to look out for blind spots and unexpected robot behaviours	2	1	2	Alis Lin	-	-
10	Testing robot's load-carrying stability	Obstructed Vision	Collision	Monitor robot closely during operation. Keep a lookout for obstacles.	2	3	6	If test for spill is not required, robot carries empty containers. Only room temperature drinks should be conveyed during tests.	2	1	2	Alis Lin	-	-
11	Testing robot with varying velocity profiles	Obstructed Vision	Collision	Monitor robot closely during operation. Keep a lookout for obstacles.	2	3	6	If area is cluttered, run at a slow speed 0.1 m/s. If no obstacles or human are nearby, the speed can be increased up to 0.2 m/s.	2	1	2	Alis Lin	-	-
6) Robot Testing														
12	Lifting of robot for trolley mounting	Repetitive Motion	Back Strains	Exercise proper lifting posture. Ask for help if required.	3	3	9	N.A.	-	-	-	Alis Lin	-	-

Likelihood Severity	Rare (1)	Remote (2)	Occasional (3)	Frequent (4)	Almost Certain (5)
Catastrophic (5)	5 (M)	10 (M)	15 (H)	20 (H)	25 (H)
Major (4)	4 (M)	8 (M)	12 (M)	16 (H)	20 (H)
Moderate (3)	3 (L)	6 (M)	9 (M)	12 (M)	15 (H)
Minor (2)	2 (L)	4 (M)	6 (M)	8 (M)	10 (M)
Negligible (1)	1 (L)	2 (L)	3 (L)	4 (M)	(M)

Level	Severity	Description
1	Negligible	Not likely to cause injury or ill-health.
2	Minor	Injury or ill-health requiring first-aid only (includes minor cuts and bruises, irritation, ill-health with temporary discomfort).
3	Moderate	Injury or ill-health requiring medical treatment (includes lacerations, burns, sprains, minor fractures, dermatitis, and work-related upper limb disorders).
4	Major	Serious injuries or life-threatening occupational diseases (Includes amputations, major fractures, multiple injuries, occupational cancers, acute poisoning, <u>disabilities</u> and deafness).
5	Catastrophic	Death, fatal <u>diseases</u> or multiple major injuries.

Level	Likelihood	Description
1	Rare	Not expected to occur but still possible.
2	Remote	Not likely to occur under normal circumstances.
3	Occasional	Possible or known to occur.
4	Frequent	Common occurrence.
5	Almost certain	Continual or repeating experience.

Risk score	Acceptability of risk	Recommended actions
Low 1-3	Acceptable	No additional risk control measures may be needed. Frequent review and monitoring of hazards are required to ensure that the risk level assigned is accurate and does not increase over time.
Medium 4-12	Tolerable	A careful evaluation of the hazards should be carried out to ensure that the risk level is reduced to as low as reasonably practicable (ALARP) within a defined <u>time period</u> . Interim risk control measures, such as administrative controls, may be implemented while long term measures are being established. Management attention is required.
High 15-25	Not acceptable	High Risk level must be reduced to at least Medium Risk before work commences. There should not be any interim risk control measures and risk control measures should not be overly dependent on personal protective equipment. If practicable, the hazard should be eliminated before work commences. Management review is required before work commences.


	DOCUMENT TITLE Risk Assessment Form		Page xlvii of 82
	EFFECTIVE DATE 20190904	PROGRAMME TITLE Risk Management Programme	VERSION 3.1

Appendix G – Laboratory Risk Assessment


Inventory of Work Activities				
Reference Number: (please refer to S&H RA Repository for next running number)			Division	MDME
Title	Waiter			
Ref	Location	Process	Work Activity	Remarks
1	NYP R407	Robot Code Development	Programming of Robot	
			Simulation of written code	
2		Robot Maintenance	Rewire or beautify the existing electrical circuitry	
3		Charging of Robot	Charging of Robot's Battery	
4		Robot Mobility Testing	Testing for detouring	
5			Testing for docking	
6			Testing robot's response time	
7		Speed Control Testing	Testing robot's braking distance	
8			Testing robot's load-carrying stability	
9			Testing robot with varying velocity profiles	

*add more rows if necessary.


** Examples: Contents will be automatically deleted when new content is typed into the rows.

	DOCUMENT TITLE Risk Assessment Form		Page xlviii of 82
	EFFECTIVE DATE 20190904	PROGRAMME TITLE Risk Management Programme	VERSION 3.1

RISK ASSESSMENT														
Reference Number					RA Leader:	Lin Mei Ling @Alis Lin			Approved by:	Dr Michael Lau				
Title:	Waiter! Campaign Please!				RA Team:				Signature:					
Division:	MDME	Location:	SIT@NYP SR6C; NYP R407	Designation:										
Last Review Date:		Next Review Date:		Date										
	Hazard Identification				Risk Evaluation				Risk Control					
Ref	Activity	Hazard	Possible injury / ill-health	Existing risk controls	S	L	RPN	Additional controls	S	L	RPN	Implementation Person	Due date	Remarks
1) Robot Code Development														
1	Programming of Robot	Sitting in front of computer screen for long hours	Neck Aches	Take a short break every 1-2 hours to stretch muscles	2	3	6	N.A.	-	-	-	N.A.	-	-
2	Simulation of written code	Sitting in front of computer screen for long hours	Neck Aches	Take a short break every 1-2 hours to stretch muscles	2	3	6	N.A.	-	-	-	N.A.	-	-
2) Robot Maintenance														
3	Rewire or beautify the existing electrical circuitry	Exposed Wires	Short Circuit	Use heat shrinking tubes or wire cutters as corrective measures	3	3	9	N.A.	-	-	-	N.A.	-	-
3) Charging of Robot														
4	Charging of Robot’s battery	Battery with high rating	Overloading	Charge battery in a Li-Po Guard Bag	4	1	4	N.A.	-	-	-	N.A.	-	-
4) Robot Mobility Testing														

	DOCUMENT TITLE Risk Assessment Form		Page xlix of 82
	EFFECTIVE DATE 20190904	PROGRAMME TITLE Risk Management Programme	VERSION 3.1

Ref	Hazard Identification			Risk Evaluation			Risk Control							Remarks
	Activity	Hazard	Possible injury / ill-health	Existing risk controls	S	L	RPN	Additional controls	S	L	RPN	Implementation Person	Due date	
5	Testing for detouring	Obstructed Vision	Collision	Monitor robot closely during operation, keep a look out for passers-by	2	3	6	Look for assistance to look out for blind spots and unexpected robot behaviours	2	1	2	Alis Lin	-	-
6	Testing for docking	Obstructed Vision	Collision	Monitor robot closely during operation, keep a look out for passers-by	2	3	6	Look for assistance to look out for blind spots and unexpected robot behaviours	2	1	2	Alis Lin	-	-
7	Testing robot's response time	Obstructed Vision	Collision	Monitor robot closely during operation, keep a look out for passers-by	2	3	6	Look for assistance to look out for blind spots and unexpected robot behaviours	2	1	2	Alis Lin	-	-
5) Speed Control Testing														
8	Testing robot's braking distance	Obstructed Vision	Collision	Monitor robot closely during operation, keep a look out for passers-by	2	3	6	Look for assistance to look out for blind spots and unexpected robot behaviours	2	1	2	Alis Lin	-	-
9	Testing robot's load-carrying stability	Obstructed Vision	Collision	Monitor robot closely during operation, keep a look out for passers-by	2	3	6	Look for assistance to look out for blind spots and unexpected robot behaviours	2	1	2	Alis Lin	-	-
10	Testing robot with varying velocity profiles	Obstructed Vision	Collision	Monitor robot closely during operation, keep a look out for passers-by	2	3	6	Look for assistance to look out for blind spots and unexpected robot behaviours	2	1	2	Alis Lin	-	-

	DOCUMENT TITLE Risk Assessment Form		Page I of 82
	EFFECTIVE DATE 20190904	PROGRAMME TITLE Risk Management Programme	VERSION 3.1

Level	Severity	Description
1	Negligible	Not likely to cause injury or ill-health.
2	Minor	Injury or ill-health requiring first-aid only (includes minor cuts and bruises, irritation, ill-health with temporary discomfort).
3	Moderate	Injury or ill-health requiring medical treatment (includes lacerations, burns, sprains, minor fractures, dermatitis, and work-related upper limb disorders).
4	Major	Serious injuries or life-threatening occupational diseases (Includes amputations, major fractures, multiple injuries, occupational cancers, acute poisoning, disabilities and deafness).
5	Catastrophic	Death, fatal diseases or multiple major injuries.

Level	Likelihood	Description
1	Rare	Not expected to occur but still possible.
2	Remote	Not likely to occur under normal circumstances.
3	Occasional	Possible or known to occur.
4	Frequent	Common occurrence.
5	Almost certain	Continual or repeating experience.

Risk score	Acceptability of risk	Recommended actions
Low 1-3	Acceptable	No additional risk control measures may be needed. Frequent review and monitoring of hazards are required to ensure that the risk level assigned is accurate and does not increase over time.
Medium 4-12	Tolerable	A careful evaluation of the hazards should be carried out to ensure that the risk level is reduced to as low as reasonably practicable (ALARP) within a defined time period. Interim risk control measures, such as administrative controls, may be implemented while long term measures are being established. Management attention is required.
High 15-25	Not acceptable	High Risk level must be reduced to at least Medium Risk before work commences. There should not be any interim risk control measures and risk control measures should not be overly dependent on personal protective equipment. If practicable, the hazard should be eliminated before work commences. Management review is required before work commences.