DISSERTATION

AUTONOMOUS ROBOT CONTROL: INTEGRATED CONTROL STRATEGIES FOR A MOBILE ROBOTIC ARM

Submitted by

Katrina Weinmann

Department of Mechanical Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2025

Doctoral Committee:

Advisor: Steve Simske

Thomas Chen
Susan James
Jianguo Zhao

ABSTRACT

AUTONOMOUS ROBOT CONTROL: INTEGRATED CONTROL STRATEGIES FOR A MOBILE ROBOTIC ARM

Autonomous robots open up a wide range of potential applications for robotic systems beyond the controlled manufacturing environments where they were originally used. These applications can include agriculture, space exploration, search and rescue, fire-fighting, performing tasks in hazardous environments, personal care robots, and much more. Some of these applications have the potential to replace humans in dangerous environments or improve quality of life for elderly or disabled individuals, thus providing great positive societal impacts. However, the technologies needed for robots to operate safely and autonomously in unstructured environments, and especially when interacting with humans, are still being developed. Designing and controlling autonomous robotic systems is a very challenging problem, with some of the major objectives including efficient autonomous navigation in both known and unknown environments; real-time, dynamic obstacle avoidance; real-time and energy-efficient trajectory generation; and safe operation with and/or in close proximity to humans. While all of these topics have been researched in the field of robotics, existing solutions still have limitations which encourage further developments improving on existing autonomous robotic capabilities. Furthermore, each application and configuration of autonomous robotic systems has a unique set of requirements. In this dissertation, the platform of a mobile robotic arm was chosen for its wide range of potential applications achieved from the combined navigation and manipulation abilities of such a robot configuration. Within the scope of autonomous mobile robot arm control, the following topics were identified and chosen for

research: (1) indoor target localization; (2) efficient navigation in partially known environments; (3) integrated control (i.e. coordinated base and arm motion) of a mobile robot arm, including both navigation and trajectory generation and tracking. For each topic, a novel or improved methodology was developed, all relevant to a wide range of autonomous robot deployments. The contributions of this dissertation are as follows: (1) Bluetooth-based homing controller for indoor target localization achieving a mean target localization accuracy of 0.12m or less with various levels of simulated sensor noise; (2) modified artificial potential field-based method for efficient navigation in a partially-known environment and for integrated control of a mobile robot arm improving autonomous navigation success rate and efficiency over existing APF-based methods in environments of varying levels of complexity; (3) real-time many objective optimization-based approach trajectory generation method for integrated motion of a mobile robot arm to reach a desired end-effector configuration, demonstrating a 100% success rate in achieving the desired configuration and reaching the configuration in under 30s in 77% of trials; (4) offline trajectory generation for mobile robot arm end-effector trajectory tracking using a sampling-based combinatorial optimization method to generate integrated motion trajectories (coordinated mobile base and robotic arm motion) achieving over 99% success rate in high accuracy (<5mm position tracking error and <0.1 radian orientation tracking error) end-effector trajectory tracking tested on 500 sample trajectories; and (5) integrated controller design for differential drive mobile robot arm trajectory tracking consisting of a fuzzy logic-based differential drive robot (DDR) controller reducing irregular trajectory tracking errors by 2.4X to 6.8X over existing DDR controller designs, and integrated robotic arm-facilitated DDR base tracking error compensation reducing mean maximum end-effector tracking errors by 18%.

ACKNOWLEDGEMENTS

I would like to thank my dissertation advisor, Dr. Steve Simske, for all of his guidance and support during the research presented in this thesis. I would also like to thank my committee members, Dr. Thomas Chen, Dr. Susan James, and Dr. Jianguo Zhao, for the opportunity to present my dissertation and their input on my work, and the CSU Department of Mechanical Engineering and Systems Engineering Department. Finally, I would like to thank the State of Colorado (CO Senate Bill 18-086) and U.S. Army DEVCOM Ground Vehicle Systems Center and the Air Force Research Laboratory (agreement number FA8650-20-2-5700) for their funding and support of this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 – Introduction

Advancements in computing power, control strategies, and machine learning and artificial intelligence have all greatly spurned the field of robotics forward. While the earliest robots were used mainly for automating tasks in controlled environments, such as manufacturing assembly lines, today's robots are no longer limited to operating under such stringent conditions. Development of technologies improving the autonomous capabilities of robots opens a wide range of new potential applications where use of robotics could prove very beneficial. These applications range from space exploration to search and rescue to fire fighting and beyond.

There are many different configurations of robotic platforms depending on the intended application. In this dissertation, we focus on the control of a mobile robotic arm. Such a robotic platform is relevant to a wide range of applications due to its combined navigation and manipulation abilities. Existing research has mostly focused on the mobile robot arm control problem as two separate control problems – navigation of the mobile base to the task location, followed by manipulation task execution by the robotic arm. We explore the alternate solution of integrated control of the mobile robotic arm. That is, coordinated motion of the mobile base and robotic arm together to execute a given manipulation task. Integrated mobile robot arm control offers several advantages over the traditional, separated control approach, such as more energy efficient trajectories and a virtually-infinite working field in the XY plane.

When considering integrated control of a mobile robotic arm, all aspects of task execution were considered. The task execution can be divided into three main steps: (i) identification and localization of target; (ii) navigation to identified task location; (iii) execution of manipulation task trajectory. Each of these steps was considered within the context of the overall task objective and

integrated control scheme, and methods and control strategies were proposed for each of the aforementioned steps. Specifically, the topics covered will be as follows: (i) development of a Bluetooth-enabled homing algorithm to identify the location of the target, (ii) modifications to the traditional artificial potential field (APF) method of path planning to encourage efficient and reliable navigation to the target location, (iii) development of an APF-based method of determining the optimal mobile robot base location and navigation path for approach to an integrated mobile robot/robotic arm task, (iv) real-time trajectory generation to a target location for an integrated control mobile robotic arm task using many-objective optimization, (v) trajectory generation for end-effector trajectory tracking using a holistic combinatorial optimization approach, and (vi) design of an integrated controller architecture to track generated trajectories using a differential-drive mobile robot arm. Chapters 2 through 7 of this dissertation each cover one of the proposed research objectives, with relevant literature presented in the appropriate chapter. Finally, a summary of the research findings, limitations, and directions for future research is covered in Chapter 8.

Chapter 2 – Bluetooth Homing Controller for Indoor Target Localization[1]

2.1 - Introduction

Some potential applications for autonomous robots include fire rescue, in-home emergency medical response, and object retrieval in hazardous environments. The characteristic that all of these applications have in common is that the task is not well-defined *a priori* because the target location for the robot may be unknown at initiation of the task. In the case of fire rescue, the robot must navigate through a burning building to a person trapped inside, but the location of this person within the building may be unknown. For emergency medical response, the robot must respond to the occurrence of an adverse medical event, and to do so must first navigate to the patient who could be located anywhere within their home. The precise locations of objects in hazardous environments may not be known, and thus the robot must determine the location of the desired object to be able to perform the retrieval task. These examples of applications for autonomous robots illustrate that the task cannot be fully defined *a priori*, meaning part of the task for the robot is to determine the location of the target in real-time to navigate to the target and complete the task. Thus, in this chapter we focus on the challenge of autonomous target localization in indoor environments for robotic tasks where the location of the target is unknown.

In outdoor environments, GPS is commonly used for target localization. However, GPS is unable to provide sufficient accuracy in indoor environments to be suitable for most applications. The solution to the indoor target localization task must satisfy four mandatory characteristics: (i)

[1] Publication info: K. Weinmann and S. Simske, "Design of Bluetooth 5.1 Angle of Arrival Homing Controller for Autonomous Mobile Robot," *Robotics*, vol. 12, no.4, 2023.

operates in scenarios where there is not a clear line of sight to the target object; (ii) provides a means of object-specific ID for the target; (iii) does not require existing infrastructure in the environment to operate; (iv) integrates with existing autonomous navigation technologies to create a comprehensive solution providing localization of and navigation to a target. In order to achieve the criteria of object-specific ID, the solution will require that the target object is equipped with some kind of beacon for identification. The objective is that this beacon can be identified and localized using only a sensor mounted on the mobile robot, without the need for any additional sensors or environmental information. Additional characteristics of the solution that are desirable include low energy consumption of both the beacon and the sensor, target localization accuracy of under 0.2m, and a widely-available technology for ease of deployment.

2.2 - Background

Sensor technologies that are commonly used in autonomous navigation tasks include LIDAR and ultrasonic (US) sensors. These are used for environment mapping and obstacle avoidance as they measure the distance to obstacles along a measurement direction. However, neither of these technologies are suitable for the proposed challenge because they do not satisfy all necessary criteria to complete the task. Both LIDAR and US are line-of-sight sensors, meaning that they only provide distance readings to the first obstacle that is encountered along the direction of sensing. These technologies provide no means of object-specific identification, in addition to requiring a line-of-sight to the object, thus eliminating these technologies as possible solutions. Cameras can be used with computer vision (CV) for object identification and localization [1], but this solution also requires a line-of-sight to the object. Additionally, while CV may be able to identify the type of object that is the target, it may be unable to distinguish between multiples of the same type of object when a specific one is needed.

RFID satisfies the object-specific ID criteria, can be used when there is not a clear line-of-sight to the target, and has been demonstrated as a solution for object tracking. Liu et al. used particle filtering with passive ultra-high frequency (UHF) RFID tags for dynamic object tracking with a mobile robot. This approach was successful for dynamic object tracking except in cases where the RFID tag was affixed to a metal object and in cases of cluttered environments where line-of-sight to the target was quickly lost [2]. Duan et al. used a fusion of RFID and CV to achieve 10.33mm mean location tracking accuracy [3], but this hybrid system requires line-of-sight for the CV component. These examples illustrate that RFID is a feasible method for target following when the target is within close range, but it has some deficiencies for target localization applications. Passive UHF RFID tags have a limited range (usually <12m), making them unsuitable for applications where the target is far away, as the signal will be lost. Additionally, RFID is sensitive to environmental factors, such as metal objects, water, and even walls, which can greatly affect transmitting range. As such, the limitations of RFID tracking are prohibitive for use in a robust target localization system.

Another related area of research is the development of indoor positioning systems. These systems are commonly used for asset tracking and even localization of robots within an environment, and employ technologies such as Ultra-Wideband (UWB) and Bluetooth. The general structure of these systems is to use static reference sensor locations, or anchors, within an environment, along with beacons fixed to the objects with desired tracking. The location of the tracked object can be estimated using readings from the anchors.

UWB technology is becoming more popular as a method for indoor robot localization because it provides relatively high accuracy and is easily scalable for multi-robot systems [4]. A system for robot self-localization using UWB technology was presented by Ledergerber et al. [5]. This system

used 3 static anchors and timed difference of arrival to track the location of a quadrocoptor within an RMS error of 0.3m. Cao et al. proposed a system using a single UWB anchor combined with a 9-DOF inertial measurement unit (IMU) for robot tracking, achieving an RMS tracking error of 0.48m [6]. For multi-robot localization applications, the most common framework uses fixed anchors which are used to localize all robots. Research is also being done using UWB for cooperative localization and to coordinate cooperative actions for multi-robot systems [7].

For Bluetooth-based systems, much research was done before the introduction of Bluetooth 5.1 and relied on received signal strength indication (RSSI) to determine the proximity between two Bluetooth units [8][9][10]. While these systems were successful at localization to a certain extent, it was found that the variability in RSSI limited the accuracy of such systems to the range of 1-2 meters, which was above the acceptable error threshold in many applications.

The introduction of Bluetooth 5.1 direction-finding greatly expanded the capabilities of using Bluetooth for indoor localization. Bluetooth angle of arrival is an implementation of direction-finding using a transmitter, which is a beacon capable of emitting constant tone extension (CTE), and a receiver, which contains an antenna array to receive the CTE signal. The angle of arrival of the signal between the beacon and the antenna array can be calculated using IQ sampling (which contains phase information about the signal) delivered via the CTE. The phase difference between the signal at the different antennas on the array and the known geometry of the antenna array allows calculation of the angle of arrival [11]. There is much ongoing research aimed at improving the accuracy of the angle estimation via various techniques, such a combination of nonlinear curve fitting, Kalman filtering, and Gaussian filtering [12], and machine learning frameworks [13].

Frameworks using Bluetooth AoA for indoor localization have exhibited varying results. Methods relying solely on AoA measurements have achieved mean accuracies of <1m [14] [15],

but suffer from variable accuracy depending on beacon and antenna locations, in addition to line-of-sight and other interference reducing measurement accuracy. Additionally, a wide variance has been reported in the accuracy of AoA measurements, from a mean error of 1.9 degrees over the full 360 degree range in [11], to a mean absolute error (MAE) of 29 degrees over a 180 degree range in [16]. To address variations in AoA measurement accuracy, [16] proposes a hybrid method of localization incorporating RSSI measurements to improve accuracy.

To the knowledge of the authors, little research has been done examining indoor target localization in the context of autonomous mobile robotic task execution. The existing research provides a framework for designing a system using Bluetooth or similar technologies for target localization. The proposed systems, however, rely on multiple reference sensors placed throughout the environment. Such a framework is not feasible for the applications presented in this paper, where modifications to the environment may not be possible. Thus, the system must function using only the robot-equipped sensors. Vasilopoulos et al. used an RF-range sensor to perform homing to an unknown target location for a legged robot, but details and performance of the target localization portion of the experiment were not provided [17].

2.3 - Methods

This chapter proposes a method by which Bluetooth 5.1 Angle of Arrival (AoA) technology is used to create a homing controller performing real-time target localization. The controller uses Bluetooth signal between a mobile robot and target (of initially unknown location) to determine the precise target location as the autonomous robot navigates through the environment. In the proposed solution, a transmitting Bluetooth beacon is attached to the target object and the mobile robot is equipped with a receiving Bluetooth antenna array (Figure 2.1). Bluetooth technology was selected for this application for several reasons: (i) ubiquity of Bluetooth in smart devices makes

the signal readily available, (ii) device-specific ID ensures navigation to correct target, (iii) low-energy signal with Bluetooth Low Energy (BLE), and (iv) the recently-introduced Bluetooth 5.1 direction finding specification expands capabilities of Bluetooth for indoor localization.

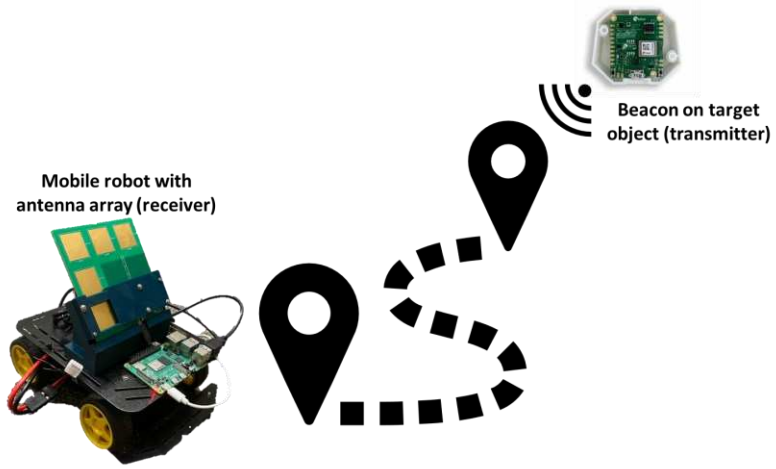


Figure 2.1. Configuration of proposed setup of Bluetooth homing controller.

The proposed algorithm implements a hybrid approach, using both AoA and RSSI for localization, such as that presented in [13], to improve accuracy and robustness in determining the target object location. Additionally, the mobile implementation of the AoA antenna on a robot serves to mitigate many of the problems preventing high-accuracy localization in stationary indoor localization applications and eliminates the need for existing sensing infrastructure in the environment. The mobile deployment allows for AoA measurements to be taken at a range of locations, resulting in a much larger dataset to use for convergence on the correct position. Additionally, much research has shown reduced accuracy of AoA near and outside of +90° and -90°. Reduced accuracy is also achieved when there is not a clear line-of-sight, creating a location-dependence on the accuracy of AoA positioning calculations. In the proposed implementation, as the robot navigates toward the target object and around obstacles, an originally obstructed line-of-

sight will become clear, and the robot will turn towards the object as the position converges, resulting in more accurate AoA measurements, and thus encouraging convergence to an accurate target position.

2.3.1 - Homing Controller Design

The primary objective of the homing controller is target localization. This target location can then be used to directly calculate the desired velocity of the mobile robot in order to navigate to the object. Alternatively, it can be fed into another control algorithm to determine the desired motion of the robot, such as a path planning algorithm. The idea behind the proposed homing algorithm is that Bluetooth AoA technology can be used to locate an object with a single beacon and antenna array by collecting data at multiple locations as the robot moves. In the proposed setup, the goal object is equipped with a Bluetooth CTE-enabled beacon and the antenna array is mounted on the mobile robot. Thus, as the robot moves, the changing angle of arrival between the robot and the object can be used to locate the object. There are two methods by which the location of the object can be calculated from the AoA data – a vector calculation or a parallax calculation. Both methods of calculation and goal position updating are described, and a hybrid implementation of the two methods is used in the controller to improve robustness and aid convergence to an accurate goal position for the desired object, henceforth referred to as the target.

2.3.1.1 - <u>Vector Position Calculation</u>

Vector position calculation of the target location is performed by creating a vector with both angle and magnitude parameters between the mobile robot and the target. The direction of the vector can be calculated from the robot orientation and the AoA reading, but the distance between the robot and the target cannot be explicitly obtained from the Bluetooth signal. However, the Bluetooth signal contains an RSSI parameter, which is correlated with the distance between the

beacon and the antenna array. An RSSI to distance calibration can be performed to estimate the distance based on the RSSI reading, as shown in Figure 2.2. Using this calibration, the estimated location of the target (x_g, y_g) can be calculated with (2.1) and (2.2), where θ_r is the heading of the robot, θ_s is the sensor angle, or AoA reading, (x_r, y_r) is the current location of the robot, and d is the estimated distance from the RSSI-distance calibration.

$$x_g = x_r + dcos(\theta_r + \theta_s) \tag{2.1}$$

$$y_g = y_r + dsin(\theta_r + \theta_s) \tag{2.2}$$

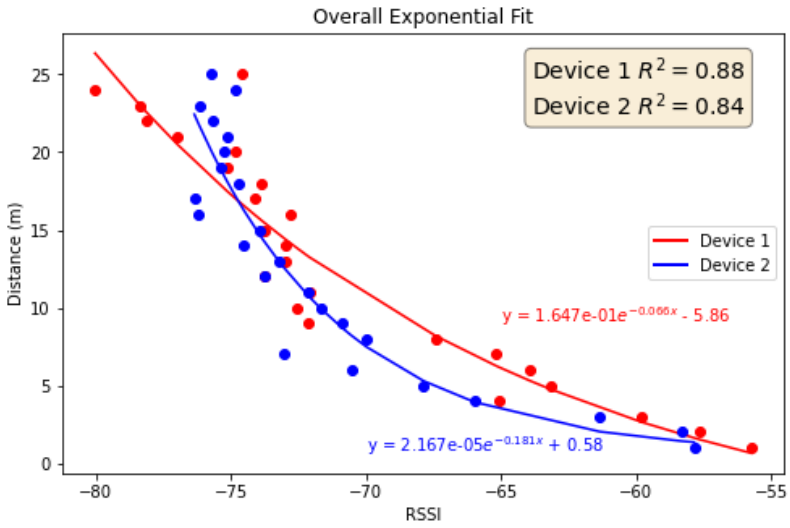


Figure 2.2. Example RSSI-Distance calibration for 2 beacons.

To reduce the impact of sensor noise on the stability of the calculated goal position, at each update event the reported vector goal position is calculated as a time-weighted average of the last 20 calculated vector goal positions. The time-weighting is implemented as an exponential decay function, given by (2.3), where w_k is the weight of the k^{th} position in the vector goal position buffer, k is the position index in the vector goal position buffer, with k=1 corresponding to the most recent calculated goal position, and τ is a time-weighting parameter.

$$w_k = e^{-\frac{k-1}{\tau}} \tag{2.3}$$

The updated vector position $(x_{g,v}, y_{g,v})$ is then calculated using (2.4) and (2.5).

$$x_{g,v} = \frac{w_1 x_{g,1} + w_2 x_{g,2} + \cdots + w_{20} x_{g,20}}{w_1 + w_2 + \cdots + w_{20}} \tag{2.4}$$

$$y_{g,v} = \frac{w_1 y_{g,1} + w_2 y_{g,2} + \cdots + w_{20} y_{g,20}}{w_1 + w_2 + \cdots + w_{20}} \tag{2.5}$$

2.3.1.2 - Parallax Position Calculation

The parallax calculation of the target location utilizes the intersection of vectors obtained from angle readings taken at different locations to determine the target location. This method eliminates the need to know the distance between the robot and the target. However, the goal position updating in this method is more complicated because, in the absence of perfect sensor readings and robot odometry data, there is not a single intersection point for more than two vectors. The initial goal position can be calculated simply using the first two angle readings using (2.6) and (2.7)

$$x_g = x_{r1} + d_1 \cos(\theta_{r1} + \theta_{s1}) \tag{2.6}$$

$$y_g = y_{r1} + d_1 \sin(\theta_{r1} + \theta_{s1}) \tag{2.7}$$

where,

$$d_1 = -d_r \frac{\sin\left(atan2(y_{r2} - y_{r1}, x_{r2} - x_{r1}) - \theta_{s2} - \theta_{r2}\right)}{\sin(\theta_{s2} + \theta_{r2} - \theta_{s1} - \theta_{r1})} \tag{2.8}$$

and,

$$d_r = \sqrt{(x_{r2} - x_{r1})^2 + (y_{r2} - y_{r1})^2} \tag{2.9}$$

Table 2.1 identifies the variables used in the parallax calculation equations.

Table 2.1: Parallax calculation variables

Symbol	Meaning
x_g	Updated target X position
y_g	Updated target Y position
x_{r1}	Robot X location at first reading
y_{r1}	Robot Y location at first reading
θ_{r1}	Robot heading at first reading
θ_{s1}	First AoA sensor reading
x_{r2}	Robot X location at second reading
y_{r2}	Robot Y location at second reading
θ_{r2}	Robot heading at second reading
θ_{s2}	Second AoA sensor reading
$x_{g,o}$	Target X position after previous update
$y_{g,o}$	Target Y position after previous update
x_{rn}	Robot X location at current update
y_{rn}	Robot Y location at current update
θ_{sn}	AoA sensor reading at current update
θ_{rn}	Robot heading at current update

Further goal position updates use a Kohonen-learning inspired updating procedure [18], which moves the last calculated goal position in the direction of the most recent vector reading according to a parameter, *k* (Figure 2.3). As more data is acquired, the calculated goal position converges to the actual target position through this continuous updating procedure. The choice of *k* is influenced by factors such as sensor noise, updating rate, etc. to achieve stable convergence in a reasonable period of time. It is possible to have *k* vary with time, or according to other metrics, but in the examples presented in this paper *k* was chosen to be a fixed value. The updating of the goal position is calculated with (2.10) and (2.11),

$$x_g = x_{g,o} + kd_e \sin(\theta_{sn} + \theta_{rn}) \tag{2.10}$$

$$y_g = y_{g,o} - kd_e \cos(\theta_{sn} + \theta_{rn}) \tag{2.11}$$

where,

$$d_e = d_o \sin\left(atan2\left(y_{g,o} - y_{rn}, x_{g,o} - x_{rn}\right) - \theta_{sn} - \theta_{rn}\right) \tag{2.12}$$

and,

$$d_o = \sqrt{\left(x_{rn} - x_{g,o}\right)^2 + \left(y_{rn} - y_{g,o}\right)^2} \tag{2.13}$$

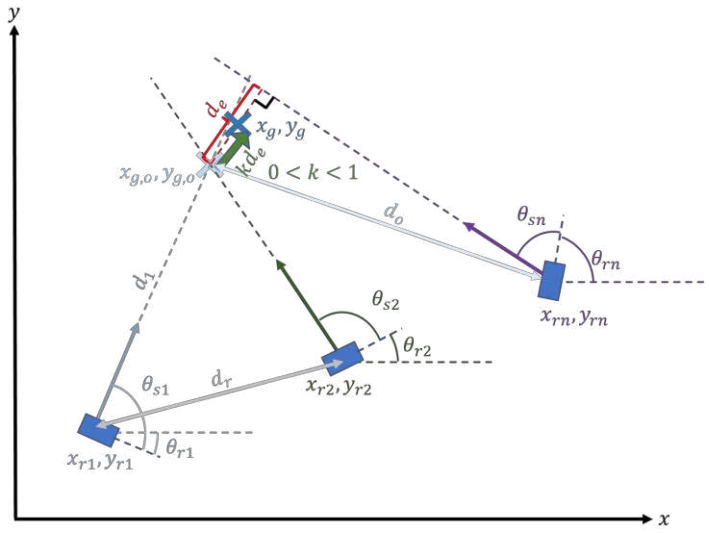


Figure 2.3. Parallax calculation goal position updating. The blue rectangles represent the robot (and antenna) positions at various points in time, while the blue X's represent the estimated target position (i.e. the estimated beacon position) at corresponding updates.

2.3.1.3 - Hybrid Controller Implementation

Both vector and parallax position calculations have drawbacks when applied using Bluetooth AoA in this application, preventing robust and accurate convergence to the true target position. Vector position calculation relies heavily on accurate distance measurements. While an approximation of distance between the beacon and the antenna array can be obtained using RSSI

to distance calibration, there is high variability in RSSI measurements, and the achievable accuracy of the distance estimation is relatively low, resulting in lower accuracy of the reported target position when using this data in the position calculation. Even with filtering of the signal, there is too much uncertainty in the distance estimation achieved using RSSI to attain high localization accuracies using a stand-alone vector calculation method, as demonstrated by the limited accuracy achieved by other RSSI-based indoor positioning systems. Parallax target position calculation is subject to high error when the angle readings are taken at locations close to each other since the vectors will be near-parallel, creating a large margin of error when calculating the intersection location for slight variations in angle. Due to the fast-updating nature of the controller, angle readings will be taken very close together, resulting in a high probability of early convergence to an inaccurate goal position, especially if the robot starts off facing in the direction of the target.

For the final controller design, a hybrid implementation of both methods of target position calculation was used in order to improve the speed and accuracy of convergence to the target position. In this hybrid implementation, at each time step the updated goal position is calculated using both the vector and parallax calculations, and these are compared as a function of distance to the target for an error metric using (2.14).

$$err = \frac{\sqrt{\left(x_{g,p} - x_{g,v}\right)^2 + \left(y_{g,p} - y_{g,v}\right)^2}}{0.5 * \left(d_p + d_v\right)} \tag{2.14}$$

Here, $(x_{g,p}, y_{g,p})$ is the parallax goal position, $(x_{g,v}, y_{g,v})$ is the vector goal position, and d_p and d_v are the distance between the robot and the goal position for the parallax and vector calculations, respectively. Equation (2.14) serves as an error-checking method for both goal position estimations by comparing the distance between them. The raw accuracy of the estimated goal positions is correlated with the distance between the robot and the goal position, so the

position error metric is normalized by the estimated distance to the target position, calculated as the average distance between each estimated goal position and the robot. The threshold for the error metric was chosen as 0.3 based on the expected noise of the vector position estimations. A lower threshold could result in accurate parallax estimations being flagged as errors due to vector position error noise, while a higher threshold increases the risk of false early convergence of the parallax position estimate due to greater allowable error between the 2 estimations.

The error metric is implemented so that if the error value is less than 0.3, the parallax goal position is reported as the new goal position for that update step. If the error value is greater than 0.3, the vector goal position is checked to see if it is an outlier to the previous 20 vector goal positions using a z-test with a threshold of 1.645, corresponding to a certainty of 90%. If it is not an outlier, then the parallax and vector goal positions are averaged and that location is reported as the new goal location. Additionally, the latest parallax goal position is updated to be equal to this averaged position to reduce the iterations required for convergence. If the latest vector position is an outlier, it is ignored and the parallax goal position is still reported as the new goal position. The controller continues updating the goal position as described until the robot has arrived at the target. Arrival at the target is achieved when the position of the robot is below a specified arrival threshold from the calculated goal position, and the difference between the parallax and vector goal positions is below an error threshold. The second condition ensures that the robot does not stop before arriving at the target due to early convergence of the parallax goal position calculation.

2.3.2 - Homing Controller Simulations

The homing controller algorithm was implemented in ROS and tested via simulations in Gazebo using a 4-wheeled differential drive mobile robot. A custom homing controller plugin was written in ROS using the algorithm described above in combination with the existing differential

drive controller in ROS2 control. The output of the homing controller was the goal velocity (determined from the target vector using proportional control) published to the differential drive controller, which then commanded the wheel motion of the robot. The odometry data of the robot was calculated within the differential drive controller and used by the homing controller. The controller architecture in Gazebo and ROS is shown in Figure 2.4. Bluetooth AoA and distance data was simulated with varying levels of sensor noise. For all simulations, the parallax updating parameter, *k*, was set to 0.8, and the arrival threshold was set to 0.3m.

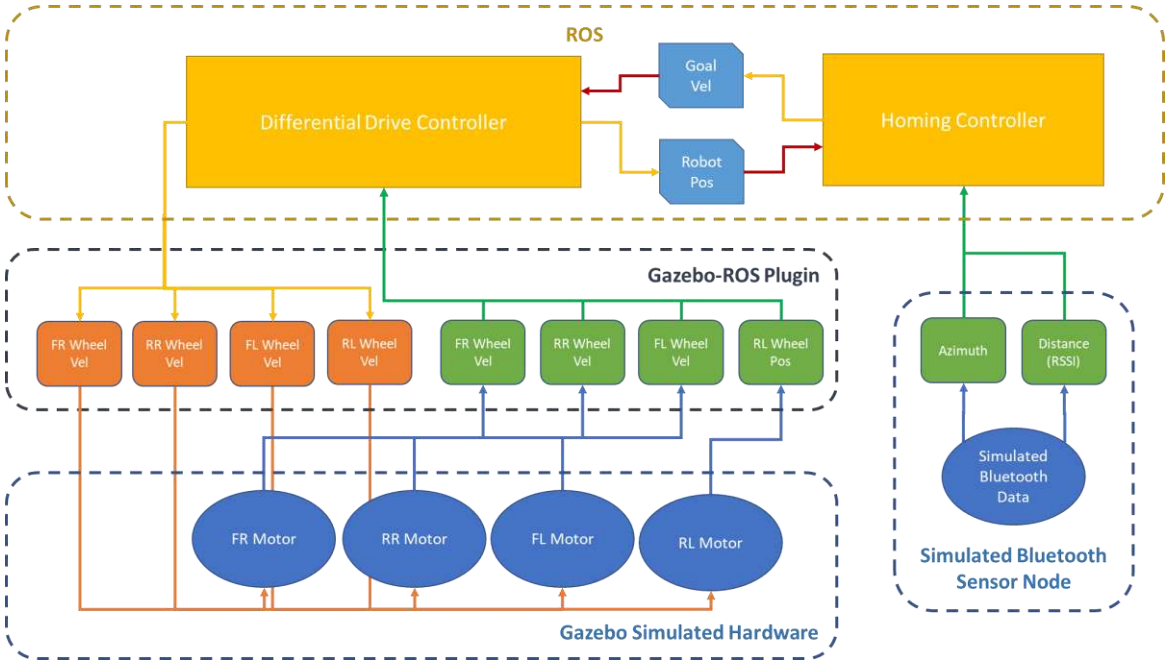


Figure 2.4. ROS controller architecture for Gazebo simulations.

Experiment 1: Homing Controller Algorithm Testing - The performance of the homing controller algorithm was first tested in an obstacle-free environment. The testing area was a -10m to +10m square with the robot starting at (0,0) and facing along the +X direction. A total of 40 simulations were performed with 10 randomly determined target locations in each quadrant of the

test area for each sensor noise configuration (shown in Table 2.2) to evaluate the algorithm sensitivity to sensor noise.

Table 2.2. Sensor noise configurations for simulations

Sensor Config.	1	2	3	4	5
Angle Noise	±5°	±5°	±10°	±10°	±20°
Distance Noise	±5%	±10%	±10%	±20%	±1m±20%

*All noise configurations correspond to uniformly distributed random noise within the given range from the ground truth value.

Experiment 2: Homing Controller with Obstacle Avoidance Testing - The homing controller algorithm with a simple rule-based obstacle avoidance algorithm was tested using the AWS Small House World. The test area was divided into 4 rooms (Figure 2.5), and 5 simulations were performed with every combination of start and end room, for a total of 80 simulations. For each simulation, the start location and orientation of the robot and the target location were randomly determined (using a random number generator) within each room. Start and target locations overlapping with objects in the world were not used. All simulations were performed using Bluetooth sensor noise configuration 5.

Figure 2.5. *AWS Small House World* rooms for obstacle avoidance simulations.

2.4 - Results

Experiment 1: Homing Controller Algorithm Testing Results

In the obstacle-free simulation testing, the robot had a 100% success rate in navigating to the target location, validating the homing controller algorithm for calculating target position. Additionally, it was found that sensor noise did not have a noticeable impact on controller performance for all tested levels of sensor noise. An example trajectory is given in Figure 2.6, and testing performance statistics are given in Table 2.3 (see footnotes of Table 2.3 for explanation of performance metrics).

Table 2.3. Homing controller algorithm simulation results

Sensor Config.	1	2	3	4	5
Success Rate (%)[1]	100	100	100	100	100
Mean Time to Target (s)[2]	14.24	12.68	12.10	12.85	12.27
Mean Net Vel. (m/s)[3]	0.62	0.57	0.60	0.60	0.57
Mean Goal Pos. Err (m)[4]	0.05	0.05	0.06	0.06	0.11
Mean Final Pos. Err (m)[5]	0.25	0.24	0.25	0.25	0.15
Mean Path Efficiency[6]	0.90	0.88	0.90	0.89	0.91

[1]Success rate is the percent of simulations which successfully terminated with the robot located <1m of actual target location.

[2]Time to target is the time for simulation to successfully terminate.

[3]Net velocity is the straight- line distance between robot start and end positions divided by the time to target.

[4]Goal position error is the distance between the final estimated target position and the actual target position.

[5]Final position error is the distance between the final robot position and the actual target position.

[6]Path efficiency is straight line distance between robot start and end positions divided by the total distance traveled.

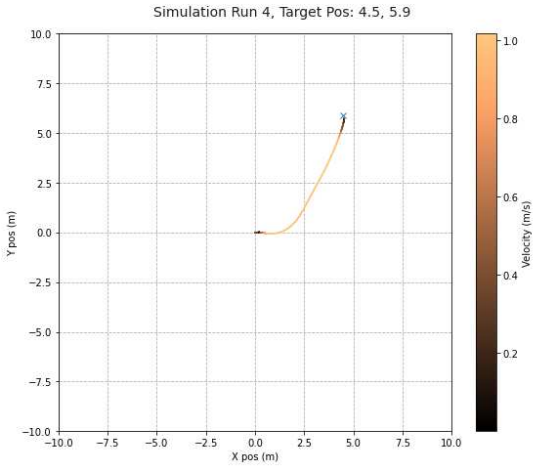


Figure 2.6. Example trajectory for obstacle-free homing controller simulation testing.

Experiment 2: Homing Controller with Obstacle Avoidance Testing Results

The homing controller with obstacle avoidance had a 96% success rate in navigating to the target. All three failures resulted from the robot hitting an obstacle that was not detected by the

sensors and being unable to recover – an expected result for simple rule-based obstacle avoidance such as what was implemented in the controller. Additionally, the mean time to target was much higher and the mean net velocity was much lower than in the obstacle-free scenarios. This is also expected because detours are required for obstacle avoidance, and reactionary rule-based obstacle avoidance generally will not result in the most efficient trajectory to the goal location. Importantly, the mean final position error and goal position error were not significantly higher than in the obstacle simulations. This validates the ability of the homing controller algorithm to converge to the correct target position even when the robot is detoured around obstacles, as would be expected in a real-world application scenario. A sample obstacle avoidance simulation trajectory is shown in Figure 2.7 and simulation results are given in Table 2.4.

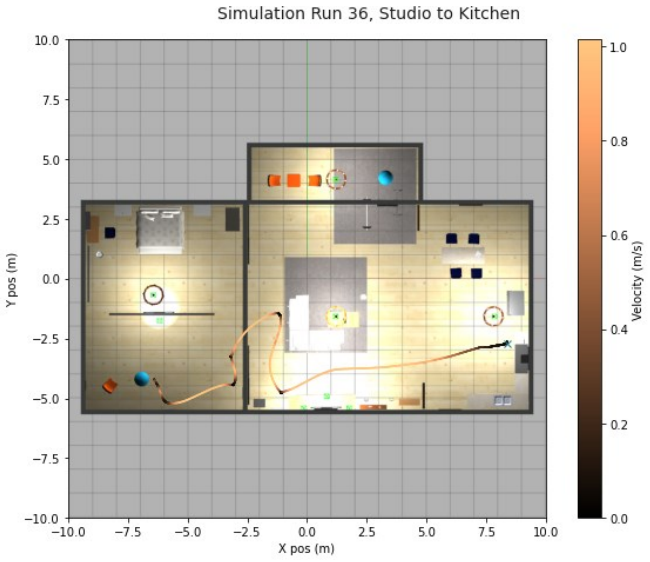


Figure 2.7. Example trajectory from homing controller obstacle avoidance simulation testing.

Table 2.4. Obstacle avoidance simulation results

Metric	Result
Success Rate (%)	96
Mean Time to Target (s)	44.91
Mean Net Vel. (m/s)	0.22
Mean Goal Pos. Err (m)	0.12
Mean Final Pos. Err (m)	0.23
Mean Path Efficiency	0.69

Metrics in Table 2.4 are equivalent to those in Table 2.3. All metrics except success rate are only calculated over successful simulations.

2.5 - Discussion

Simulation results validated the ability of the Bluetooth AoA-based homing controller algorithm to accurately converge to a desired target position. Additionally, the controller algorithm was successfully implemented as a standalone controller to command goal velocities to navigate the robot to a target location in an unknown environment. This shows the versatility of the algorithm in its ability to produce different outputs (goal position or goal velocity) depending on the implementation for which it is used. While compatibility with path-planning or other existing autonomous navigation algorithms was not tested, it is expected that the controller would be able to converge to an accurate goal position in these implementations as well. Furthermore, the obstacle avoidance simulation testing validated the algorithm in the scenario where the robot is unable to navigate directly to the goal position and must detour along the way. There was no decrease in target position accuracy when the robot was required to take a detour.

The controller algorithm also proved robust to noise in sensor readings, achieving high target position accuracy even in the presence of significant sensor noise. For the presented simulations, the noise level was constant throughout each simulation. It should be noted, however, that in real-

world implementations it is expected that sensor noise and accuracy will vary with distance to target and line-of-sight obstructions. While the simulations were performed with high levels of noise to approximate worst-case scenario sensor noise and inaccuracies, future work should include more detailed modelling of variations in sensor readings due to these factors, along with further validation through hardware testing in a range of scenarios, including no line-of-sight to the target.

The homing controller was implemented such that the goal was to navigate to within 0.3m of the estimated target location. The mean final position errors in all simulations, both obstacle-present and obstacle-free were below this threshold, validating that in addition to accurately localizing the target, this localization was sufficient to navigate the robot to an acceptable placement with respect to the target. The value of this mean final position error is dependent on the approach distance metric of 0.3m used for termination of the simulation. A lower final position error could be achieved by reducing this metric, but for most identified potential applications of the proposed homing controller, navigating to within 0.3m of the target would be sufficient. Once the robot is within this distance, it is likely that additional sensing would be employed to visualize the target and carry out the desired task.

The hardware implementation of AoA technology presents additional challenges, the most significant of which is the current limitation of most algorithms reporting angles only between ±90°. This presents a challenge for the controller if there is not an indication whether or not the angle readings being received are valid and should be used to update the goal position. There are numerous solutions that could be explored to address the current limitations with AoA hardware, such as signal processing with machine learning and use of other sensors to determine orientation of the antenna array with respect to the target (such as a blocked antenna that only receives signal

when facing the target). If it can be determined whether or not the AoA reading is within the good range of the antenna array, the controller algorithm can easily be modified to only update the goal position when the robot is within this range, and be commanded to rotate until it is in the range if it is currently outside of the usable range.

Chapter 3 – Efficient Navigation in Partially-Known Environment

3.1 - Introduction

Mobile robots greatly expand the capabilities of robotics by allowing robots to navigate through an environment to perform jobs, such as object retrieval or delivery tasks. A primary challenge of mobile robot control is autonomous navigation. Autonomous navigation consists of two main objectives – reaching the desired target location and avoiding obstacles in the environment while navigating to the target location. There are two main approaches toward solving the mobile robot navigation problem: offline path planning and online reactive navigation.

In offline path planning, the full environment is known *a priori* and this information can be used to plan the path for the robot to take through the environment using a variety of optimization techniques. Once the full path has been optimized, it is then sent to the robot, which follows the designated path to arrive at the target location. The advantage of this method is that full knowledge of the environment allows for optimization of the robot path, resulting in shorter distances traveled and faster arrival to the target location. However, the major drawback of this method is its lack of adaptability. Offline path planning does not account for unknown or dynamic obstacles that the robot may encounter or the potential for changing of the target location during the navigation. These are all common occurrences in real-world mobile robot deployment scenarios, and thus offline path planning does not provide a robust solution for real-world environments.

Reactive navigation strategies operate in real-time, relying on sensor feedback to determine the optimal navigation path. Because these techniques plan their navigation in real-time, they are much more effective at avoiding dynamic obstacles and operating in unknown environments. However, because these strategies rely on sensor data within the line-of-sight of the robot without

full knowledge of the environment, such methods are susceptible to encountering dead-ends and taking less efficient paths than would be found by optimizing over the full environment. This becomes a greater problem as the complexity of the environment increases and there are more opportunities for the robot to navigate along a sub-optimal path to the target, and even potentially to get stuck if the navigation algorithm is unable to extract the robot from a dead-end.

The shortcomings of both offline and reactive navigation strategies suggest the need for a hybrid solution. There are already some navigation algorithms that can be used for both offline path planning and in an online reactive implementation. One of these is the artificial potential field (APF) path planning algorithm [19]. This algorithm uses the idea of potentials to create a representation of the environment, where locations far from the target and near obstacles have higher potential values, while locations further from obstacles and closer to the target location have lower potentials. The global minimum of the potential is located at the target location. Using this potential-based representation of the environment, a steepest gradient descent approach can be used on the potential field to guide the robot to the target location. The APF map can be created using either known environmental information or created in real-time using sensor data from the robot. In both cases, APF suffers from the possibility of converging to a local minimum, preventing the robot from reaching the final target location. Additionally, following the method of steepest gradient descent does not guarantee the shortest (or an alternate definition of the "optimal") path to the target will be taken. In this chapter, we propose a modified implementation of the APF method to address this shortcoming. The proposed method is implemented as a hybrid online/offline algorithm, where known environmental information is used to create a skeleton highway map through the environment, which is then used in real-time to guide the robot along a

near-optimal path to the target location. The real-time implementation allows the robot to avoid unmapped and dynamic obstacles and to adapt to changing target locations.

3.2 - Background

Mobile robot navigation is an extensively studied topic in autonomous robotics, and there are many approaches for both offline path planning and reactive navigation strategies. Examples of offline path planning approaches include cell decomposition [20][21], the roadmap approach [22][23], and the artificial potential field approach [24][25]. Some variations of these methods incorporate real-time updating based on sensor data. Examples of the most popular reactive navigation strategies include evolutionary (e.g., genetic) algorithms [26][27], fuzzy logic [28][29], neural networks [30][31], the firefly algorithm [32][33], and particle swarm optimization [34][35]. These reactive approaches have proven effective for navigating unknown environments, avoiding dynamic obstacles, and even adapting to changing goal positions [36].

APF is an attractive navigation strategy because it can be applied as both an offline path planning method and an online reactive navigation strategy using real-time sensor information. However, in both implementations, traditional APF suffers from some drawbacks, including a tendency to converge to local minima in cluttered environments, which prevents the robot from reaching the actual goal location. Another potential issue with APF is failure to reach the goal location when there are obstacles located within the vicinity of the target location due to the creation of a false global minimum. Various solutions have been proposed to modify the traditional APF methodology to address some of these shortcomings. When using APF as an offline path planner, optimization techniques have been used to minimize these problems. Lin et al. proposed a particle swarm optimization (PSO) search to identify the optimal parameters for APF to improve performance, implemented alongside a fuzzy-based dynamic window approach to improve

dynamic obstacle avoidance [37]. Orozco-Rosas et al. developed a membrane evolutionary APF to improve path planning over the traditional APF [38].

Alternatively, for real-time applications, various online adaptive modifications have been proposed to improve on traditional APF. Azzabi et al. proposed the use of a virtual escape force when a local minimum was detected to address the local minimum problem. Furthermore, they proposed a modified attractive force to address the problem of the robot being unable to reach the goal location if obstacles were nearby [39]. Bounini et al. presented a modified APF in which repulsive potentials were added in locations of local minima [40]. The concept of a black hole potential field (BHPF) was introduced in [41], whereby a potential field with a much higher gradient was employed in the immediate vicinity of a target to prevent the problem of a local minimum created by multiple target points in close proximity to each other. Furthermore, reinforcement learning was used in combination with the BHPF to help the robot to learn to avoid or escape from local minima created by obstacles. Fan et al. used the regular hexagon-guided method for escape from local minima [42]. Szczepanski et al. proposed a predictive APF algorithm which uses future movement prediction to identify local minima in advance and place virtual obstacles, thus preventing the robot from navigating into these local minima [43]. This method also uses future movement prediction to produce smoother and more efficient paths, but requires a higher amount of computation due to the prediction occurring at each update cycle. It should be noted that this is not an exhaustive summary of all APF-based robot navigation methodologies, but an effort was made to present a representative sampling, with a focus on more recent works.

3.3 - Methods

In this chapter, we propose two modifications to the standard APF implementation. The main goal of these modifications is to improve the path efficiency of APF-based navigation by guiding

the robot along a near-optimal path. Additionally, the proposed methodology helps to prevent convergence to local minima by reducing local areas of high potential in narrow corridors which prevent passage with traditional APF and providing additional attractive force to pull the robot through local minima encountered during navigation to the target. For the offline portion of the proposed algorithm, we use a grid-based representation of the environment. In such a representation, the known map of the environment is represented using grid squares, which are designated as either obstacle squares or free squares. During the grid construction, an offset was added to all obstacles equal to the distance from the center of the robot to its furthest edge so that the robot could be treated as a point object while avoiding collisions. The relatively low cost of computation for constructing the potential field, especially using a grid-based model which can control the number of computations per iteration, allows for updating of the potential field in real-time. This in turn results in the ability to avoid unmapped obstacles and adjust to potential changes in target location. Grid size can be pre-specified or determined using an adaptive method based on environmental complexity. In this chapter, results were obtained using an adaptive method of grid size determination. The complexity of the environment was estimated using a raster scan to create a list of the lengths of lines in free space both horizontally and vertically. The grid size was then computed using the number and length of free space lines with (3.1), where l_i is the length of the *i*th line and *n* is the number of free space lines.

$$gridSize = \sqrt{\frac{\sum_{i=1}^{n} l_i}{n}} \tag{3.1}$$

3.3.1 - Modified Obstacle Influence Distance

In the standard APF algorithm, two parameters are set which determine the magnitudes and influence distances of the repulsive forces generated by obstacles in the environment. These

parameters are taken to be constant throughout the environment, regardless of the position of obstacles relative to each other in the environment. This can create local minima generated when two obstacles are too close together, such that the influence distances of the repulsive forces are large enough to make the obstacles overlap. We propose a modification to the influence distance parameter when this situation occurs, reducing the influence distance of the repulsive force of the obstacles at the affected locations so there is no longer an overlap, thus removing the local minimum and creating a pathway of decreasing potential between the two obstacles (Figure 3.1).

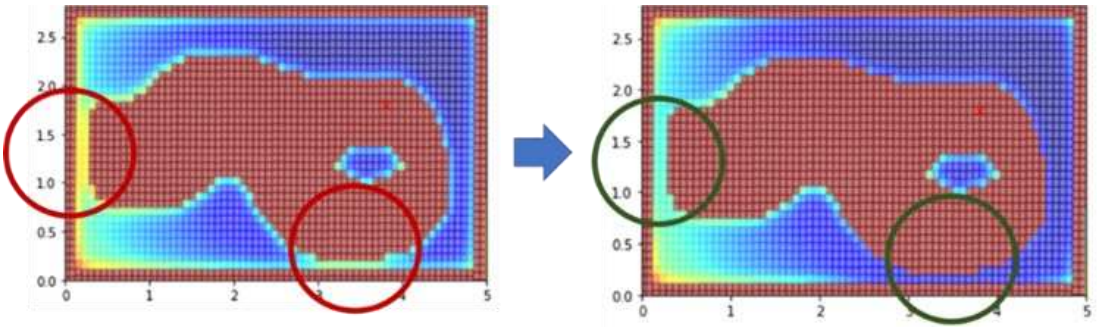


Figure 3.1. Example of modified obstacle influence distance eliminating higher potentials which prevent passage through narrow corridors.

Using the grid representation of obstacle space, the modified obstacle influence distance is determined by clustering obstacle squares together and determining the minimum distance between each square and the closest square in all other obstacle clusters. The minimum distance to any square in another obstacle cluster is used to modify the obstacles influence distance using,

$$d_{new} = \frac{d_c}{2} - d_{min} \tag{3.2}$$

Where d_c is the distance to the closest obstacle square, and d_{min} is a parameter specifying the minimum desired clearance distance (i.e. width with no repulsive force influence between obstacles). If d_{new} is initially calculated as <0, the obstacle influence distance is not modified. In

this case the obstacles are too close together to maintain the desired minimum clearance distance and there is no way that the robot can pass safely between them.

The modified obstacle influence distance is then stored in a lookup table such that each obstacle square has a calculated modified obstacle influence parameter. When calculating the repulsive potential field, the potential is calculated for each individual obstacle square with its corresponding influence distance parameter, and the full repulsive field is the summation of each of these individual repulsive fields. In order to prevent repulsive field magnitude changes with variations in grid size, each individual repulsive field is scaled by the area of the grid square.

3.3.2 - APF with Highway (APF-HW)

The introduction of the modified repulsive force influence distance addresses the case where a local minimum is encountered due to obstacles being positioned too close to each other. However, there are other environmental configuration scenarios that can also result in convergence to a local minimum using APF path planning which are not solved by this modification. Therefore, we propose a method of introducing a 'highway' framework into the potential field which will guide the path planning along a near-optimal route to the target, including moving through local minima encountered along this path. The idea here is that a number of highway points are generated in free space of the robot, and paths between these points are constructed based on line-of-sight. These local paths can then be used to determine the optimal 'highway' path between any set of these points. Points along this path are used to generate an additional attractive force which guides the steepest gradient descent along this path until the target position can be reached without encountering any additional obstacles. A detailed description of the construction and implementation of this 'highway' infrastructure is described in the following sections.

3.3.2.1 - Construction of the Highway

Construction of the highway consists of two steps: highway point selection and creation of the best path table. Highway point selection can be performed using user-selected parameters or using an adaptive method based on the environment map. First, we describe the basic method of highway point selection with user selected parameters. Then, we describe an adaptive variable-density method of highway point selection designed to optimize the number and locations of highway points to increase the coverage of the possible highway paths in the environment.

User selected parameters include the number of points and minimum distance between points. At initialization of the point selection, the center of every free grid square is considered a potential highway point location. Point selection occurs in an iterative fashion, where each new potential point is randomly selected from the remaining available free grid squares. This potential point is then checked for its distance to each of the already selected highway points. If it is at least the specified minimum distance from all the already selected points, the point is selected and added to the list of highway points. If it is located less than the minimum distance from any of the already selected points, the point is not selected and a new random potential point is checked for the minimum distance. The highway points are selected iteratively in this manner until the specified number of points has been selected. If at any time during this point selection procedure no more points can be selected maintaining the minimum distance criteria, the minimum distance parameter is reduced by 10% and the point selection process is continued.

The minimum distance parameter is employed to ensure a pseudo-uniform distribution of points throughout the free space, and thus should be chosen based on the size of the working area. It should be noted that the minimum distance parameter will be reduced automatically if the original value prohibits the selection of the designated number of highway points, and thus it is

preferable to start with a larger value for this parameter, since a small value could result in areas with high concentration of highway points and underrepresentation in other areas, which will result in suboptimal performance. The selection of the "number of points" parameter is highly dependent on the configuration of the environment. For highly complex environments, a larger number of points is needed to improve the probability of having highway paths between all areas of the environment and the optimality of these paths. However, the construction of the best path list becomes more computationally expensive with increasing numbers of points, so it is desirable to minimize this number as much as possible.

Alternatively, an adaptive method of highway point selection can be employed, which seeks to optimize the selection of highway points based on the given environment map. First, a density map is created of the environment, representing the location and proximity of each square to obstacles in the environment. For each free space grid square in the environment, the density value for that square is calculated with (3.3), which is a measure of the local complexity of the environment and the proximity of the point to obstacles. This metric is designed to increase the probability of point selection in complex areas with dense obstacles, while encouraging point selection located further from actual obstacle space. It is calculated by determining how centered the selected grid square is in free space and dividing that by the overall width of the surrounding free space in both the horizontal and vertical directions. n_l, n_r, n_u, and n_d are the number of consecutive free space grid squares located left, right, up, and down from the given square, respectively.

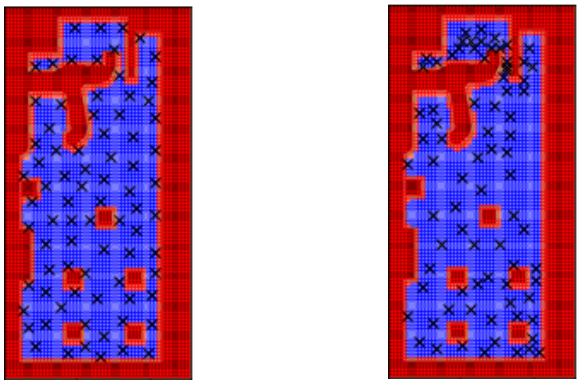
$$d_i = \frac{1 - \frac{|n_l - n_r|}{n_l + n_r + 1}}{(n_l + n_r + 1)} + \frac{1 - \frac{|n_u - n_d|}{n_u + n_d + 1}}{(n_u + n_d + 1)} \tag{3.3}$$

This density map is then smoothed using a Gaussian blur function and the number of points is computed with (3.4).

$$n = 0.75 * \sum d_i \tag{3.4}$$

Finally, the highway points are selected iteratively using probability-based random selection, where the probability of each point being selected is its corresponding value in the density map. The minimum distance a point must be located from other selected points is also adjusted based on its density map value using (3.5), where n is the number of map points, n_f is the total number of free grid squares, and d_i is the density map value. Figure 3.2 shows a comparison of random highway point selection versus this variable-density point selection method. Note that the scaling factors for the adaptive number of points and minimum distance were determined experimentally.

$$dist_{min,i} = \frac{6 * gridSize * n}{n_f d_i} \tag{3.5}$$

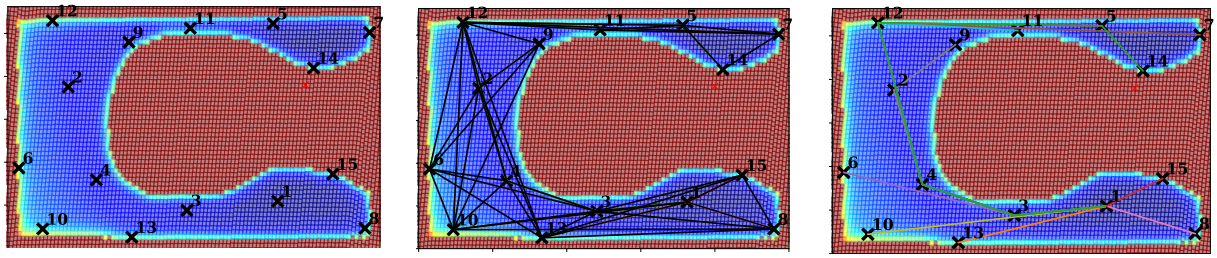


a) b)

Figure 3.2. Comparison of random psuedo-uniform highway point selection (a) with variable-density point selection (b).

Once the set of highway points has been selected, the best path table is computed. The first step in computing the best path table is to calculate connections between points. This is done using line-of-sight, where a given point has a connection to another point if a straight line can be drawn

between the two points without encountering an obstacle grid square. For each point, all of its possible point connections are stored in a table. This point connection table is then used to construct the best path table, which stores the shortest highway path between each set of highway points. The best path list is constructed iteratively using all of the point connections from a point, then all of the connections from those points, etc. The distance of the paths between each set of points is calculated and if a new path between two points is shorter than the existing best path, this new path is selected as the best path between those points. This iterative path construction procedure continues until no new shorter paths can be generated between any set of points. The steps of the highway construction procedure are shown for an example environment in Figure 3.3.



a) b) c)

Figure 3.3. Example of the highway construction procedure. (a) shows the highway point selection, (b) shows the generation of all possible point connections, and (c) shows the best path between one highway point and all other highway points.

3.3.2.2 - Implementation of Highway

The selection of highway points and computation of the best path table occurs only once at the initialization of the path planning problem. The tables storing the locations of the highway points and the configurations of the best path lists are then referenced when the highway is engaged. The objective of implementing this highway method with APF is to guide the robot on a near-optimal path in complex environments where local minima must be navigated to reach the final target. To this end, use of the highway is not always required – specifically in the case where there is already a line of sight to the target location. Furthermore, the addition of the highway increases the

computational complexity for each update iteration, and could reduce the path optimality by pulling the robot along the highway path instead of directly toward the target. Because of this, the APF with highway method is designed so that the highway points are only engaged when the robot does not have a line of sight to the target. This is calculated at the beginning of each update iteration by checking if a straight line drawn between the robot and the target location intersects with any obstacles. If it does, the highway is engaged; if not, the APF proceeds using the typical method. For each update iteration when the highway is engaged, the following steps are executed.

Step 1: Selection of Near Points

The first step in engaging the highway is the selection of near points. These are highway points located near the robot which serve as potential 'entry' points to the highway paths. Near points are selected as any points with a line of sight to the robot. Alternatively, near point selection could be limited to points located within a designated radius of the current robot position (still with a line of sight to the robot), but this has the potential to lead to problems following highway paths in scenarios where the points are spread out.

Step 2: Selection of Target Points

Next, the target points are selected. The target points are highway points located close to the target location and serve as a potential 'exit' to the highway. Target points are selected as any highway points located within a distance r_t and with a line of sight to the target location. If no highway points meet both of these criteria, the selection of target points is extended to include any highway points with a line of sight to the target location, even if they are located outside of the radius r_t.

Step 3: Selection and Weighting of Potential Highway Paths

In order to avoid redundancy, highway paths are selected such that only one highway path to each target point is implemented. The selection process of highway paths is performed as follows. For each target point, there will be up to n potential highway paths to that point, where n is the number of near points. Each path corresponds to the path from one of the near points to that target point. The configuration of each of these paths is looked up from the best path list table. Then, the distance of each of these paths is calculated by summing the distance of each leg of the path (determined using the locations of the points on the path) with the distance from the robot to the corresponding near point. All of these distances are compared, and the path with the minimum total path distance is selected as the final path for that target point. Finally, the weighting for the highway path associated with that target point is calculated using (3.6). This is done to increase the weight of shorter highway paths, which correspond to more optimal paths to the target.

$$w_{path,i} = \frac{10}{d_{path}} \tag{3.6}$$

This same highway path selection procedure is repeated for each target point, so that at the end of highway path selection, there are a number of highway paths equal to the number of target points, each corresponding to a unique target point and weighted by a parameter corresponding to its total distance.

Step 4: Implementation of Highway Paths

Once the highway paths have been selected, they must be transformed into forces which can be summed with the forces generated by the original potential field to create a final force that can be used to determine the path of the robot. For each selected highway path, its contribution to the overall force vector exerted on the robot is calculated as follows. First, each point on the highway

path is checked for a line of sight to the current robot position. All points on the highway path with a line of sight to the robot are added to a list. Each of these points exerts an attractive force on the robot proportional to the distance between the point and the robot, and is normalized by the number of attractive points on the path, the number of highway paths, the path weighting, and the magnitude of the general attractive force at that point. The full equation for calculating the attractive force of point *j* on the path for target point *i* is given in (3.7),

$$F_{HW,i,j} = w_{path,i} \| F_{att} \| \frac{1}{n_t n_p} \vec{q}_j \tag{3.7}$$

Where n_t is the number of target points, n_p is the number of points along path *i* within the line of sight of the robot, and $\vec{q}_j$ is the vector between the robot and point *j* on path *i*. Such a force vector is computed for every highway point with a line of sight to the robot for each of the selected highway paths. These force vectors are then summed together and weighted by a highway force weighting parameter, *m*, and added to the existing force vector computed from the potential field. The final force vector is computed using (3.8).

$$\vec{F}_{tot} = \vec{F}_{att} + \vec{F}_{rep} + \vec{F}_{HW} \tag{3.8}$$

The full procedure of near and target point selection, highway path selection and weighting, and implementation of highway paths is performed for every update iteration where the highway is engaged. Figure 3.4 summarizes the algorithm for each controller update period.

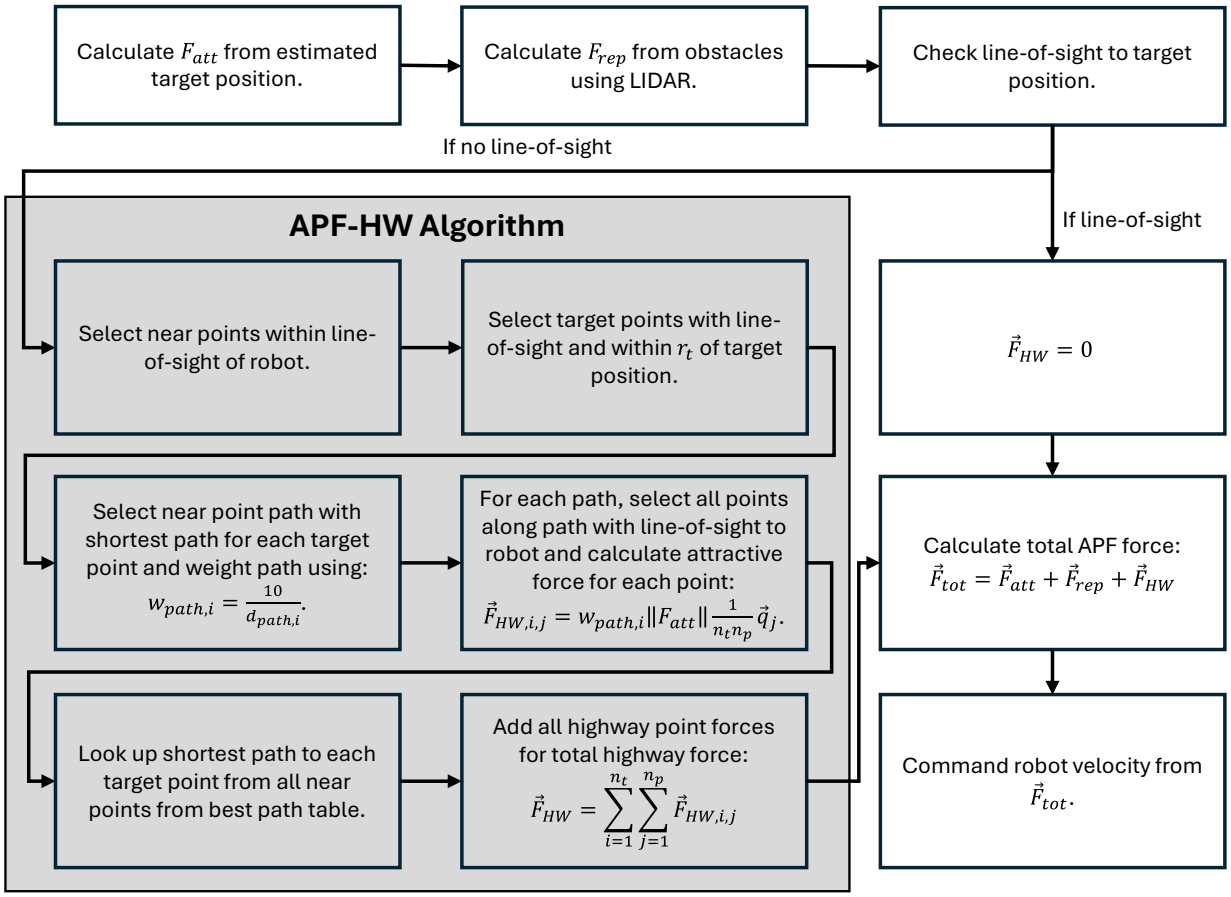


Figure 3.4. Flowchart of the APF-HW algorithm implementation for one controller update period.

3.3.3 - APF-HW Path Planning Experiments

The first set of experiments focused on offline path planning using the proposed APF with Highway (APF-HW) method. The goals of these experiments were to validate the proposed method as a path planning method by evaluating performance, investigate the impact of highway path weighting on the performance of the method, and identify strengths and weaknesses of the method with respect to varying environmental characteristics.

The path planning experiments were run in Python using the APF-HW path planning algorithm as outlined in this paper. The adaptive method of parameter and highway point selection was used

for all experiments. Some notes on the implementation of this algorithm for these experiments follow.

- Path construction – At each update iteration, the next path point was calculated by moving along the force vector calculated at the current robot location, scaled by the magnitude of the force vector. Minimum and maximum step sizes of 0.005m and 0.05m, respectively, were implemented.

- Convergence criteria – The path planning was considered converged if any of the following criteria (determined experimentally) were met:

    - $\|F\| < 1e-2$

    - The direction angle change in the force vector was > (π-0.3) rad between consecutive iterations for more than 2 of the last 10 iterations (to prevent oscillations)

    - 2,000 iterations were performed

A total of 10 environment configurations of varying complexity were used for path planning testing. In addition to the obstacles in the environment, the edges of each floorplan were considered to be walls that were also treated as obstacles, and thus had an offset added to prevent collisions. For each environment configuration, four start positions and 3 target positions were identified. The sets of start and target positions were unique to each room configuration. The experiments investigated the effect of the highway weighting parameter on the performance of the algorithm. A total of five weighting parameters were tested with the same four grid sizes.

3.3.4 - Real-Time Gazebo Simulations Comparing Traditional APF, APF w/Escape Force, and APF-HW

The second set of experiments was performed to compare the performance of the proposed APF-HW algorithm with the traditional APF method, and the APF with Escape Force method [39]. These experiments were performed as real-time simulations using a ROS controller simulated in Gazebo, where the APF was implemented in real-time using sensor and target position feedback. Some notes on the implementation of the algorithms for these simulations follow:

- Control infrastructure – All simulations were performed using ROS2 control and Gazebo using the same mobile robot configuration. A custom controller was written for the APF control portion, and commands were sent directly to the mobile robot using the integrated ROS2 control differential drive controller.

- Repulsive force calculation – The repulsive force was calculated using LIDAR feedback implemented in the simulation. The modified obstacle influence distance was implemented by clustering LIDAR readings into separate obstacles and using the clusters to determine the appropriate modified obstacle influence distance in a similar method to that described in Section 3.3.1 - Modified Obstacle Influence Distance. It should be noted that the modified obstacle influence distance was only implemented for the APF-HW trials.

- Target position – For all simulations, the target location was calculated in real-time using the Bluetooth homing algorithm presented in Chapter 2. It should be noted that use of this algorithm results in fluctuations in the estimated target position throughout the simulation, in comparison to a constant pre-determined target position which is usually seen in robot path planning.

- Line of sight to highway points – All highway points and paths were constructed using the floorplan maps shown in Figure 3.5. However, during the simulations, line of sight to highway points was determined using the odometry data of the robot, the table of highway point locations, and LIDAR data from the robot detecting obstacles. The LIDAR reading in the direction of the highway point was compared to the calculated distance between the robot location and the highway point, and if the LIDAR reading was greater than this distance, then it was taken that there was a line of sight to the point.

- Robot motion commands – The ROS2 control differential drive controller takes inputs of goal linear and angular velocities. At each update iteration, the output of the APF algorithm is a force vector. This force vector was then converted into goal velocities using (3.9) and (3.10), where θ_F is the angle of the force vector, and k_l and k_a are the gain parameters for the linear and angular velocities, respectively.

$$v = \begin{cases} k_l \|F\| \dfrac{2}{\pi} \left| \dfrac{\pi}{2} - |\theta_F| \right| & if\ |\theta_F| < \dfrac{\pi}{2} \\ 0\ otherwise \end{cases} \tag{3.9}$$

$$\omega = k_a \theta_F \tag{3.10}$$

- Convergence – The APF was considered converged to the final position when both of the following criteria were met (second condition was determined experimentally):
    - Line of sight between robot and estimated target position
    - $\|F\| < 0.4$ or $\|F_{att}\| < 0.4$

Test simulations were performed using three different test environments in Gazebo - the AWS Small House World, the AWS Bookstore World, and the Café World (Figure 3.5). The AWS Small House World was divided into four ‘rooms’, and five simulations were performed for each

combination of start and end room, for a total of 80 simulations for each method. For each simulation, both the start and target locations were randomly generated within the designated room. The same 80 start and target positions were used for comparison between the three methods. For both the AWS Bookstore World and Café World a total of 50 randomly generated starting and target positions were used for simulations. The same 50 simulations were used for all control methods for the most accurate comparison.

The floorplans used to create the highway for the APF-HW simulations are shown in Figure 3.5. In some cases, the floorplan is not an exact match with the available free space of the mobile robot as some obstacles are missing, and obstacles such as tables and chairs are configured differently between the floorplan and in robot space (e.g. the robot can move under the table but in the floorplan it is represented as a solid obstacle). This was done intentionally to test the robustness of the algorithm to differences between the floorplan used for highway construction and the actual environment encountered by the robot. Additionally, the highway points were selected using both the random selection method and the variable-density method for comparison of performance in the 3 different environments. Based on the results of the offline path planning experiments, a highway weighting parameter of m=50 was used for all APF-HW simulations.

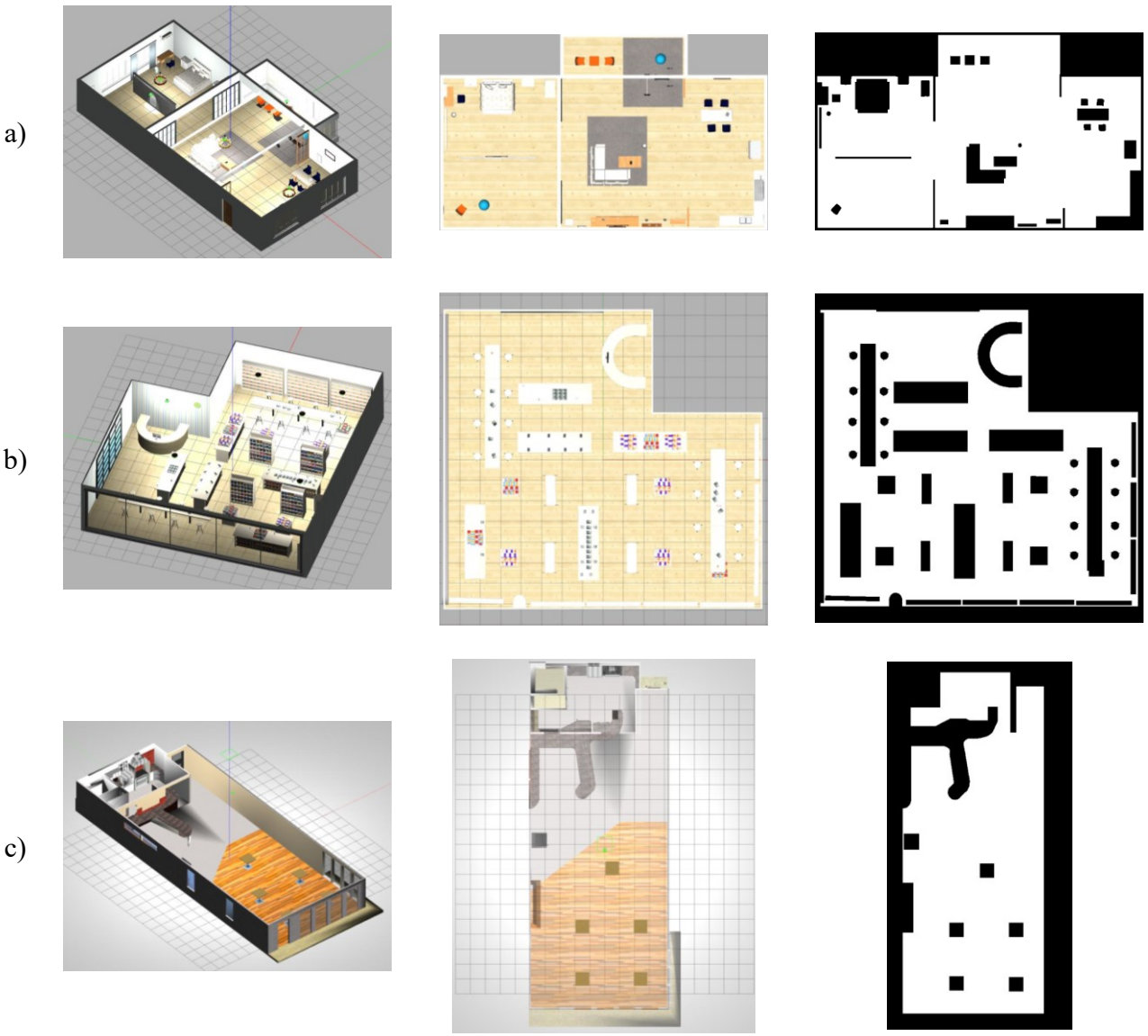


Figure 3.5. Gazebo test worlds and the corresponding floorplans used for real-time simulation testing of the various APF algorithms. (a) AWS Small House World, (b) AWS Bookstore World, (c) Café World.

3.4 - Results

3.4.1 - APF-HW Path Planning Results

The goal of the experiments presented in this section was to evaluate the effect of the highway weighting parameter, m, on algorithm performance. This was assessed in two ways: (i) the influence of the parameter on success rate, where it was hypothesized that a minimum threshold

for the parameter could be determined to maximize success rate, above which changes to the parameter would have little effect on the success rate of the algorithm, and (ii) the influence on other algorithm performance metrics, the most important of which were determined to be path distance and number of steps to convergence.

First, we evaluate the influence of the highway weighting parameter on the success rate of the algorithm over the different simulations. A trial was considered successful if it terminated within 0.3m of the target position. The success rate over all simulations as a function of weighting value given in Table 3.1. We can see that the lowest success rate is achieved with the lowest weighting value of 10. However, for weighting values of 25 and above, little increase in success rate is achieved, as all weighting values above 10 achieve success rates of at least 99%. Any trial failures for these weighting values can likely be attributed to variations in highway point selection rather than the selected highway weighting parameter.

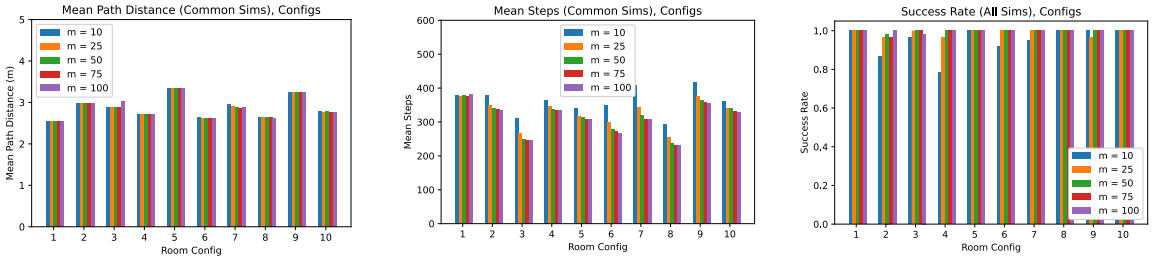
Table 3.1. Offline path planning results with APF-HW for varying highway weighting parameter values

Highway Weight, m	**Path Distance (m)**		**Number of Steps**		**Success Rate**
	Mean	Median	Mean	Median	
10	2.87	3.00	360.15	376.5	0.948
25	2.87	3.02	326.67	333.5	0.990
50	2.87	3.01	316.11	314.5	**0.998**
75	2.86	3.02	310.54	306.0	0.997
100	2.88	3.04	309.38	305.5	**0.998**

*Metrics were calculated over trials (environmental configuration, start position, and target position) that were commonly successful between all highway weighting parameter values.

To further investigate the effect of the weighting parameter in different environments, the success rate was evaluated for each weighting parameter value for each test environment (Figure

3.6). This was used to evaluate the correlation between the environment and the weighting parameter value on algorithm success. In other words, it was hypothesized that more complex environments may require a higher value for the weighting parameter to achieve high success rates. We see this trend in the lower success rates achieved by m = 10 in configurations 2, 3, 4, 6, and 7, which were the more complex environments in the test set. This makes sense from a structural perspective, as more complex environments have narrower pathways which must be navigated, requiring stronger highway pull to travel through local minima created by repulsive obstacle forces in these narrow areas. For the simpler environments, even the smallest weighting parameter was sufficient to achieve success in reaching the target.



a) b) c)

Figure 3.6. Performance metrics for difference highway weighting parameter values evaluated for each environmental configuration. Mean path distance (a) and mean steps (b) were evaluated on commonly successful simulations for all weighting parameter values for a given environment.

The other important performance metrics to evaluate are path distance and number of steps. No statistically significant correlation was found between path distance and highway weighting parameter value. However, there is a trend of decreasing number of steps with increasing highway weighting parameter values. This indicates that higher weighting parameter values result in larger step sizes, and thus fewer total steps to reach the final target location. This also makes sense, as higher weighting values will correspond to higher forces, which influences the step size, itself determined by the magnitude of the net force. Generally, larger step sizes correspond to more

efficient navigation, as fewer steps must be taken to reach the target. However, too large of step sizes can result in overshoot or oscillations, decreasing the path efficiency. In the results presented here, however, this does not appear to be the case, as the path distances are not affected by the highway weighting parameter or number of steps, indicating the same path is being taken, just over a smaller number of steps. Based on this, it appears that higher highway weighting parameter values improve the efficiency of the path planning.

The path distance and number of steps performance metrics were also evaluated for each individual environmental configuration (Figure 3.6). In these figures, the mean statistics were evaluated over all successful simulations for any given highway point weighting and grid size configuration, as well as over trial configurations which were commonly successful for all weighting values. The comparison of performance metrics over commonly successful trials gives a better performance comparison, as it avoids skewing the statistics to be worse for parameters that achieved success on more difficult trials. These results show no clear trend between mean path distance and weighting parameter value, but for most environmental configurations and grid sizes, higher weighting parameter values corresponded to lower numbers of steps. Table 3.1 gives the performance statistics calculated over commonly successful simulations at all highway weighting parameter values.

3.4.2 - Real-Time Gazebo Simulations Comparing Traditional APF, APF w/Escape Force and APF-HW Results

The goal of the presented APF with highway method of robot navigation is twofold: (i) to prevent the occurrence of convergence to a local minimum before arriving at the target, and (ii) to guide the robot along a near-optimal path to the target, thus increasing the path efficiency compared to other APF-based path planning methods. The simulations presented in this section

were performed in the three indoor testing environments described in the previous section to evaluate the algorithm performance against other APF-based navigation methods in a variety of real-world scenarios. The performance metrics used to evaluate the performance of the algorithm were time to target, final position error, net velocity, path efficiency, and, perhaps most important, success rate. Time to target was calculated as the time for the simulation to terminate according to the convergence criteria defined in Section 3.3.3 - APF-HW Path Planning Experiments. The final position error was defined as the net distance between the robot position at simulation termination and the actual target location. It should be noted that this is deemed the least important of the evaluated performance metrics because it is highly dependent on the chosen convergence criteria, and thus could likely be improved by altering these parameters. The net velocity is calculated as the net distance traveled (i.e. straight-line distance between the start and end positions of the robot) divided by the total simulation time. Path efficiency is the total distance traveled by the robot divided by the straight-line distance between the start and end positions of the robot. Finally, success rate is the percentage of simulations that terminated successfully, defined as the robot being located within 1m of the target location.

The performance metrics for each algorithm for the AWS Small House World, AWS Bookstore World, and the Café world, are given in Table 3.2, Table 3.3, and Table 3.4, respectively. Besides success rate, all other performance metrics are calculated only over commonly successful simulations for all three algorithms to provide a better comparison of methods. In all simulation environments, the APF-HW method with variable-density highway point selection achieved the highest success rate, with success rates of 98.75%, 100.0%, and 98.0% in the Small House World, Bookstore World, and Café World, respectively. For all environments, the success rate of the APF-HW algorithm was significantly higher than the other two methods

using both methods of point selection, indicating that the APF-HW method is successful in decreasing the occurrence of convergence to local minima before arriving at the target.

Table 3.2. Small house world simulation results

Metric		APF Only	APF w/F_{esc}	APF-HW Adaptive	APF-HW Random
Time to Target (s)	Mean	15.628	17.032	**13.125**	12.759
	Median	**11.20**	13.45	11.375	11.425
Final Position Error (m)	Mean	0.299	0.272	0.278	0.255
	Median	0.289	0.274	0.277	0.232
Net Velocity (m/s)	Mean	0.520	0.495	0.568	**0.578**
	Median	0.555	0.518	0.580	**0.587**
Path Efficiency	Mean	0.830	0.792	0.844	**0.860**
	Median	0.854	0.806	**0.873**	0.872
Success Rate		0.775	0.825	**0.9875**	0.975

*Calculation of performance metrics is explained in the main text.

Table 3.3. Bookstore world simulation results

Metric		APF Only	APF w/F_{esc}	APF-HW Adaptive	APF-HW Random
Time to Target (s)	Mean	20.932	24.286	**13.391**	15.141
	Median	14.05	18.90	**11.25**	12.03
Final Position Error (m)	Mean	0.291	0.304	0.290	0.315
	Median	0.253	0.288	0.294	0.291
Net Velocity (m/s)	Mean	0.363	0.336	**0.464**	0.449
	Median	0.348	0.317	0.476	**0.478**
Path Efficiency	Mean	0.710	0.669	**0.779**	0.754
	Median	0.680	0.689	**0.823**	0.794
Success Rate		0.60	0.64	**1.00**	0.98

*Calculation of performance metrics is explained in the main text.

Table 3.4. Cafe world simulation results

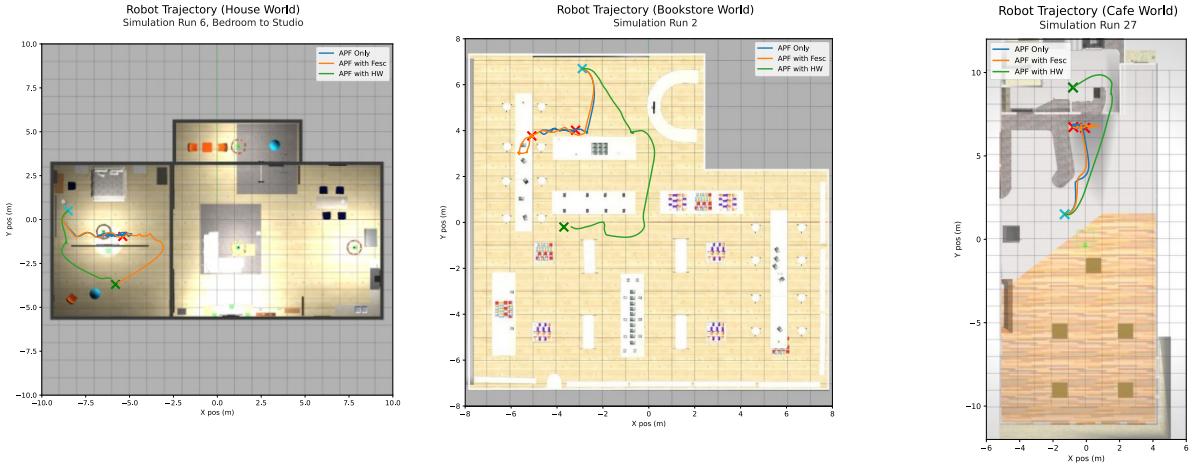
Metric		APF Only	APF w/F_{esc}	APF-HW Adaptive	APF-HW Random
Time to Target (s)	Mean	11.009	10.739	10.368	**10.208**
	Median	9.13	9.23	9.45	**8.625**
Final Position Error (m)	Mean	0.261	0.274	0.283	0.253
	Median	0.236	0.269	0.255	0.259
Net Velocity (m/s)	Mean	0.611	0.630	0.625	**0.637**
	Median	0.633	0.644	0.645	**0.657**
Path Efficiency	Mean	0.914	0.919	0.913	**0.927**
	Median	0.937	0.940	0.928	**0.948**
Success Rate		0.84	0.80	**0.98**	0.92

*Calculation of performance metrics is explained in the main text.

For the Small House World and the Bookstore World, the two methods of highway point selection for the APF-HW algorithm performed similarly, likely due to the fact that these environments have relatively consistent complexity throughout. However, the variable-density method of point selection achieved a significantly higher success rate (98% vs 92%) in the Cafe World, which has inconsistent complexity throughout the environment, with the main dining area being relatively low complexity, and the kitchen and passageway between the kitchen and dining room being much higher complexity, and thus requiring a higher density of highway points to ensure the presence of fully-connected highway paths throughout the full environment. However, there was still one failure in the Café World using the variable-density point selection method, caused when there was no highway path leading between the start and target position through the narrow corridor between the kitchen and the main dining area. While the variable-density point selection greatly improves the chances of having a full distribution of highway paths throughout the environment, narrow corridors like this still present the opportunity for separated areas

depending on the semi-random selection of points. A greater number of points would further increase the chances of a fully-connected highway network, but at the cost of increased computational complexity of the algorithm. Selection of the highway points is a trade-off between complexity, performance, and efficiency, and further exploration of alternate methods of point selection may improve algorithm performance beyond the presented results.

For the other performance metrics, the APF-HW method performed better than or equal to the other methods for all performance metrics, with the greatest improvements in performance coming from higher net velocities and path efficiencies. To show the difference in performance between the 3 methods, Figure 3.7 shows the trajectory for each of the methods for the same simulation. Here, we can see that the APF-HW method creates more efficient trajectories and avoids or navigates through local minima where the other algorithms get stuck.



a) b) c)

Figure 3.7. Example trajectories comparing the three tested methods for trials in each of the 3 environments. The start position is indicated by a blue ‘X’ and the target position is designated with a green ‘X’. Red ‘X’s denote final positions for failed trajectories. APF-HW trajectories shown used the variable-density method of selecting highway points.

Figure 3.8 shows the final position errors for the AWS Small House World, AWS Bookstore World, and Café World. The arrows in each figure point to the final location of the robot from the

actual target position to show where the robot was located when each simulation terminated. These figures show that many failures occurred for both the APF-only and APF with F_{esc} [39] simulations due to convergence to a local minimum, as can be seen from the presence of an obstacle directly between the robot position and the final target location. This likely occurred for the F_{esc} method as well due to areas of high repulsive force created by a high density of obstacles, and thus overlapping repulsive force influence areas. In the APF-HW method, this was addressed by the modified obstacle influence distance, and thus failures due to high density of obstacles are much less prevalent.

These results show that the APF-HW method out-performs the other methods, especially in more complex environments, where it achieved a much higher success rate, along with overall better performance. This method is able to successfully avoid convergence to local minima which result in navigation failures. It also guides robot navigation along more efficient paths, resulting in shorter times to target, higher net velocities, and higher path efficiencies. The simulations performed validate that the APF-HW method improves performance over existing APF-based navigation methods in a variety of real-world environments, with the greatest performance improvements being seen in more complex environments.

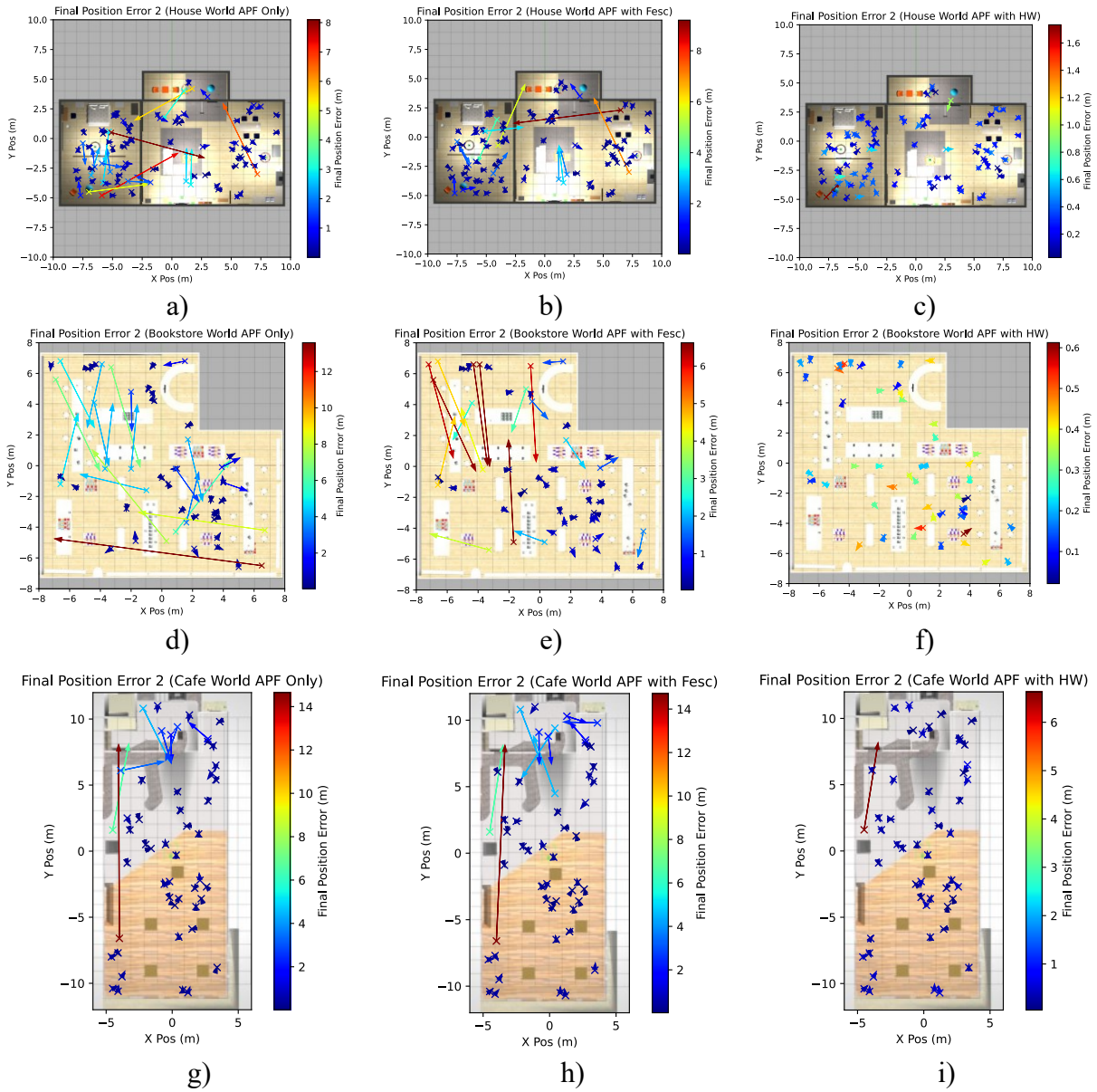


Figure 3.8. Final position errors for all trials using each navigation method (APF-only, APF with F_{esc}, and APF-HW) in each test world: AWS Small House World (a-c), AWS Bookstore World (d-f), and Café World (g-i). APF-HW results are shown using the variable-density point selection method. Arrows point from actual target position to the final robot position.

3.5 - Discussion

There are a number of different parameters and factors which affect the performance of the proposed APF-HW algorithm presented in this paper. In most cases, the optimal parameters are dependent on the environmental configuration, and thus an adaptive method of selecting

parameters, such as the one presented in this paper, is preferable to optimize performance of the algorithm. These parameters include grid size, highway weighting value, number of highway points, and the method of highway point selection. For the highway weighting parameter, it was found that a minimum value of 25-50 was required for consistent success in more complex environments, while values higher than this could create more efficient trajectories. In general, more complex environments require smaller grid sizes and more highway points to achieve good performance, but this comes at the cost of increased computational complexity both while initializing and running the algorithm. Because of this, the minimum parameter values that result in success in a given environment should be selected to minimize computational effort while maximizing other algorithm performance metrics. It is an expected result that more complex environments require more complexity in the algorithm to achieve success, and the experiments presented in this paper show the adaptability of the proposed algorithm to different environments through parameter adjustment. A recommendation for a generalized method of parameter selection for a given environment was presented, but further optimization could improve algorithm performance, either through trial and error of parameter selection for a specific case, or through development of a more sophisticated, machine learning-based parameter selection method.

The performance of the APF-HW algorithm is highly dependent on the number and location distribution of highway points which are used, and the optimal number for this parameter varies depending on the environment where the algorithm is deployed. Thus, determining the optimal number and distribution of points is important for the success of the algorithm. To this end, an adaptive algorithm for variable-density highway point selection was presented in this chapter. It was found that this variable-density method of highway point selection resulted in higher success rates than random selection of highway points in some cases, specifically when the complexity of

the environment varied locally. An example of this was seen with the Café World, where the section of the environment with the kitchen and hallway was much more difficult for the robot to navigate through than the main dining area of the café. Our results demonstrate that the use of variable-density highway point selection greatly improves the success rate of the algorithm in cases like this. As with the parameter selection, further research into machine learning-based methods of highway point selection may improve algorithm performance by optimizing the number and placement of highway points.

Chapter 4 – Navigation for Approach to Target (Integrated Control of Mobile Robot Arm)

4.1 - Introduction

Robotic arms are one of the most widely used type of robot. Their ability to perform manipulation tasks quickly, repeatably, and accurately, along with a variety configurations and compatibility with various end-effectors, allow them to be used in a wide range of applications, from manufacturing lines to surgical robots. In general, robotic arms are used in stationary applications, where the base is fixed in a certain location, and the task must be executed within the pre-determined working area of the robot. This is optimal for repetitive tasks requiring high repeatability, such as in manufacturing applications. However, when considering the expanding capabilities of robots as autonomous capabilities are improved, there are many potential applications of robotics where a mobile implementation of a robotic arm would prove advantageous. Autonomous navigation strategies for mobile robots have greatly improved, allowing robots to now navigate through complex and unknown environments. Combining these autonomous navigation capabilities with the manipulation abilities of robotic arms introduces many more potential application areas for robots in the future.

There are two methods to approach the control problem of a mobile robotic arm. The first method is to consider the control of the mobile robot and the control of the robotic arm separately. In this case, the first control problem is to navigate the mobile robot to the desired location for performing the manipulation task. Once the robot has reached this location, the mobile robot becomes stationary, and the second, separate control problem is that of the manipulation task of the robotic arm. If this method is chosen, existing control methods for both mobile robots and robotic arms can be employed, and the main difficulty is determining the desired base location of

the robotic arm to which the mobile robot must navigate in order to perform the manipulation task. The second method is to employ an integrated control strategy, in which the control of the mobile robot and the robotic arm are approached as a single problem, and the mobile robot and the robotic arm work together to execute the manipulation task. In this scenario, the mobile robot must still navigate to the desired location for performing the manipulation task, but once it is there, instead of staying stationary, the mobile capabilities of the robot, and thus the robotic arm base, are leveraged to provide desirable advantages during task execution when compared to a stationary robotic arm.

When considering the control problem of a mobile robotic arm, one of the first considerations is determining the optimal base position for the robotic arm. This is applicable to both separate and integrated control methods, because in both methods the base location of the robotic arm will have an impact on the execution performance of the task. For the separate control problem, the base location throughout the duration of the manipulation task execution will affect various performance metrics, such as achievable trajectory tracking accuracy and energy consumption. For the integrated control problem, the starting location of the robot base for the manipulation task will similarly affect the overall performance throughout the manipulation task. Thus, it is important to determine the optimal starting position (or end position of the mobile robot) for performing the manipulation task. However, the other factor which must be considered is the environment through which the mobile robot must navigate to reach this starting position. In more complex environments, there may be a large difference in difficulty for the mobile robot to reach different potential starting positions. For example, one starting position may require a much longer approach path, or navigation through narrower corridors with much more obstacle avoidance required than a different, possibly less optimal starting position. Because of this, both the optimality of the

starting position when considering the manipulation task and the difficulty of reaching this starting position must be considered when determining the overall optimal goal location and path for the mobile robotic arm. This complicates the mobile robot navigation task because, unlike most navigation scenarios, there is no longer a fixed goal location, but rather a number of potential goal locations with varying degrees of optimality.

The integrated control problem can be divided into three main tasks: (1) navigating the mobile robot to the optimal starting position for the integrated control task, (2) trajectory generation and control formulation for approach to the target object from this starting position, and (3) trajectory generation and control formulation for the execution of the desired task. This chapter proposes an approach to solve task (1) of the integrated control problem – navigating the robot to the optimal starting position for the integrated control task. This task is further broken down into the following sub-tasks, each of which is detailed in a section of this chapter – (i) formulate an objective function to determine the optimal starting mobile robot position for a given target, (ii) optimize and map the objective function within reachable space for the mobile robot using a grid-based method, (iii) use this objective function mapping to determine to optimal starting location of the 'approach' task (task 2), (iv) navigate the robot to this starting location. We introduce an APF-based method for robot navigation, where the potential field is modified to represent the distribution of potential goal locations weighted by their optimality for performing the desired manipulation task. This modified potential field is combined with the proposed APF-HW navigation method presented in Chapter 3 to achieve more efficient navigation to the starting location.

4.2 - Background

Robot arm trajectory generation has been extensively studied, and a comprehensive review of all related works here will not be given, as the focus of this paper is concentrated on mobile robot

navigation. However, in the context of a mobile robot arm, a major consideration is energy consumption of the robot throughout task execution. While this is a consideration for static robotic arm applications as well, it is especially consequential in the case of a mobile robot arm, which must have an on-board power source (i.e. a battery) with limited energy, and thus is it of utmost importance that energy be conserved as much as possible. Some research has been done regarding energy consumption minimization in robotic arm trajectory generation. Mohammed et al. studied the optimization of robot arm trajectories using inverse dynamics to minimize energy consumption and showed significant impacts of task location within the robotic arm workspace on the energy to perform a given task [44]. Mahdavian et al. also studied minimization of energy consumption in robotic arm trajectory generation with the added challenge of avoiding a moving obstacle using a genetic algorithm. This work showed significant improvements in energy consumption using the optimization over a standard linear motion approach [45]. Siar et al. also used genetic algorithm to address energy optimization in robotic arm trajectory generation by converting the optimal control problem to a nonlinear program and optimizing with a genetic algorithm [46]. These works demonstrate the importance of the location of the robot arm base with respect to the task being performed on the energy consumption of performing the task. This provides clear evidence that solving the optimization problem of determining the best robot base location presented in this paper can lead to significant energy savings, which is a very important consideration for mobile robot control.

4.3 - Methods

4.3.1 - Formulation of Objective Function

There are a variety of performance objectives that could be considered for this integrated control problem. In this chapter, the two most important objectives that were identified to evaluate

task execution were total energy consumption to perform task, and end-effector sensitivity of robot arm (both position and orientation). The objective function proposed in this section is a combination of these two metrics, with an adjustable weighting parameter for each depending on the relative importance of each metric for a given task.

4.3.1.1 - Energy consumption metric

In general terms, the energy consumption would be calculated as the total amount of energy consumed over the entire task execution. However, for Task 1, the goal is to determine the optimal starting location of the mobile robot of the task without calculating the entire task trajectory. This makes it impossible to determine the entire energy consumption of the system throughout the execution of the task, since the full trajectory of execution is not yet known. Instead, in this task, the energy consumption component of the objective function is calculated based on the holding torque of the robotic arm at the start of the task. There is no contribution to this component by the mobile robot because the system is taken to be at rest, and it is assumed that the mobile robot consumes minimal energy in a stationary state (and that this energy consumption is relatively independent of the robotic arm configuration). To calculate the holding torque of the robotic arm for a given mobile robot position (task start position), the joint positions must first be determined. This is done by computing the inverse kinematics of the robotic arm for the relative position $\vec{x}_p$ calculated by:

$$\vec{x}_p = \vec{x}_t - \vec{x}_r \tag{4.1}$$

Where $\vec{x}_t$ is the task start position vector (or target position of the end-effector) and $\vec{x}_r$ is the mobile robot position vector (or base position of the robotic arm).

The examples presented in this paper were all computed using the WidowX-6DOF robotic arm, which follows the standard 6DOF robotic arm configuration with 6 revolute joints, the first 3 of which control the arm position, and the last 3 of which control the end-effector orientation (wrist). For this robot configuration, the wrist and arm positions can be decomposed into 2 separate inverse kinematic problems, and the possible joint positions for a given end-effector configuration can be calculated analytically using the Denavit-Hartenberg (DH) parameters for the arm. Because of redundancy, any given end-effector position can have between 0 and 16 possible inverse kinematic solutions. Once the inverse kinematics of the relative position have been computed and the possible joint configurations are known, these can be used to find the holding torque for each joint using the Euler-Lagrange equations and the mass and inertial parameters for each link of the robotic arm.

Once the holding torque for each joint is calculated, this must be converted into a single value corresponding to the energy consumption contribution to the overall objective function value. One method for this would be to simply add the torque values for each joint together to give a cumulative torque score, which could then be used in the objective function. However, another consideration for this score is how close each individual joint is to the torque limit of its corresponding motor(s). It is necessary to prevent robotic arm configurations which cause any joints to exceed their torque limit, as this will cause the robotic arm to be unable to hold the desired positions, and to penalize configurations which too closely approach this limit. To achieve this objective, the torque value for each joint was scaled by an inverse function, so that the torque score for each joint was calculated as:

$$t_i = -|\tau_i| \left(\frac{a}{\min\big(|\tau_i|, (r_i - 0.01)\big) - r_i} + 1 \right) \tag{4.2}$$

Where τ_i is the torque for joint i, r_i is the joint torque limit for joint i, and a is a scaling parameter determining the shape of the torque score curve, as shown in Figure 4.1. This scaling prevents configurations where the torque of any joint is approaching or past the torque limit for that joint by causing the torque score to approach infinity as the required holding torque approaches the joint torque limit. Finally, the energy consumption contribution to the full objective function is calculated using the summation of the torque scores for each joint:

$$T = \sum_{i=1}^{n} t_i \tag{4.3}$$

Where n is the number of joints of the robotic arm.

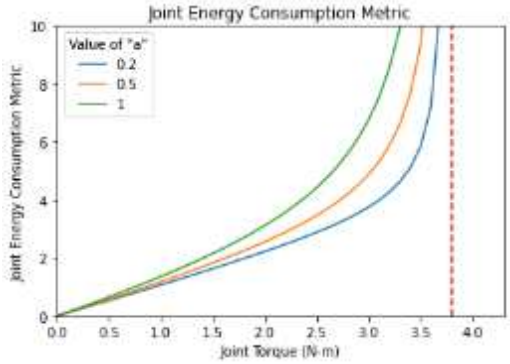


Figure 4.1. Joint energy consumption metric for different values of 'a'. The vertical red dashed line indicated the motor torque limit.

4.3.1.2 - End-effector Sensitivity Metric

The end-effector sensitivity metric is a measure of how sensitive the end-effector position and orientation are to individual joint movements or errors. This is an important metric because for tasks requiring high precision, the end-effector sensitivity will greatly impact the quality of task execution. In many cases, the required end-effector sensitivity will vary with direction, with high sensitivity required in some directions, while low sensitivity is desired in others. For example, a task requiring tracking along a surface requires low sensitivity normal to the surface to minimize

movement towards/away from the surface, but high sensitivity along the tracking vector, to ensure the arm can follow the desired trajectory. (Note that in this case, low sensitivity refers to small end-effector movement in a direction for a specified joint motion, while high sensitivity refers to large end-effector movement in a direction for the same amount of joint motion.) Additionally, the end-effector sensitivity metric is implemented to avoid singularities of the robotic arm, as these prevent the ability of the robotic arm to move in certain directions, which is undesirable in almost all cases.

The end-effector sensitivity metric has a number of parameters that can be adjusted based on the desired task performance – orientation of position sensitivity axes, relative sensitivity weighting along position axes, orientation of attitude sensitivity axes, relative sensitivity weighting along attitude axes. Both the position and attitude sensitivity scores are calculated using the Jacobian for the given robotic arm configuration. The Jacobian for the end-effector is calculated using the same joint configurations calculated in the inverse kinematics procedure given in the previous section. Once the Jacobian for the end-effector is calculated, a modified Jacobian is calculated for both the position and attitude sensitivity calculations by multiplying the Jacobian by the specified rotation matrix parameter for the respective sensitivity calculation and a diagonalized matrix of the normalized relative axis weighting vector for the respective sensitivity score ((4.3) and (4.4)). The position and attitude sensitivity scores can then be computed for each axis using the summation of the corresponding column of the modified Jacobian matrix, giving 6 individual sensitivity scores – 3 position scores along the desired positional sensitivity axes, and 3 attitude scores along the desired attitude sensitivity axes.

$$J_{v,new} = diag(\vec{w}_l) R_{rot,v} J_v \tag{4.4}$$

$$J_{\omega,new} = diag(\vec{w}_r) R_{rot,\omega} J_{\omega} \tag{4.5}$$

$$s_{p,i} = \sum_{j=1}^{n} \left| J_{v,new}(i,j) \right| \tag{4.6}$$

$$s_{r,i} = \sum_{j=1}^{n} \left| J_{\omega,new}(i,j) \right| \tag{4.7}$$

The final component of computing the end-effector sensitivity score is the singularity score. This component scales both the position and attitude sensitivity scores to prevent minimization of the objective function at a singularity. For the 6DOF decoupled arm-wrist robot configuration used in this paper, the robotic arm can experience either an arm or a wrist singularity. When the Jacobian is split as shown in (4.8), an arm singularity occurs when $\det(J_{11}) = 0$ and a wrist singularity occurs when $\det(J_{22}) = 0$. Avoidance of these configurations is achieved by introducing a singularity factor, calculated by (4.9), where *b* is a scaling parameter.

$$J = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{4.8}$$

$$\sigma = \frac{b}{|\det(J_{11}) \det(J_{22})|} + 1 \tag{4.9}$$

The final position sensitivity score and attitude sensitivity scores are calculated as given by (4.10) and (4.11) respectively.

$$S_p = \sigma \sqrt{s_{p,1}^2 + s_{p,2}^2 + s_{p,3}^2} \tag{4.10}$$

$$S_r = \sigma \sqrt{s_{r,1}^2 + s_{r,2}^2 + s_{r,3}^2} \tag{4.11}$$

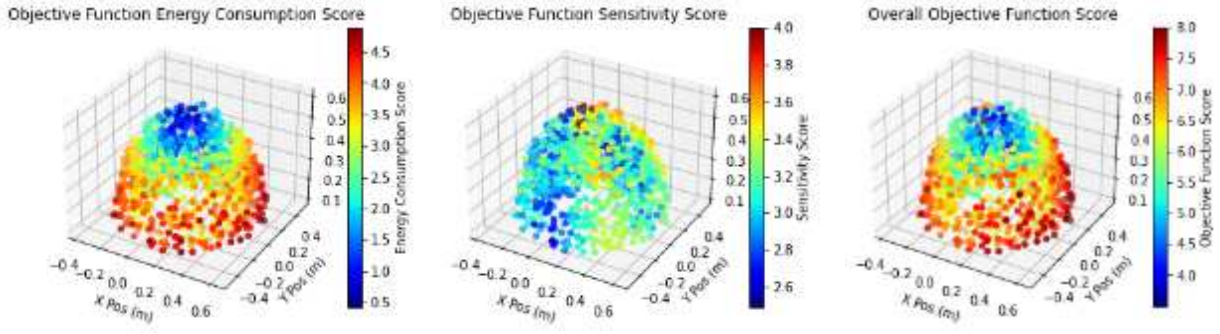
4.3.2 - Computing the Overall Objective Function

For each possible joint configuration of the relative robotic arm position, the energy consumption score, position sensitivity score, and attitude sensitivity scores are computed as given

in (4.3), (4.10), and (4.11), respectively. The final objective function value for each joint configuration is calculated using (4.12), where w_t, w_p, and w_r are the energy consumption, position sensitivity, and attitude sensitivity weighting parameters, respectively.

$$j = w_t T + w_p S_p + w_r S_r \tag{4.12}$$

These parameters can be adjusted based on the task requirements. Finally, the objective scores for all possible joint configurations for a given robotic arm position are compared, and the lowest score is taken as the objective function value for that position. It should be noted that this objective function value is only valid for the corresponding joint configuration associated with that value, and other joint configurations for the same robotic arm position have different objective function values. Example values of each component of the objective function and the overall objective function for different robot arm end effector positions are shown in Figure 4.2.



a) b) c)

Figure 4.2. Example scores for each portion of the objective function (a and b), and overall objective function score (c) for different end-effector positions

4.3.3 - Grid-based Mapping and Optimization of the Objective Function

In this section, a methodology is presented for creating a grid representation of the possible locations for the mobile robot, and using this grid to map and optimize the objection function to determine the optimal starting position of the mobile robot for a given integrated control task. A

grid-based approach was selected because the objective function is relatively costly to compute as it involves evaluation of the Euler-Lagrange equations, which can become quite complex for more complicated robotic arm configurations. Using a grid-based approach can greatly reduce the optimization search area by reducing the global optimization area from the entire possible area for the mobile robot to a number of grid squares selected based on objective function value at the center of the grid squares. Reducing the search area improves the speed of convergence of the optimization problem, while the use of a grid system provides a framework for mapping the objective function values throughout the workspace, which becomes very useful when minimization of the objective function is not the only control objective.

A grid representation of robot space can be constructed by applying an offset to all obstacles so the robot can be considered as a point. Each grid square is then designated as either a free square, where the robot can go, or an obstacle square, which the robot must avoid. A square overlapping any amount of obstacle space must be designated as an obstacle square to ensure the robot can move to any location in a free square while avoiding collisions. The goal of creating the grid is to maximize the amount of free space covered by free squares to create the most accurate grid representation of the usable space in the environment. This is accomplished using three parameters – column offset, column orientation, and height optimization within columns. Column offset refers to the offset of the first column from the left side of the floorplan map, given in units of column width (or grid size), and column orientation refers to the angle at which the columns are constructed. Height optimization within columns determines a unique vertical offset for the grid squares within each column to optimized the number of free space grid squares in that column. A 2-variable global optimization is performed to determine the orientation (between 0° and 90°) and offset (between 0 and 1) which result in the highest percentage of free space area coverage by the

free squares. The differential evolution optimization algorithm from the SciPy Optimize library was used to perform this optimization.

4.3.3.1 - Objective Function Mapping in Grid Space

Once the free space for the mobile robot has been transformed into grid space, this grid representation of the robot space can be used to map the objective function over the working area of the mobile robot. For each grid square, the objective function value can easily be calculated using the center of each grid square as the location of the mobile robot and robot arm base. In some cases, there will not be a solution to the inverse kinematics problem for the relative position between the grid center and the desired target location. In this case, the objective function cannot be computed, and the square is not a valid solution. The mapping procedure can be streamlined by only computing the inverse kinematics and objective function for free squares that are less than or equal to the maximum robot arm extension from the target location, since grid squares located further than this distance from the target location are guaranteed to have no inverse kinematics solution. For cases where there are multiple inverse kinematic solutions corresponding to multiple objective function values, the minimum objective function value should be selected as the value for that grid square. The quality of the mapping is very dependent on the grid resolution. A finer grid resolution will cover more of the possible free area and provide higher quality mapping, at the cost of more calculations and higher computational effort.

4.3.4 - Determining Optimal Approach Starting Position Using APF

When implementing integrated control of a mobile robot arm for a designated task, optimization of the objective function at the starting position of the task is not the only objective that should be considered. In fact, the first step in performing a designated control task is to navigate the mobile robot to the vicinity of the target to set up for the integrated control task. It is

likely that a certain trajectory will need to be taken by the robotic arm while approaching the target to accomplish objectives such as obstacle avoidance and approach from a specified direction. Because of this, it is desirable to start this approach integrated control task before the mobile robot has arrived at the final starting position for the task. To address this constraint, we introduce the idea of an 'approach bubble.' The 'approach bubble' corresponds to a region around the final starting position of the integrated manipulation task. Prior to entering the 'approach bubble', the mobile robot can navigate through the environment with the robotic arm in its collapsed home position. Once within the 'approach bubble', Task 2 of using integrated control to approach the target object subject to additional constraints is started. In this chapter, we address the problem of navigating the robot to the perimeter of the approach bubble and determining the optimal location for entering this 'approach bubble' to initiate integrated control Task 2.

When considering navigation to the perimeter of the 'approach bubble', the feasibility and difficulty of navigating the robot to the chosen location should also be considered. It may occur that the optimized entry location determined solely from the objective function does not end up as the fully optimized entry point when considering the path of the mobile robot to this location as well. This location may be unreachable, or very difficult to reach, depending on the environment configuration and the starting location of the mobile robot. In this case, there may be a different starting location, corresponding to a higher objective function value, that is more optimal when considering the entire control problem of navigating the mobile robot to that location. Furthermore, even when considering a single optimal starting location for the integrated control task, the perimeter of the 'approach bubble' creates a circle of possible entry locations, all of which correspond to the same task starting location. In this section, we present a method of using a modified APF approach to plan a path for the robot to an entry point to the 'approach bubble'.

4.3.4.1 - APF Path Planning with Distributed Target Force

APF is a method of robot path planning where the robot environment is represented by a potential field created by the target location and obstacles. For the integrated control problem, the APF can be modified to create a distributed attractive force, constructed from the objective function mapping. This is advantageous because a potential field can be created that represents the variations in optimality in starting position which, combined with representations of the overall environment, allows for convergence to the overall optimal entry position when considering the full robot path to all possible entry positions. This distributed potential field is created by treating each mapped grid square with a feasible objective function as a target exerting attractive force. The force exerted from each of these attractive squares is then scaled by the value of the objective function, such that squares with lower objective function values exert higher attractive force, (4.13). The overall potential field is created by summing the potential field generated by each attractive square, along with the potential fields created by the obstacles in the environment. This final potential field creates a holistic representation of the environment, considering the integrated control problem with the robot environment, and the path planning problem can be solved like a traditional APF problem using steepest gradient descent on the modified APF.

$$F_{att,i}(q) = -\nabla U_{att,i}(q) = -\frac{1}{j_i}\zeta d(q, q_{goal,i})\nabla d(q, q_{goal,i}) \tag{4.13}$$

4.3.4.2 - Modified Attractive Potential Function to 'Approach Bubble' Perimeter

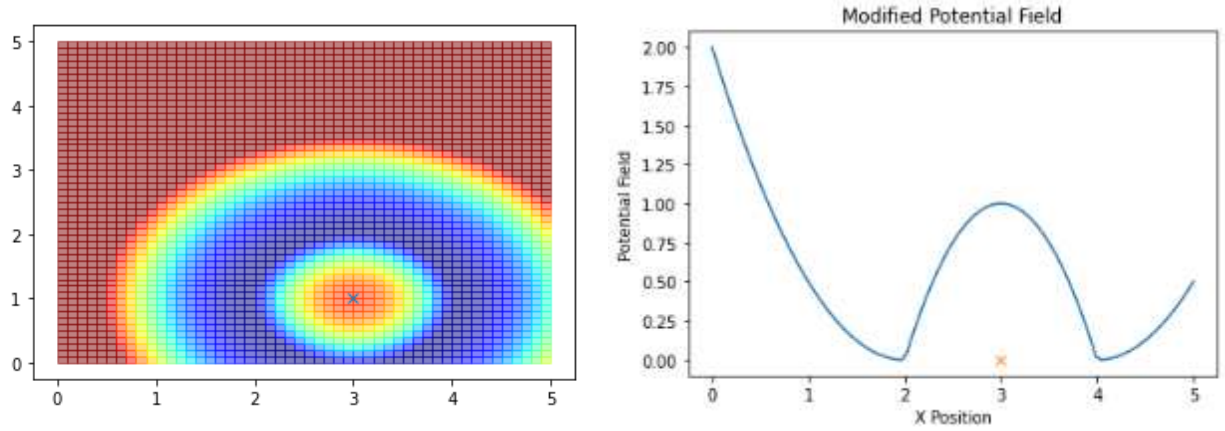
A further consideration for modifying the APF method to solve for the optimal entry location to the 'approach bubble' is that the final converged position should be located a distance equal to the radius of the 'approach bubble' away from the final starting position for the task. To accomplish this, the potential function for the attractive force must be modified such that the location of lowest potential is located at a distance *r* from the final starting position, where *r* is the

approach radius. Any locations within this radius will be at a value of higher potential to prevent the APF path planning from converging to an entry position located within the approach bubble. The modified attractive potential function and force equations to create this potential field configuration are given in (4.14) and (4.15).

$$U_{att,mod} = \begin{cases} \frac{1}{2}\zeta_1\left(\|q - q_{goal}\| - r\right)^2, if\ \|q - q_{goal}\| \geq r \\ -\frac{1}{2}\zeta_2\left(\|q - q_{goal}\|^2 - r^2\right), otherwise \end{cases} \tag{4.14}$$

$$F_{att,mod} = \begin{cases} -\zeta_1\left(\|q - q_{goal}\| - r\right)(q - q_{goal}), if\ \|q - q_{goal}\| \geq r \\ \zeta_2(q - q_{goal}), otherwise \end{cases} \tag{4.15}$$

These equations can be applied to the distributed attractive force model described in the previous section, and the final potential field is created by summing the modified potential field generated from each attractive square and the repulsive fields generated by any obstacles in the environment. Steepest gradient descent performed on this final potential field results in convergence to the optimal entry position to an 'approach bubble' when considering the distribution of objective function values along with the configuration of the environment and the robot's starting position.

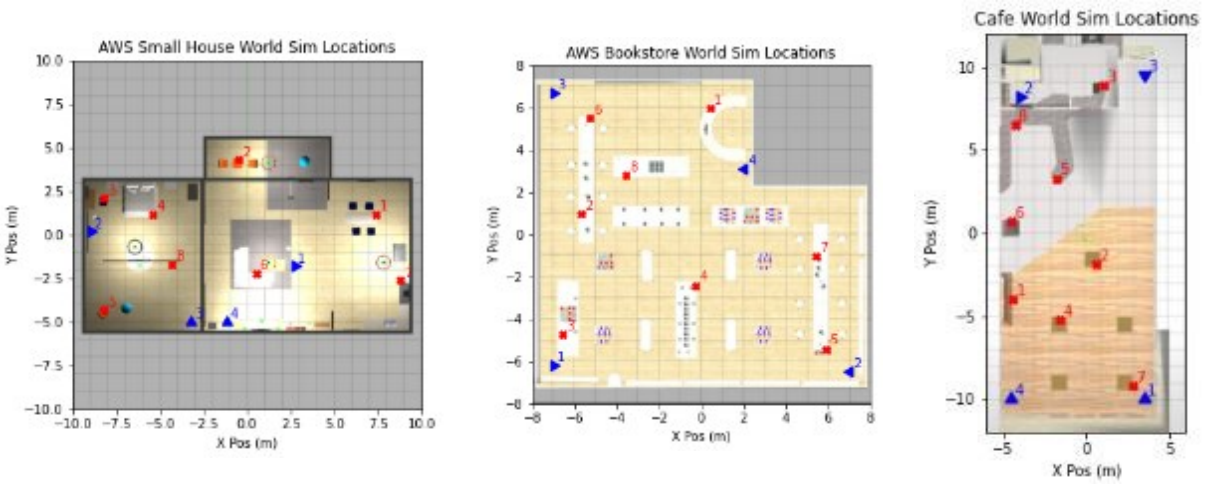


a) b)

Figure 4.3. Modified artificial potential field for 'approach bubble'

4.3.5 - Real-Time Gazebo Simulations Testing APF-HW Integrated Control

A set of experiments was performed to test the performance of the APF integrated control algorithm with the APF-HW algorithm presented in Chapter 3, and to examine the effect of the number of highway points on the algorithm performance in different environments, and compare random highway point selection to the adaptive point selection method presented in Chapter 3. For these experiments, simulations were run through ROS in Gazebo using three different worlds (or environments) – AWS Small House World, AWS Bookstore World, and Café World. For each environment, 4 starting locations and 8 target locations were chosen. When feasible, target locations were chosen to be in obstacle space to test the integrated control portion of the algorithm. Figure 4.4 shows the starting and target locations for each environment. For each world, two simulations were run for every combination of starting and target locations, for a total of 64 simulations. Additionally, different quantities of minimum and maximum highway points were chosen for each environment, and all 64 simulations were run for each set of these parameters.



a) b) c)

Figure 4.4. Starting and target positions for AWS Small House World (a), AWS Bookstore World (b), and Café World (c) simulations. Blue triangles are starting positions pointing in the direction of start orientation. Red X's are target positions.

The ROS2 and Gazebo infrastructure was the same for these experiments as described in the previous chapter. The implementation was the same except as noted below, corresponding to the addition of integrated control. Some notes on the implementation of the algorithm for these simulations.

- Attractive force calculation – For all simulations, the attractive force was calculated using the modified attractive force equation given by (4.15) for each attractive grid square, as determined by the presence of a feasible objective function at the center of that grid square. An attractive force vector was calculated using this equation for each attractive grid square, and the final attractive force vector was created by summing all of the individual attractive force vectors. If there were no attractive squares for the estimated target location, a single attractive force vector was calculated from the estimated target position using (4.15), where q_{goal} corresponded to the actual goal position.

4.4 - Results

The performance statistics for the APF with Highway Integrated Control simulations are given in Table 4.1, Table 4.2, and Table 4.3 for the AWS Small House World, AWS Bookstore World, and Café World, respectively. As with the previous experiments, performance metrics were calculated over only the commonly successful simulations for each world. Simulations were considered successful if the final robot position was located within 1.5m of an attractive square for the given target location. This metric was chosen based on an approach radius of 1m, combined with the mean final position error of ~0.5m observed in the previous trials, due to convergence criteria. Additionally, if a target position had no corresponding attractive squares, the simulation was considered successful if the final robot position was within 2.0m of the target location.

Table 4.1. AWS Small House World APF-HW/IC simulation results

Method of Point Selection		**Random**				**Adaptive**
Number of HW Points		**25-40**	**50-75**	**75-100**	**200**	**92**
Time to Target (s)	**Mean**	43.227	33.276	27.093	50.406	**18.717**
	Median	31.31	25.36	24.89	32.55	**13.98**
Time to 0.1m of Target (s)	**Mean**	39.721	31.613	23.576	40.626	**18.179**
	Median	28.11	24.57	20.39	28.30	**13.55**
Time to 0.2m of Target (s)	**Mean**	38.149	30.166	21.87	36.401	**17.460**
	Median	26.05	23.30	18.59	26.45	**13.15**
Final Distance to Target (m)	**Mean**	1.284	1.275	1.269	1.232	1.465
	Median	1.245	1.238	1.224	1.159	1.416
Net Velocity (m/s)	**Mean**	0.276	0.323	0.344	0.248	**0.498**
	Median	0.278	0.333	0.357	0.239	**0.506**
Path Efficiency	**Mean**	0.774	0.799	0.841	0.760	**0.842**
	Median	0.735	0.836	0.830	0.719	**0.876**
Success Rate		0.7813	0.7813	0.875	**0.953**	0.9375

Table 4.2. AWS Bookstore World APF-HW/IC simulation results

Method of Point Selection		**Random**			**Adaptive**
Number of HW Points		**35-60**	**75-100**	**200**	**276**
Time to Target (s)	**Mean**	39.639	38.652	34.659	**18.518**
	Median	39.85	31.75	29.00	**18.45**
Time to 0.1m of Target (s)	**Mean**	35.283	32.576	31.845	**17.302**
	Median	34.85	29.85	27.90	**17.85**
Time to 0.2m of Target (s)	**Mean**	33.714	31.256	30.588	**16.915**
	Median	34.05	27.65	27.15	**17.15**
Final Position Err (m)	**Mean**	1.331	1.343	1.320	1.460
	Median	1.371	1.351	1.319	1.469
Net Velocity (m/s)	**Mean**	0.239	0.257	0.283	**0.438**
	Median	0.241	0.282	0.305	**0.441**
Path Efficiency	**Mean**	0.732	0.763	0.768	**0.803**
	Median	0.724	0.768	0.762	**0.805**
Success Rate		0.8125	0.875	**0.969**	0.906

Table 4.3. Cafe World APF-HW/IC simulation results

Method of Point Selection		**Random**			**Adaptive**
Number of HW Points		**50-75**	**75-100**	**200**	**79**
Time to Target (s)	**Mean**	30.02	42.973	34.466	**18.379**
	Median	20.80	25.30	25.30	**17.80**
Time to 0.1m of Target (s)	**Mean**	28.474	38.757	32.609	**17.257**
	Median	20.45	24.95	22.75	**17.40**
Time to 0.2m of Target (s)	**Mean**	27.314	36.886	31.093	**16.751**
	Median	20.35	23.2	22.20	**17.00**
Final Position Err (m)	**Mean**	1.330	1.257	1.259	1.237
	Median	1.412	1.284	1.288	1.200
Net Velocity (m/s)	**Mean**	0.424	0.350	0.362	**0.551**
	Median	0.485	0.366	0.369	**0.630**
Path Efficiency	**Mean**	0.882	0.842	0.864	**0.937**
	Median	0.916	0.920	0.937	**0.971**
Success Rate		0.672	0.828	0.875	**0.923**

For all testing environments, there was a clear correlation between an increasing number of highway points and a higher success rate when using random point selection. This makes sense, as a larger number of highway points increases the chances of having a highway path between any starting and target locations, thus improving the chances of successful navigation to the target location via the highway points. The APF-HW/IC algorithm achieved 95.3% and 96.9% success rates for the Small House World and the Bookstore World simulations, respectively, with 200 HW points. However, the algorithm only achieved an 87.5% success rate for the Café World with the same number of HW points, indicating that even more HW points may still be able to significantly improve the success rate. Alternatively, for the Small House World and Café World, the adaptive point selection method was able to achieve higher success rates (93.75% and 92.3%) with far fewer

highway points than the random selection method. This did not hold true for the Bookstore World, where the adaptive point selection method only achieved a success rate of 90.6% despite having the highest number of HW points tested in that environment. This is likely because the adaptive point selection method provides the most benefit in environments of varying complexity, such as the Small House World and Café World, but does not provide as much benefit in more consistently complex environments, such as the Bookstore world, where large numbers of points are required for good performance regardless of the distribution throughout the environment.

Interestingly, for the other performance metrics, the highest number of HW points did not correspond to the best performance. The metrics presented in Table 4.1 - Table 4.3 are the same as those used in the previous chapter, with the addition of Time to 0.1m of Target, and Time to 0.2m of Target. These correspond to the time it took for robot to navigate to within 0.1m and 0.2m of the final robot position, respectively. These metrics were added to investigate the amount of time spent near the final position before the convergence criteria were met, as difficulty was observed reaching final convergence in some simulations. The chosen performance metrics indicate different optimal numbers of HW points based on the complexity of the environment. In the case of the Café World, a relatively simple environment for those simulations not going between the main area and the kitchen (which was true for nearly all commonly successful simulations), the best performance was achieved with the smallest number of HW points. In a medium complexity environment, the Small House World, an intermediate number of HW points achieved the best performance, with 75-100 outperforming the other configurations on nearly all performance metrics (except success rate as discussed above). Finally, the most complex world, the Bookstore World, needed the highest number of HW points to achieve the best performance. From these results, it is clear that there is not one optimal number of HW points to achieve optimal

performance, but rather it is dependent on the configuration and complexity of the environment. However, since the success rate was almost always improved with an increase in HW points, there is clearly a trade-off between other measures of performance and the success rate. While in the most complex environments, increasing HW points improves all of these metrics, for simpler environments, an increase in success rate may come at the trade-off of decreased performance measured by other metrics.

The adaptive point selection method had a significant impact on most of the performance statistics in all environments. This method consistently generated faster times to target and more efficient paths than the random point selection method. Despite not increasing the success rate of the simulations in all cases, this improvement in other performance metrics with comparable (or better) success rates to higher numbers of randomly selected points indicates the adaptive method of point selection generates higher quality navigation trajectories than the random selection method.

4.5 - Discussion

The integrated control portion of the algorithm validated the distributed attractive force model in its ability to guide the robot along the overall optimal path when considering both the difficulty of the path and the optimality of a given target location. This is a novel capability, since autonomous navigation and path planning algorithms typically consider a specific or set of specific targets, rather than a distribution of potential target locations. Various trials demonstrated the advantage of this algorithm by converging to different final locations depending on the starting location of the robot – proving that the algorithm balances the optimality of the target position with the difficulty and length of the path to each potential target location. The importance of the location of a manipulation task within the working space of a robotic arm has already been

demonstrated with regard to various performance metrics, such as energy consumption, and thus optimization of the base location of the robotic arm for a mobile robot task can greatly improve overall task performance. Because energy conservation is so important in mobile robotic applications due to the use of an on-board power source, the task must be considered holistically including both the energy expenditure of navigating to the location for the manipulation task, and the energy expenditure to perform the manipulation task. The proposed algorithm elegantly considers both of these task objectives and provides the best solution to minimize overall energy expenditure.

Chapter 5 – Real-time Trajectory Generation for Mobile Robot Arm Approach to Target[2]

5.1 - Introduction

As described in previous chapters, there are many advantages to an integrated control approach for a mobile robot arm. However, the trajectory generation problem for an integrated control mobile robot arm becomes more complex than for a stationary robot arm because the number of control variables increases (the mobile base wheel joints become additional control variables) and there is no longer a closed form solution for the robot arm joint positions for a given end-effector configuration due to the potential for different base locations. In this chapter, we consider the problem of real-time trajectory generation for a differential drive mobile robot arm.

Robot arm trajectory generation traditionally refers to determining the required joint positions, velocities, and accelerations required for the end-effector of the robotic arm to follow a path specified in Cartesian-space. In this case, there is a specific desired position for the end-effector at any given time during the execution of the trajectory. However, another application of trajectory generation which becomes relevant when considering a mobile robot arm is to move the robot arm and mobile base into the desired starting positions for the execution of a task. We refer to this as the approach trajectory where there is not a specific desired path which must be followed, but rather only a desired final configuration of the end-effector at the termination of the trajectory execution. Because of this, the mobile robot arm approach trajectory generation problem becomes

[2] Publication info: K. Weinmann and S. Simske, "Online trajectory generation for mobile robot arm using many-objective optimization," in 2024 IEEE International Conference on Advanced Intelligent Mechatronics (AIM), Boston, MA, 2024.

a fundamentally different problem from traditional robot arm trajectory generation because the specific desired trajectory is not identified. Another consideration is that the specific target position and orientation, environment configuration surrounding the target (i.e. obstacles in potential robot space), and even the starting mobile base position, may not be known *a priori*, but rather determined in real-time via sensor data. Thus, an online trajectory generation (OTG) solution is required to achieve the objective of moving the robot arm into the starting position for the desired task.

In this chapter, we develop a many-objective optimization (MOO)-based method for OTG which uses an integrated control approach, combining, if appropriate, movement of a differential drive mobile base with that of the robotic arm for optimal approach trajectory generation for a given task. Due to the complexity of the mobile robot arm integrated control problem, we focus on the objective of generating the approach trajectory (moving to a specified target end-effector configuration, rather than following a specified time-space path) for a mobile robot arm assuming an obstacle-free environment.

5.2 - Background

5.2.1 - Mobile Robot Arm

Some research has been done on the control of a mobile robot arm, but much of this research addresses the mobile robot and robotic arm control problems separately. Belda et al. investigated the use of model predictive control for a mobile robot arm [47]. Their work focused on the development of kinematic and dynamic models for a 5DOF robotic arm, which were used to develop a model predictive control system, but did not address the mobile robot aspect of the control problem. Widhiada et al. used PID control for a mobile robot arm, first modeling the mobile robot in Simulink, then building a physical prototype with motion control from a user via Bluetooth

[48]. Seddaoui et al. used the NSGA-II algorithm for many-objective optimization (MOO) to generate offline trajectories for a space robot, consisting of a robotic arm and a mobile base with 6 degrees of freedom (DOF) [49]. In their work, the target end-effector configuration was specified only by position versus the complete position and orientation configuration addressed in this chapter. However, this work did address the control problem of coordinated base and arm motions, albeit with a 6DOF base. Fotouhi et al addressed both mobile robot and manipulator trajectory planning in the context of a mobile robot arm [50]. The proposed differential drive trajectory planning algorithm addressed path planning in 3D (i.e. over uneven surfaces) and accounted for dynamic obstacle avoidance. Manipulator path planning compared planning in Cartesian space using inverse kinematics and in joint space, finding joint space planning superior due to singularities in the inverse kinematics of the manipulator. However, this research addressed the mobile and arm trajectory planning separately, as opposed to the integrated control approach explored in this chapter.

5.2.2 - Robotic Arm Online Trajectory Generation

Robotic arm trajectory generation can become a very computationally intensive task, especially for high-DOF manipulators or when considering trajectories subject to many constraints or objectives. Additionally, in many real-world applications, trajectories must be generated online in real-time to allow for adaptation to sensor readings, obstacles, and other factors. This need for online trajectory generation introduces the need for computationally efficient methods of trajectory generation which are still able to satisfy the relevant constraints. Various approaches have been proposed to provide a solution for the online trajectory generation problem. Ramos et al. developed a method of trajectory generation using closed-form equations to generate synchronized joint movements [51]. The closed-form implementation minimizes the computation required to generate

trajectories and thus allows for online implementation. Katzscmann et al. proposed a velocity-based online trajectory generation method which considers robot dynamics to ensure torque limit constraints are satisfied using a one-dimensional representation of the target path [52]. Zhao et al. used a two-step method, first generating time-optimal straight-line trajectories between via points along the path, and then smoothing the corners between path segments to create a smooth trajectory [53]. A visual servo-based implementation of online trajectory generation was presented in [54]. Another method for online trajectory generation uses a distribution function to compute trajectories [55].

More recent research on OTG increases the complexity of the trajectories which can be generated. The additional of jerk constraints is included in the work of Wang et al. on synchronous trajectory generation [56]. Wall et al. provide a method for OTG for a 3DOF robotic arm in an environment with many obstacles. Their method focuses on reducing the increase in complexity which accompanies an increasing number of obstacles in the environment, thus allowing OTG even in the presence of many obstacles [57]. A table tennis playing robot implementing OTG was presented in [58]. Colan et al. developed an optimization-based method of OTG for a surgical robot used for endonasal surgery [59].

5.2.3 - Many Objective Optimization (MOO)

Multi- and many-objective optimization is an approach for solving complex problems with multiple objectives. It has been used for a variety of real-world problems from public transport system optimization [60] to job shop scheduling [61] and many others. MOO has also been used for robotic arm trajectory generation. Malik et al. used a combination of swarm intelligence with a “product of exponentials” formulation of robot kinematics for trajectory generation for a 7DOF

robotic arm [62]. Most many-objective optimization algorithms are based on evolutionary algorithms, and a variety of approaches have been developed [63].

5.3 - Methods

We propose a many-objective optimization method for online approach trajectory generation to a target end-effector configuration for a mobile robot arm. The mobile base is a differential drive wheeled robot, and the robotic arm is a standard industrial 6DOF configuration. For the optimization problem, the solution vector is defined as the desired joint velocity vector at the next controller update, resulting in 7 solution variables – 2 joint velocities for the differential drive robot (left and right wheels) and 5 joint velocities for the robotic arm (the 6th joint of the robotic arm is held stationary, as explained below). The optimization is implemented in real-time, where a new goal joint velocity vector is generated at each time step from the optimization and passed to the individual joint controllers. To maximize the efficiency of the optimization so that it can be reasonably implemented in real-time, many of the objective functions use linearization to minimize the required number of computations, and then variables are updated at each time step between optimizations. For each time step optimization, a vector of solutions is returned, and a specific solution must be chosen from this vector for use. Compromise programming [64] was used to select the final solution from the vector of potential solutions, and the optimal weighting vector was determined experimentally.

5.3.1 - Defining the Target End-effector Configuration

The tool-tip configuration of a robotic arm end-effector is typically defined in Cartesian space using 6 variables – X, Y, and Z for the position and roll, pitch, and yaw (RPY) angles to describe the orientation. However, RPY is not the most intuitive method when defining a target configuration for a specific task on a surface. In practice, it is generally desired for the tool-tip to

be positioned normal to the specified surface, and the orientation around this normal vector may be arbitrary. To this end, the target orientation of the tool-tip was defined using a vector normal to the desired surface, and the position of the wrist rotate joint (controlling the orientation around this vector) was set to 0. In the case where a specific wrist rotation is desired, the necessary position of that joint can be calculated using the other joint positions determined from the optimization. The target position of the end-effector was defined using the typical XYZ Cartesian coordinates.

5.3.2 - Estimating the New Robot State

All of the objective functions are formulated using the estimated robot state at the next time step for a given joint velocity solution vector. The joint velocities for the time step are approximated as the mean of the current joint velocities ($\dot{q}_t$) and the next joint velocity command (solution vector, $\dot{q}_c$):

$$\dot{q} = \frac{\dot{q}_t + \dot{q}_c}{2} \tag{5.1}$$

5.3.3 - Joint Positions

The new joint positions are estimated by integrating the joint velocities over the time step:

$$\vec{q}_{t+1} = \vec{q}_t + \dot{q}dt \tag{5.2}$$

5.3.4 - Mobile Robot Base Configuration

The new mobile robot base configuration is estimated using the Jacobians for a differential drive mobile robot ($J_{v,b}$, and $J_{\omega,b}$). These Jacobians must be transformed from the mobile base frame to the global frame using the rotation matrix of the mobile base (5.3), where θ_b is the current base rotation. The new estimated base position and orientation are then calculated with (5.4) and (5.5), respectively.

$$R_{base} = \begin{bmatrix} \cos(\theta_b) & -\sin(\theta_b) & 0 \\ \sin(\theta_b) & \cos(\theta_b) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

$$\vec{x}_{b,t+1} = \vec{x}_{b,t} + R_{base} J_{v,b} \dot{q} dt \tag{5.4}$$

$$\vec{k}_{b,t+1} = \vec{k}_{b,t} + R_{base} J_{\omega,b} \dot{q} dt \tag{5.5}$$

5.3.5 - Tool-tip Configuration

As with the mobile robot base, the new robotic arm tool-tip position is estimated using the Jacobians ($J_{v,b,tt}, J_{v,a,tt}$), comprised of contributions from the mobile base wheel joints and the robotic arm joints. The position Jacobian of the tool-tip for the differential drive robot is given by (5.6), where d_{tt} is the vector from the center of rotation of the mobile base to the end-effector. These must then be transformed into the global frame using (5.3), giving the final tool-tip Jacobian of (5.7), and the new tool-tip position is estimated using (5.8).

$$J_{v,b,tt} = \begin{bmatrix} d_{tt}[1] * \frac{r}{w} + \frac{r}{2} & -d_{tt}[1] * \frac{r}{w} + \frac{r}{2} \\ -d_{tt}[0] * \frac{r}{w} & d_{tt}[0] * \frac{r}{w} \\ 0 & 0 \end{bmatrix} \tag{5.6}$$

$$J_{v,tt} = \begin{bmatrix} R_{base} J_{v,b,tt} \\ R_{base} J_{v,a,tt} \end{bmatrix} \tag{5.7}$$

$$\vec{x}_{tt,t+1} = \vec{x}_{tt,t} + J_{v,tt} \dot{q} dt \tag{5.8}$$

The orientation Jacobians ($J_{\omega,b,tt}, J_{\omega,a,tt}$) must also be transformed into the global frame using (5.3), giving the final orientation Jacobian of (5.9).

$$J_{\omega,tt} = \begin{bmatrix} R_{base} J_{\omega,b,tt} \\ R_{base} J_{\omega,a,tt} \end{bmatrix} \tag{5.9}$$

The orientation Jacobian represents the angular velocity of the tool-tip per joint motion, so the angular velocity of the tool-tip for the given time step can be calculated as (5.10).

$$\omega_{tt} = J_{\omega,tt}\dot{q} \tag{5.10}$$

This angular velocity is then represented as a rotation of an angle equal to the magnitude of the angular velocity vector around an axis equal to a unit vector in the direction of the angular velocity vector ((5.11) and (5.12)).

$$\theta_{tt} = \|\omega_{tt}\|dt \tag{5.11}$$

$$\widehat{\omega}_{tt} = \frac{\omega_{tt}}{\|\omega_{tt}\|} \tag{5.12}$$

The new X-vector of the tool-tip ($\vec{k}_{tt,t+1}$) is estimated from this rotation using (5.13).

$$\vec{k}_{tt,t+1} = \vec{k}_{tt,t}\cos(\theta_{tt}) + \left(\widehat{w} \times \vec{k}_{tt,t}\right)\sin(\theta_{tt}) + \widehat{w}\left(\widehat{w} \cdot \vec{k}_{tt,t}\right)(1 - \cos(\theta_{tt})) \tag{5.13}$$

5.3.6 - Overall Algorithm Structure

To improve the robustness of the real-time approach trajectory generation method, the trajectory generation problem was divided into 3 different cases (implemented at different times during the trajectory generation based on the criteria described below) – normal approach trajectory generation, initial approach trajectory generation, and escape from a local minimum (Figure 5.1). The two latter cases seek to mitigate some shortcomings associated with OTG. The initial approach trajectory generation case addresses the scenario where the robotic arm is located in a direction opposite that of the desired surface normal at the initialization of the trajectory. This can result in the end-effector approaching the surface from the wrong side (i.e. from below the surface), which is undesirable and can result in inefficient trajectories. The goal of this optimization is to navigate the mobile base of the robot to a starting location in an appropriate direction from the target end-effector location given the desired surface normal vector before beginning the normal approach trajectory generation towards the target end-effector configuration.

Finally, due to the real-time trajectory generation approach and unknown final joint configurations, it is possible for the trajectory generation to move the robot into a local minimum configuration from which it can no longer move closer to the desired target configuration (often due to reaching joint limits). If this scenario is detected, the algorithm switches to an optimization designed to extricate the robot from this local minimum configuration and return to a more neutral configuration from which it can successfully reach the target configuration. At each time step, the algorithm optimizes the desired joint velocity vector using the appropriate optimization problem formulation for that time step. Each of the three problem formulations are described in detail in the following sections.

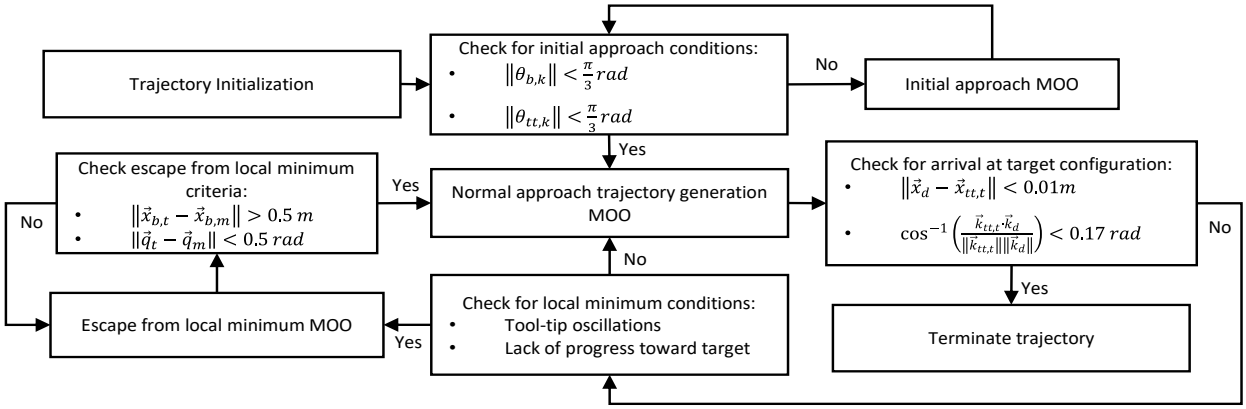


Figure 5.1. Flowchart of trajectory generation MOO algorithm structure.

5.3.7 - Problem 1: Normal Approach Trajectory Generation

The goal of the normal approach trajectory generation problem is to navigate the mobile robotic arm into the desired target end-effector configuration as quickly and efficiently as possible. This optimization problem formulation has 7 solution variables and 7 objective functions. The objective functions are summarized in Table 5.1.

Table 5.1. Objective functions for normal approach trajectory generation problem formulation

	Objective Function	Formula	Purpose
1	Minimize distance to target position	$f_1(\dot{q}) = \left\|\vec{x}_d - \vec{x}_{tt,t+1}\right\|$	Move the tool-tip towards the desired target position.
2	Minimize target orientation error	$f_2(\dot{q}) = \left\lvert \cos^{-1}\left(\frac{\vec{k}_{tt,t+1} \cdot \vec{k}_d}{\left\|\vec{k}_{tt,t+1}\right\|\left\|\vec{k}_d\right\|}\right)\right\rvert$	Move the tool-tip towards the desired target orientation.
3	Optimize mobile base direction	$f_3(\dot{q}) = \left(\left\lvert cos^{-1}\left(\frac{\vec{u}_{b,t+1} \cdot \vec{k}_{b,t+1}}{\left\|\vec{u}_{b,t+1}\right\|\left\|\vec{k}_{b,t+1}\right\|}\right)\right\rvert\right)^2$	Rotate the mobile robot base so that it can move linearly in the direction of the target.
4	Optimize approach angle	$f_4(\dot{q}) = \left(\left\lvert cos^{-1}\left(\frac{\vec{u}_{tt,t+1} \cdot \vec{k}_d}{\left\|\vec{u}_{tt,t+1}\right\|\left\|\vec{k}_d\right\|}\right)\right\rvert\right)^2$	Approach the target location normal to the desired surface to prevent collisions.
5	Minimize energy consumption	$f_5(\dot{q}) = \left\|\vec{\tau}^2\right\| f_1(\dot{q})$	Balance torque output of joints with movement toward the target to generate energy-efficient trajectories.
6	Maximize manipulability of joints	$f_6(\dot{q}) = \frac{1}{\vec{q}_r - \left\|\vec{q}_m - \vec{q}_{t+1}\right\|}$	Keep joints closer to their center positions to maximize manipulability and prevent joint limit violations.
7	Minimize overall target configuration error	$f_7(\dot{q}) = f_1(\dot{q}) + f_2(\dot{q})$	Minimize the overall target configuration error at each time step.

*$\vec{x}_d$ is the desired position, $\vec{k}_d$ is the desired normal vector, $\vec{u}_{b,t+1}$ is a vector from $\vec{x}_{b,t+1}$ to $\vec{x}_d$, $\vec{u}_{tt,t+1}$ is a vector from $\vec{x}_{tt,t+1}$ to $\vec{x}_d$, $\vec{q}_r$ is the joint range from center positions, $\vec{q}_m$.

*Joint torques were computed using linearizations of the Euler-Lagrange equations.

5.3.7.1 - Constraints

Two types of constraints were used for the optimization problem formulation – hard constraints and soft constraints. Hard constraints must be satisfied to ensure the commanded trajectory of the robot can be executed and were defined as joint position limit constraints and joint torque limit constraints. Soft constraints were imposed to improve the quality of the solution vectors and the efficiency of the trajectory but could be violated if no solutions were found which satisfied all of

the constraints. In this case, the solution which had the lowest cumulative sum of soft constraint violations was selected. The soft constraints were defined as motion constraints which limit the end-effector motion to decreasing the error to the target configuration in both position and orientation. Additionally, a constraint was added which prevents the difference between the position and orientation error from increasing to encourage coordinated convergence to the target position and orientation.

5.3.8 - Problem 2: Initial Approach

The initial approach optimization problem was used to navigate the mobile robot to a location 'in front' of the desired surface normal vector to encourage an optimal approach angle to the desired target location. At the initialization of the trajectory generation, this optimization problem was used until the following two criteria were met, at which point the normal trajectory generation problem was used, where $\theta_{b,k}$ is the angle between the vector going from the base position to the target position and the target normal vector and $\theta_{tt,k}$ is the angle between the vector going from the tool-tip position to the target position and the target normal vector.

- $\|\theta_{b,k}\| < \frac{\pi}{3} rad$
- $\|\theta_{tt,k}\| < \frac{\pi}{3} rad$

In some cases, these criteria are already satisfied at the initialization of the trajectory, and thus the initial approach optimization problem is never used. During the initial approach, the robotic arm was kept in the home position, so the solution vector consists of only 2 variables – the left and right wheel joint velocities of the mobile robot. The initial approach optimization problem was formulated with 3 objective functions (Table 5.2) and 1 constraint function.

Table 5.2. Objective functions for initial approach trajectory generation problem formulation

	Objective Function	Formula	Purpose
1	Optimize base approach angle	$f_1(\dot{q}) = \left\|\theta_{b,k}\right\|$	Move mobile base to a position located 'in front' of target surface to optimize approach trajectory.
2	Optimize end-effector approach angle	$f_2(\dot{q}) = \left\|\theta_{tt,k}\right\|$	Move mobile base to a position where the tool-tip is located 'in front' of target surface to optimize approach trajectory.
3	Minimize end-effector position error	$f_3(\dot{q}) = \left\|\vec{x}_d - \vec{x}_{tt,t+1}\right\|$	Prevent the mobile base from moving too far from target position while performing initial approach.

*$\theta_{tt,k}$ is the angle between the vector from $\vec{x}_{tt,t+1}$ to $\vec{x}_d$ in the XY plane and the desired normal vector, $\vec{k}_d$.

5.3.8.1 - Constraints

The only constraint function for this problem formulation is a soft constraint requiring the base approach angle is decreased at each time step to ensure efficient completion of the initial approach step.

5.3.9 - Problem 3: Escape from Local Minimum

The final optimization problem is implemented when a local minimum is detected, determined by tool-tip oscillations and lack of progress toward the target end-effector configuration. In this case, the escape from local minimum optimization problem is used to move the robot out of the minimum and into a position from which it can more easily continue toward the target. This optimization problem formulation is used to determine the desired joint velocities until the robot has escaped the local minimum, as determined by the following criteria:

- $\left\|\vec{x}_{b,t} - \vec{x}_{b,m}\right\| > 0.5\ m$
- $\|\vec{q}_t - \vec{q}_m\| < 0.5\ rad$

This problem formulation has 5 objective functions and the same 7 solution variables (all joint velocities) as the normal approach trajectory generation problem formulation. The objective functions are summarized in Table 5.3.

Table 5.3. Objective functions for escape from local minimum problem formulation

	Objective Function	Formula	Purpose
1	Minimize base distance from local minimum position	$f_1(\dot{q}) = \left\|\vec{x}_{b,t+1} - \vec{x}_{b,m}\right\|$	Move the mobile base of the robot out of the local minima position ($\vec{x}_{b,m}$) to allow the robot arm to move toward the desired end-effector configuration from a different base position.
2	Center robot arm joints	$f_2(\dot{q}) = \left\|\vec{q}_{t+1} - \vec{q}_m\right\|$	Move the robotic arm joints to their center positions to maximize the possible motion once motion toward the target configuration resumes.
3	Optimize base approach angle	$f_3(\dot{q}) = \left\|\theta_{b,k}\right\|$	Move mobile base to a position located 'in front' of target surface to optimize approach trajectory.
4	Optimize approach angle	$f_4(\dot{q}) = \left(\left\|cos^{-1}\left(\frac{\vec{u}_{tt,t+1} \cdot \vec{k}_d}{\left\|\vec{u}_{tt,t+1}\right\|\left\|\vec{k}_d\right\|}\right)\right\|\right)^2$	Approach the target location normal to the desired surface to prevent collisions.
5	Minimize energy consumption	$f_5(\dot{q}) = \left\|\vec{\tau}^2\right\|\left\|\vec{x}_d - \vec{x}_{tt,t+1}\right\|$	Balance torque output of joints with movement toward the target to generate energy-efficient trajectories.

*$\theta_{b,k}$ is the angle between the vector from $\vec{x}_{b,t+1}$ to $\vec{x}_d$ in the XY plane and the desired normal vector, $\vec{k}_d$.

5.3.10 - Constraints

This problem formulation has the same hard constraints as the normal trajectory generation problem formulation of joint position and torque limits. Additionally, soft constraints requiring base motion away from the stuck position and robotic arm joint motion toward the center positions are added to encourage efficient escape from the local minimum.

5.3.11 - Joint Velocity Limits and Velocity Smoothing

As with most real-time generated trajectories, MOO-based trajectory generation is prone to producing trajectories with oscillations. To reduce the oscillations, double exponential smoothing was applied to the raw joint velocity commands generated from the MOO problem. However, smoothing the joint velocity commands also introduced the possibility for the joints to violate position constraints when near the ends of their ranges due to lower reactivity in the velocity commands. To address this, the joint velocity limits passed into the MOO problem were adjusted for each optimization iteration based on the position of the joints within their range using (5.14) and (5.15), thus preventing velocity commands which would lead to joint position limit violations, where $\vec{q}_l$ and $\vec{q}_u$ are the lower and upper joint limits, respectively, and $\vec{q}_t$ is the current joint positions.

$$\vec{x}_{l,new} = \max(\vec{x}_l, \vec{q}_l - \vec{q}_t) \tag{5.14}$$

$$\vec{x}_{u,new} = \min(\vec{x}_u, \vec{q}_u - \vec{q}_t) \tag{5.15}$$

5.3.12 - Biased Initialization

To help improve the quality of the solutions, a biased initialization of the population was implemented when the normal approach trajectory generation problem formulation was used. The population was initialized using a combination of the solution set from the previous optimization iteration and the joint motions which would move the end-effector in the desired direction as determined using the pseudoinverse Jacobian. The initial population was created using (5.16), where $\vec{\alpha}$ is a random vector between 0 and 1, $\vec{s}_{t-1}$ is the previous solution set, and $\vec{s}_d$ is an array of the Jacobian-based desired solution vector with random noise added.

$$\vec{s} = \vec{\alpha}\vec{s}_{t-1} + (1 - \vec{\alpha})\vec{s}_d \tag{5.16}$$

5.1 - Experiments

For all experiments, 100 target end-effector configurations were randomly generated for testing. Each target configuration consisted of a target normal vector and a target position. The target normal vectors were randomly generated in any direction. For each target normal vector, the target position was randomly generated within a 2m radius in the XY-plane, and within the achievable Z-height for the given target end-effector orientation. The same 100 target configurations were used for all experiments to ensure an accurate comparison. Additionally, all experiments used an optimization time step of 0.2s, which was proven to be achievable for real-time implementation for the selected MOO algorithm parameters. Trajectories were terminated when the end-effector was within 0.01m and 0.17 rad (~10 degrees) of the target configuration. All optimizations were performed using the NSGA-III [65] algorithm implemented in Pymoo [66].

5.1.1 - Offline Trajectory Generation and Validation

First, the MOO trajectory generation method was tested with offline trajectory generation in Python. Five trials were performed for each of the 100 target configurations, for a total of 500 trials. The first 100 trajectories (1 for each target configuration) generated using MOO in Python were recorded for validation in ROS/Gazebo. In ROS, the joint trajectory controller was used, and the joint positions at each time step were derived from the optimized joint velocity commands to create each joint trajectory. The end-effector configuration and joint torques were recorded for the duration of each trajectory execution and compared to the optimization-generated values to validate the trajectories.

5.1.2 - Simulated Real-time Trajectory Generation in ROS/Gazebo

Finally, the MOO trajectory optimization was implemented in real-time using ROS/Gazebo simulations. The same 100 target configurations were used for the real-time simulations, also with

5 trials for each configuration. For these experiments, a ROS node was written that performs the MOO and sends the commanded joint velocities to a ROS joint velocity controller. Because the time of the optimization is much longer than a traditional real-time controller update rate, predictive control was implemented by estimating the robot state at the expected termination time of the optimization (t+1) using the current joint velocity commands from the most recent optimization. The optimization is then performed for the time step t+1 to t+2 so that it finishes at time t+1 and the commanded joint velocities passed to the joint velocity controller from the optimization are current.

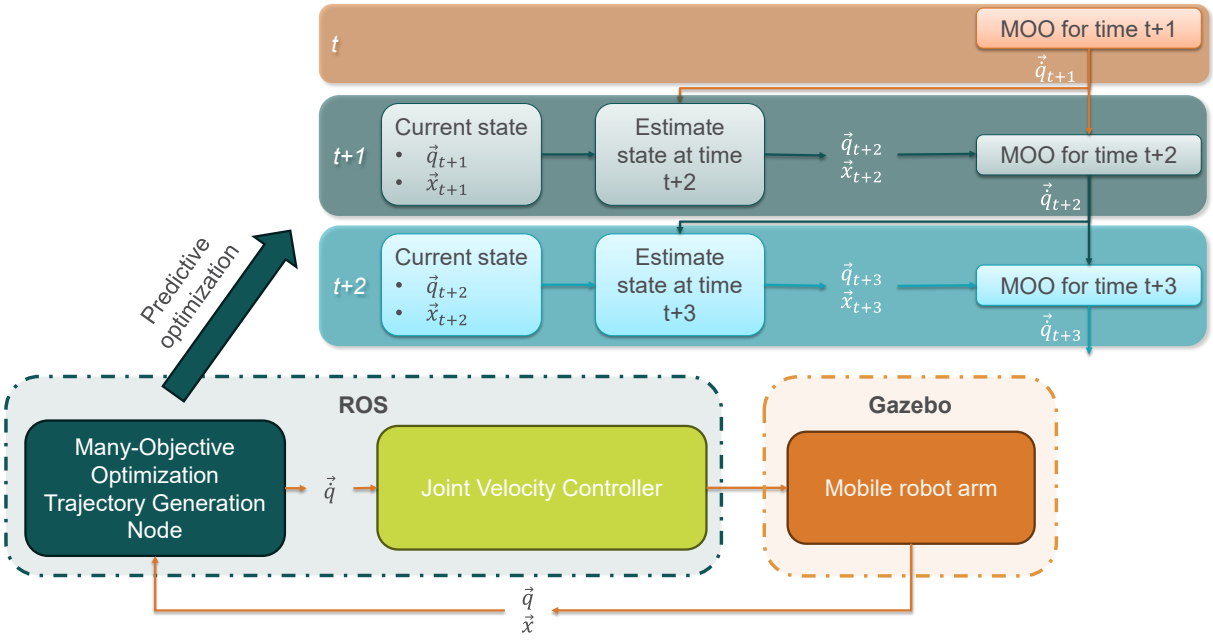


Figure 5.2. Real-time simulation testing with predictive optimization.

5.2 - Results

Results for all experiments will be presented in this section. Relevant performance metrics were determined to be as follows.

- Trajectory time: Amount of trajectory execution time required for end-effector to reach the desired target configuration within specified error tolerances.
- Trajectory length: Distance travelled by end-effector before trajectory termination at reaching target configuration.
- Path efficiency: Efficiency of the end-effector path relative to the shortest distance (straight-line) between the starting and ending end-effector positions. Calculated as straight line distance/trajectory length.
- Energy consumption: Estimated total energy consumption for trajectory execution determined by summing joint torques over the full trajectory.
- Total arm joint motion: Sum of the total motion of each of the robotic arm joints over the full execution of the trajectory.
- Arm joint efficiency: Efficiency of the arm joint trajectories relative to the shortest motion required to reach the final joint configurations from the initial joint configurations. Calculated as minimum required joint motion/total arm joint motion.

5.2.1 - Offline Trajectory Validation

For the offline trials, the proposed real-time MOO trajectory generation method generated trajectories which reached the target end-effector configurations in all trials. The target configuration was reached in under 30s for 77% of the trials, and in under 1 minute for 95% of the trials. Figure 5.3 shows very similar energy consumption distributions between the offline and simulated values, validating the method of energy consumption estimation used for the optimizations. However, the simulated trajectories show higher position error than estimated from the offline trajectories. This is due to the error in estimating the mobile base motion throughout the trajectories. Unlike the robot arm position, which has a closed form forward kinematics

solution, the position of a differential drive mobile robot must be estimated iteratively at each time step. This leads to small, compounding errors, which result in larger position estimation errors for longer trajectories, in addition to estimation errors caused by wheel slippage. For the orientation, the mobile base position estimation only affects one of the three degrees of freedom (yaw), and thus we see a much lower impact of the estimation error on the end-effector orientation.

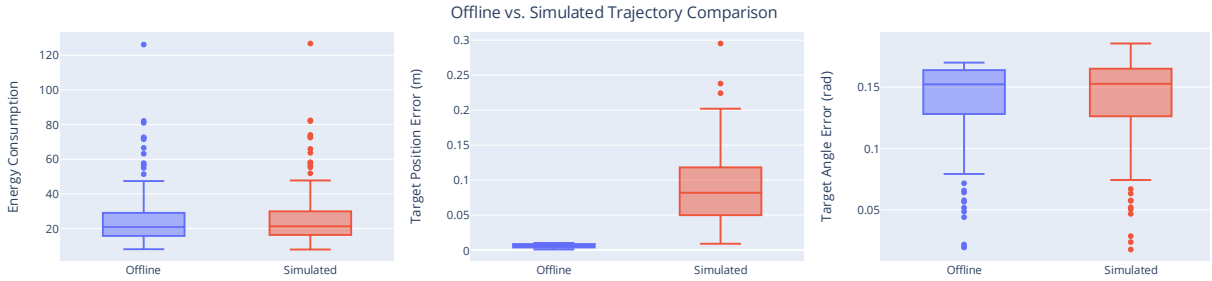


Figure 5.3. Offline versus simulated trajectory statistics.

5.2.2 - Simulated Real-time Trajectory Generation

The simulated real-time trajectories also reached the target configuration 100% of the time, with 73% of the trials terminating in under 30s, and 94% reaching the target configuration in under 1 minute (Figure 5.4). Figure 5.5 shows a comparison of various performance statistics between the offline and real-time trajectories. The similar results validate the MOO method for real-time implementation. Figure 5.6 shows a comparison of the offline and real-time generated trajectories for one target configuration.

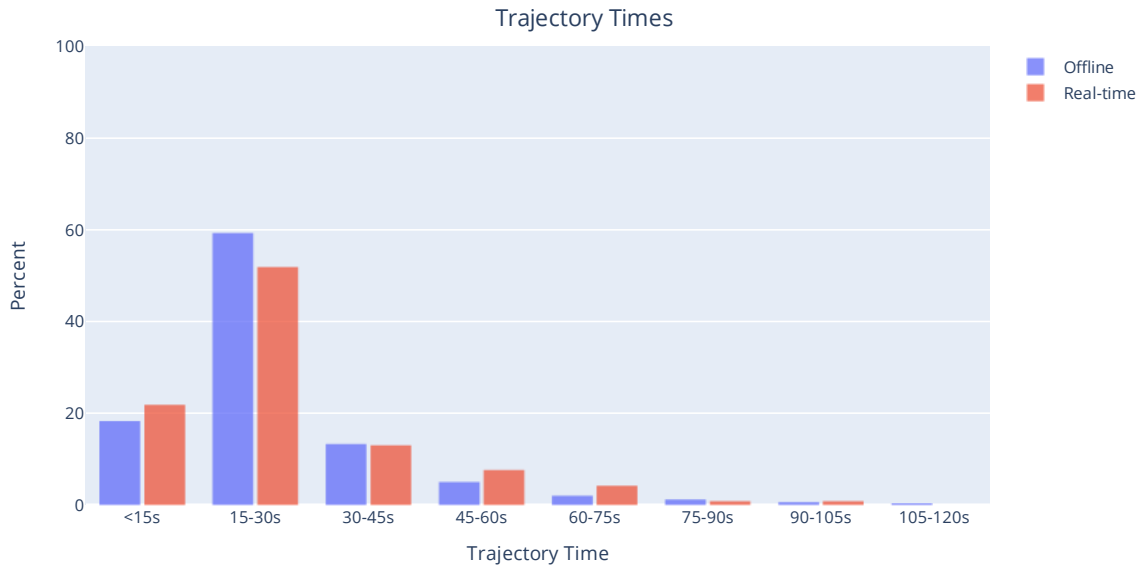


Figure 5.4. Trajectory time distribution for offline and real-time trials.

Figure 5.5. Comparison of performance metrics between offline and real-time simulated trajectories.

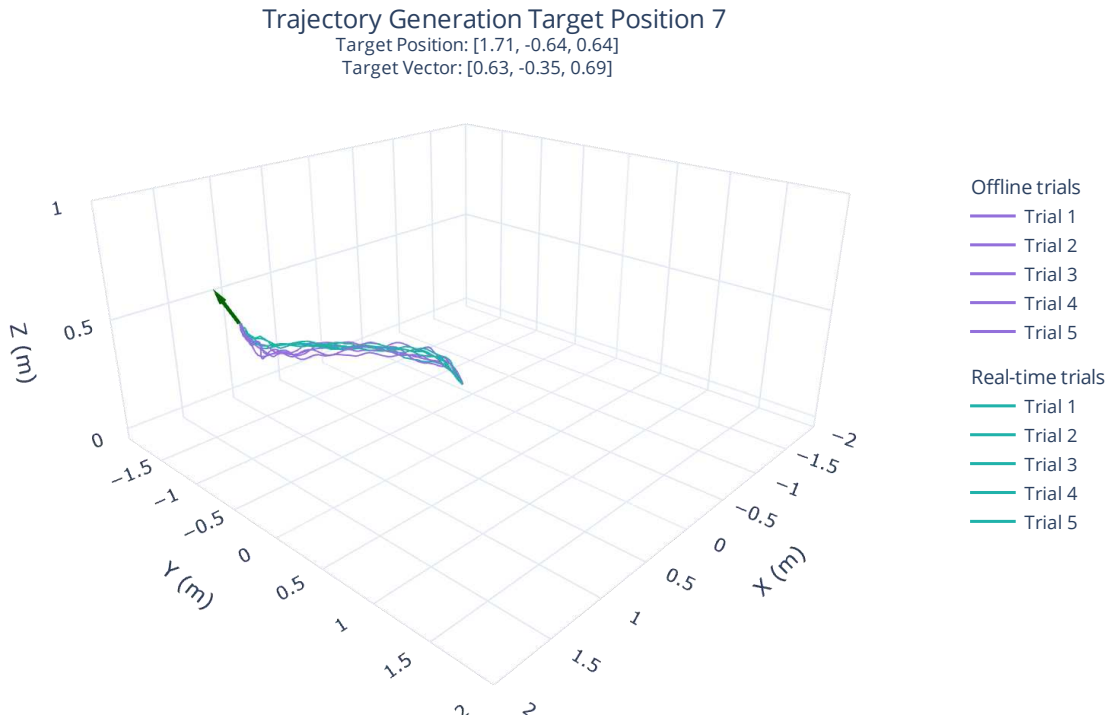


Figure 5.6. Example end-effector trajectories for all offline and real-time trials for one target configuration.

5.4 - Discussion

When generating trajectories for a robotic arm, there are two parts of the end-effector configuration which must be considered – position and orientation. While for some tasks the end-effector orientation may be unimportant (e.g. moving an object from one location to another), in many cases specific orientations of the end-effector must be achieved alongside the specified positions (e.g. grinding, bolt tightening, surface inspection, etc.). For a standard 6DOF industrial robot arm, the position and orientation of the end-effector are coupled – changing one also changes the other. This means that the complexity of the trajectory generation problem is increased by adding in desired orientations of the end-effector. An additional difficulty is coordinating the convergence of the end-effector position and orientation to the target configuration. In many cases, one converges to the desired value first, increasing the difficulty of navigating the other to the

desired value. Additionally, it is also desirable for the position and orientation to coordinate the convergence for efficient, high-quality trajectories. In this work, the objective functions and constraints were formulated to encourage coordinated convergence, but in practice this was not always achieved. Additional work exploring potential modifications to the optimization problem formulation to encourage more coordinated convergence could improve the quality of the generated trajectories.

Another major shortcoming of real-time trajectory generation is the possibility of getting stuck in local minima. This was observed frequently in the generated trajectories, resulting in longer and less efficient trajectories as the mobile robot arm had to extricate itself from a local minimum before continuing on to reach the final target configuration. Due to the indeterminate nature of the mobile robot arm control problem (infinite possible base locations, and thus inverse kinematic solutions for the robot arm), it is not feasible to determine the final desired joint positions in real-time to encourage motion toward that configuration. However, more sophisticated measures could be employed to both detect local minima and move away from them in a more efficient manner. Additionally, it could be possible to employ measures to avoid local minima and even encourage movement through a local minima continuing toward the target in such cases where that would be possible. These are all techniques that could be explored in the future to improve the quality of the trajectories but are outside the scope of this current work.

Finally, the trajectories generated in this work assume an obstacle-free environment. This is an unrealistic case in real-world applications and also allows for the possibility of self-collisions. In a few trials, it was observed that the mobile robot arm experienced self-collisions, which should be avoided in practice. Future work will focus on real-time trajectory generation with obstacle avoidance, including prevention of self-collisions.

Chapter 6 – Mobile robot arm trajectory generation for trajectory following

6.1 - Introduction

In the previous chapter, a method was presented for moving a mobile robot arm into position to complete a task. Executing a manipulation task often involves executing a specific trajectory. In this chapter, two methods are explored for generating a mobile robot trajectory for executing a desired end-effector trajectory. While trajectory generation has been widely researched for a stationary robotic arm, much less research exists addressing coordinated movement of a robotic arm mounted on a mobile base. There are many advantages that could be achieved with such a configuration, expanding the range of capabilities and manipulation tasks achievable with a robotic arm. Some advantages include tracking of more complex trajectories, essentially creating an infinite working field in the XY plane, along with more energy-efficient trajectories and the ability to use lower DOF robotic arms to accomplish the same tasks as a higher DOF robotic arm when coupled with a mobile base.

While there are a number of advantages in using a coordinated mobile robotic arm platform for manipulation tasks, the addition of the mobile base also significantly increases the complexity of the control and trajectory generation problems. The added redundancy of the degrees of freedom of the mobile base greatly increases the size of the solution search space for the trajectory generation problem. While this improves the chances of the existence of a continuous solution for any given end-effector trajectory, it also increases the computational intensity of searching for this solution, especially since computationally-efficient methods such as inverse kinematic analysis can no longer be employed independently. In this chapter we explore two methods of approaching the mobile robotic arm trajectory generation problem, with the goal of finding a reasonable

solution for an arbitrary trajectory within a reasonable amount of computational time. It should be noted that neither method presented exhaustively searches the solution space, and thus it is likely that a closer-to-optimal solution could be found in nearly all cases. Another consideration is that the definition of the optimal trajectory can vary depending on the objectives and requirements of the task. In the following methods, we focus on generating continuous trajectories for smooth tracking, minimizing joint motions, and minimizing energy consumption for the execution of the trajectories. The following chapter will address controller design for integrated control of a mobile robot arm for tracking the trajectories generated using the methods outlined in this chapter.

6.2 - Background

6.2.1 - Robotic Arm Trajectory Generation

Much research has been performed in the space of robotic arm trajectory generation. Much of the research has focused on the generation of smooth trajectories for improved execution. Zhang et al. used inverse kinematics with 5^{th}-order polynomial interpolation to produce smooth end-effector trajectories for a 4DOF material handling robot [67]. Raji et al. propose using a 7^{th}-order polynomial with 2 via points to generate limited jerk trajectories for a 6DOF robot in joint space [68]. Huda et al also used inverse kinematics to perform trajectory planning for a 6DOF arm for straight-flat path tracking for welding applications [69]. A deep reinforcement learning (DRL) approach was presented in [70] for planning 6DOF robotic arm trajectories in a more computationally efficient method than more traditional inverse kinematics-based methods. [71] used PSO to optimize trajectories in joint space for minimum execution time with a smooth and stable trajectory.

Additionally, research has been performed on creating energy-efficient robotic arm trajectories. Das et al. used a differential evolution optimization algorithm to optimize the locations

of intermediate points along a trajectory for a 3DOF robotic arm with the objectives of minimizing energy consumption and avoiding obstacles [72]. Their proposed energy consumption minimization function was the summation of differential potential energy and kinetic energy between points along the trajectory. [73] used invasive weed optimization to optimize path for a 2DOF robot arm by optimizing via point positions for obstacle avoidance and minimizing energy consumption. Zhang et al used many-objective optimization within the framework of DRL for trajectory planning of a 6DOF robotic arm for accuracy, smoothness, and energy consumption minimization [74].

6.3 - Methods

The original goal of this research was to expand the real-time many-objective optimization (MOO) trajectory generation method presented in the previous chapter (Chapter 5) to include path following and obstacle avoidance. This method was explored for end-effector trajectory tracking and the modifications and results obtained will be presented in this chapter. However, this approach presented some shortcomings for this problem, resulting in a less-than-satisfactory success rate in following a specified end-effector trajectory within a reasonable error tolerance. Because of this, the objective of the research was modified to offline trajectory generation focused on path following (with methods that could potentially be extended to include obstacle avoidance). In this section, two different methods of trajectory generation – MOO-based trajectory generation and sampling-based combinatorial optimization (SBCO) trajectory generation – will be explained in detail.

6.3.1 - Differential Drive Trajectory Generation

The mobile robot arm configuration used in this research uses a differential drive style mobile robot for the mobile base. The differential drive robot (DDR) has a simple mechanical design and

control infrastructure and is therefore one of the most common types of wheeled robots. However, the differential drive configuration is nonholonomic, meaning that it has 3 degrees of freedom (X, Y, and orientation) controlled by 2 variables. This creates a more difficult control problem because the robot is limited in the direction that it can move by the direction it is facing – it can only move in the XY plane in the direction it is oriented and cannot move perpendicular to this direction (Figure 6.1), which makes it difficult to track irregular paths. In the previous chapter, the wheel motion of the differential drive robot and the joint motion of the robotic arm were solved together to create a cohesive end-effector trajectory. However, the nonholonomic constraints of the DDR base make the solution space more complex when the trajectory problem is framed in this way. Another option is to generate the base path trajectory (XY path only) and then design a DDR controller to allow the robot to track this base trajectory. This option is explored to simplify the trajectory generation problem for both the MOO and sampling-based combinatorial optimization methods and leads to further research designing a cohesive mobile robot arm controller including the DDR trajectory tracking controller, presented in the following chapter (Chapter 7).

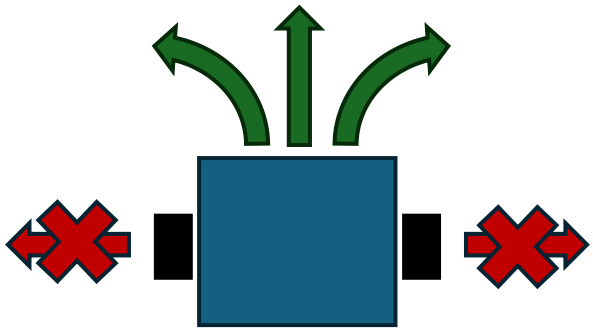


Figure 6.1. DDR non-holonomic motion constraints

6.3.2 - MOO Method

The original approach for path following mobile robot arm trajectory optimization was to modify the MOO problem presented in the previous chapter for path following. To accomplish this, the objective functions were kept the same, but constraint functions were modified to keep the position and orientation end-effector errors below acceptable thresholds for path following. To follow the path, the target orientation of the end-effector was then updated at each time step to follow the desired end-effector trajectory. The full trajectory was generated by running the approach trajectory generation problem (Chapter 5) with the starting end-effector configuration of the path as the target configuration until the end-effector was within the specified error thresholds of that configuration, and then switching to the trajectory following problem formulation with updating target positions until the full trajectory was executed. The objective functions for this problem formulation are given in Table 6.1.

Table 6.1. Objective functions for original trajectory following problem formulation

	Objective Function	Formula	Purpose
1	Minimize distance to target position	$f_1(\dot{q}) = \left\|\vec{x}_d - \vec{x}_{tt,t+1}\right\|$	Move the tool-tip towards the desired target position.
2	Minimize target orientation error	$f_2(\dot{q}) = \left\lvert \cos^{-1}\left(\frac{\vec{k}_{tt,t+1} \cdot \vec{k}_d}{\left\|\vec{k}_{tt,t+1}\right\|\left\|\vec{k}_d\right\|}\right)\right\rvert$	Move the tool-tip towards the desired target orientation.
3	Optimize motion direction	$f_3(\dot{q}) = \left\lvert cos^{-1}\left(\frac{\vec{v}_t \cdot \vec{v}_{d,t}}{\left\|\vec{v}_t\right\|\left\|\vec{v}_{d,t}\right\|}\right)\right\rvert$	Move the end-effector in the same direction as the desired trajectory motion.
4	Optimize linear velocity	$f_4(\dot{q}) = \left\|\vec{v}_t - \vec{v}_{d,t}\right\|$	Track the desired trajectory linear tool-tip velocity.
5	Minimize energy consumption	$f_5(\dot{q}) = \left\|\vec{\tau}^2\right\|$	Minimize torque output of joints to generate energy-efficient trajectories.
6	Maximize manipulability of joints	$f_6(\dot{q}) = \frac{1}{\vec{q}_r - 0.8 * \left\|\vec{q}_m - \vec{q}_{t+1}\right\|}$	Keep joints closer to their center positions to maximize manipulability and prevent joint limit violations.
7	Minimize joint velocities	$f_7(\dot{q}) = \vec{\dot{q}}_t$	Reduce joint motion and joint velocities.

*$\vec{x}_d$ is the desired position, $\vec{k}_d$ is the desired normal vector, $\vec{v}_{,t}$ is a vector of the linear tool-tip velocity, $\vec{v}_{d,t}$ is a vector of the desired trajectory linear tool-tip velocity, $\vec{q}_r$ is the joint range from center positions, $\vec{q}_m$, for the 2nd through 5th robotic arm joints.

*Joint torques were computed using linearizations of the Euler-Lagrange equations.

After some preliminary testing, it was clear that this method of real-time trajectory generation was unable to produce satisfactory results for trajectory following. It is likely this was due to combination of factors, with the two main contributions being the limitation of searching such a large solution space (7 variables and 7 objective functions) quickly for real-time implementation, and the dependence of a given trajectory being achievable on the configuration of the mobile robot

arm at the initiation of the trajectory. While there are theoretically 'infinite' possible configurations of the mobile robot arm for a given target configuration due to DOF redundancies, only a subset of these may allow for tracking of a given trajectory without encountering discontinuities in the required joint configurations for the full trajectory. However, the set of feasible starting configurations is specific to the entirety of the desired trajectory, making it nearly impossible to determine the feasible set for any possible trajectory (i.e. a universal method for predicting the feasible starting configuration set for any possible trajectory). This makes it very difficult to create a real-time trajectory-following method if the feasible starting configurations for the trajectory cannot be determined *a priori*, since this means that, if one is generating the trajectory in a forward-stepping method such as is required for real-time generation, the starting configuration could predetermine failure of successful trajectory tracking if not in the (unknown) feasible set. Based on these observations, a few different modifications were tried for offline trajectory generation, still using a MOO-based approach.

6.3.2.1 - Inverse Kinematics for Reduction of Optimization Search Space

One of the potential problems of using the MOO-based method is the size of the search space for the given problem formulation. The mobile robot arm has seven solution variables - two for wheel joint velocities and five for robot arm joint velocities (recall the 6^{th} joint velocity is kept to 0 due to how the end-effector orientation is specified (Section 5.3.1 - Defining the Target End-effector Configuration)), which creates a very large search space for the optimization. A potential solution to this problem is to decrease the size of the search space by reducing the number of variables in the optimization problem. For the specific case of a standard industrial 6DOF robot arm, such as that used for this research, there exists a closed-form analytic inverse kinematic solution (note that this is not the case for any robotic arms with redundant degrees of freedom).

(Inverse kinematics solution is given in Appendix C.) This analytic inverse kinematics solution was used to decrease the search space of the optimization problem for seven variables to two variables – just the two wheel joint velocities of the differential drive robot. This was done by using the wheel velocities to calculate the new base configuration (position and orientation), then calculating the new relative target position of the end-effector from this base configuration (i.e. calculating the desired end-effector configuration for the robot arm in isolation), and then solving the inverse kinematics of the robotic arm for this relative target configuration and determining the robotic arm joint velocities from the inverse kinematics solution. The objective functions for this modified problem formulation are given in Table 6.2.

Table 6.2. Objective functions for inverse kinematics trajectory following problem formulation

	Objective Function	Formula	Purpose
1	Minimize distance to target position	$f_1(\dot{q}) = \left\|\vec{x}_d - \vec{x}_{tt,t+1}\right\|$	Move the tool-tip towards the desired target position.
2	Minimize target orientation error	$f_2(\dot{q}) = \left\lvert \cos^{-1}\left(\frac{\vec{k}_{tt,t+1} \cdot \vec{k}_d}{\left\|\vec{k}_{tt,t+1}\right\|\left\|\vec{k}_d\right\|}\right)\right\rvert$	Move the tool-tip towards the desired target orientation.
3	Minimize energy consumption	$f_3(\dot{q}) = \left\|\vec{\tau}^2\right\|$	Minimize torque output of joints to generate energy-efficient trajectories.
4	Maximize manipulability of joints	$f_4(\dot{q}) = \frac{1}{\vec{q}_r - 0.8 * \left\|\vec{q}_m - \vec{q}_{t+1}\right\|}$	Keep joints closer to their center positions to maximize manipulability and prevent joint limit violations.

*$\vec{x}_d$ is the desired position, $\vec{k}_d$ is the desired normal vector, $\vec{q}_r$ is the joint range from center positions, $\vec{q}_m$, for the 2nd through 5th robotic arm joints.

*Joint torques were computed using linearizations of the Euler-Lagrange equations.

There are a few considerations when implementing this method of variable reduction in the search space. The first is that for a 6DOF robotic arm, there are up to eight possible inverse

kinematic solutions for a given end-effector configuration. In the case where there is more than one possible solution, the best solution must be chosen. This was done based on trajectory continuity, so the inverse kinematics solution with the smallest difference in joint positions from the robotic arm joint configuration at the previous time step was chosen to create the most continuous trajectory and minimize the robotic arm joint velocities. The second consideration is that, because there are multiple possible inverse kinematic solutions for a given end-effector configuration, it is possible that, while there exists a solution for a given base configuration, it may not be a continuous solution from the robotic arm configuration at the previous time step (i.e. a very large jump in joint positions between consecutive time steps). Obviously, this scenario would cause the required joint velocities to exceed joint velocity limits and create an infeasible trajectory for the robotic arm. To prevent this, when solving for the required robotic arm joint velocities for an inverse kinematic solution, the joint velocities were capped at the specified joint velocity limits. The actual resulting robotic arm configuration would then be calculated from the new velocities, and the resulting end-effector error would be calculated using the same method as when the robotic arm velocities were solution variables. Finally, there is the case where a base configuration would result in a relative target position with no inverse kinematic solutions. In this case, the robot arm joint velocities would be set to zero and the end-effector error would again be calculated in the same way as previously.

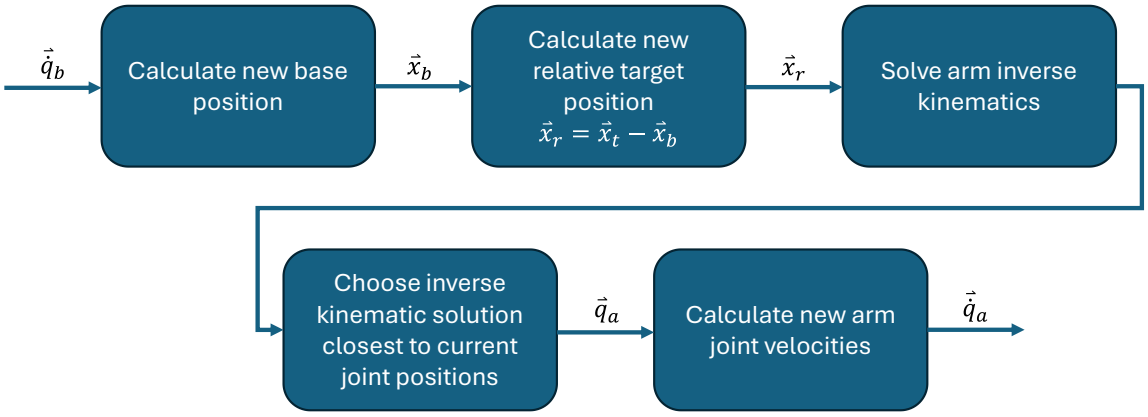


Figure 6.2. Flowchart of method for using robotic arm inverse kinematics to reduce MOO

One of the main drawbacks of this method is that, when the differential drive base gets into a configuration with no inverse kinematics solutions for the robotic arm, the robotic arm no longer tracks the trajectory at all (even outside of the error tolerances), making it very difficult to recover from tracking errors. To address this issue, this solution was implemented as a hybrid method where the inverse kinematics search space reduction was implemented while the trajectory was being tracked, and the original full search space (with seven solution variables) was used for error recovery until the end-effector was once again tracking the trajectory within the error tolerances (Figure 6.3).

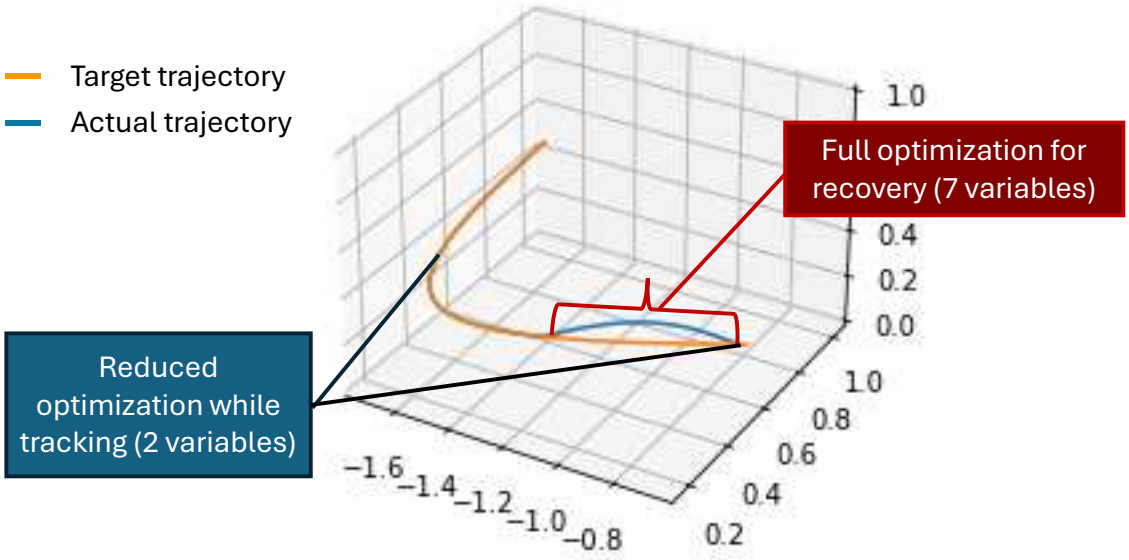


Figure 6.3. Example of hybrid MOO approach implementation.

6.3.2.2 - Omnidirectional Base Trajectory

To simplify the mobile base solution space, the MOO problem was formulated using the X and Y base velocities as the solution variables, rather than the left and right wheel velocities of the DDR. This was done using both the full arm joint solution space and the hybrid inverse kinematics method. Table 6.3 gives the objective functions for these problem formulations.

Table 6.3. Objective functions for omnidirectional base path following problem formulation

	Objective Function	Formula	Purpose
1	Minimize distance to target position	$f_1(\dot{q}) = \left\|\vec{x}_d - \vec{x}_{tt,t+1}\right\|$	Move the tool-tip towards the desired target position.
2	Minimize target orientation error	$f_2(\dot{q}) = \left\|\cos^{-1}\left(\frac{\vec{k}_{tt,t+1} \cdot \vec{k}_d}{\left\|\vec{k}_{tt,t+1}\right\|\left\|\vec{k}_d\right\|}\right)\right\|$	Move the tool-tip towards the desired target orientation.
3	Smooth base trajectory	$f_3(\dot{q}) = \left\|cos^{-1}\left(\frac{\vec{v}_{b,t} \cdot \vec{v}_{b,t-1}}{\left\|\vec{v}_{b,t}\right\|\left\|\vec{v}_{b,t-1}\right\|}\right)\right\|$	Minimize the change in angle between the linear base velocity at the current and previous time steps.
4	Minimize joint accelerations	$f_4(\dot{q}) = \left\|\dot{\vec{q}}_t - \dot{\vec{q}}_{t-1}\right\|$	Reduce changes in joint and base velocities.
5	Minimize energy consumption	$f_5(\dot{q}) = \left\|\vec{\tau}^2\right\|$	Minimize torque output of joints to generate energy-efficient trajectories.
6	Maximize manipulability of joints	$f_6(\dot{q}) = \frac{1}{\vec{q}_r - 0.8 * \left\|\vec{q}_m - \vec{q}_{t+1}\right\|}$	Keep joints closer to their center positions to maximize manipulability and prevent joint limit violations.

*$\vec{x}_d$ is the desired position, $\vec{k}_d$ is the desired normal vector, $\vec{v}_{b,t}$ is a vector of the linear base velocity, $\vec{v}_{b,t-1}$ is a vector of the linear base velocity at the previous time step, $\vec{q}_r$ is the joint range from center positions, $\vec{q}_m$, for the 2nd through 5th robotic arm joints.

*Joint torques were computed using linearizations of the Euler-Lagrange equations.

6.3.2.3 - Restarting Trajectories

Finally, to attempt to avoid the mobile robot arm failing to track the trajectory due to reaching 'dead end' configurations where continued tracking was no longer possible, the best method was reimplemented using trajectory restarting. This was implemented by allowing the algorithm up to four attempts to successfully track the trajectory, as follows. If the first trajectory failed, the 'fail configuration' was recorded and the trajectory generation was restarted from the beginning of the trajectory adding an objective function seeking to maximize the distance between the current robot

configuration and the ‘fail configuration’ in order to avoid reaching the same ‘dead end’ configuration. If the trajectory generation failed again, the robot configuration at the start of the trajectory was recorded (‘fail start configuration’) and the approach trajectory was rerun adding an objective function to maximize the distance between the robot configuration and the ‘fail start configuration.’ If the trajectory failed again from this new starting configuration, the trajectory was run one more time, avoiding the new ‘fail configuration.’ If this final try exceeded the tracking error tolerances, the trajectory generation was allowed to finish running, and the trajectory was considered a failure.

6.3.3 - Sampling-Based Combinatorial Optimization (SBCO) Method

All of the MOO-based approaches that were explored used a forward-stepping trajectory generation method with a predetermined starting configuration. However, these approaches do not address the problem of the feasible starting configuration set, and the forward-stepping method is more constrained from exploring all possible joint trajectories for a given trajectory. For offline trajectory generation, it makes sense to explore a more holistic method of framing the problem by considering the optimal trajectory as a whole rather than on a step-by-step basis. To do this, a method was developed where a sampling of possible configurations (robot base position and corresponding robotic arm joint configuration) was generated for each point on the trajectory, and then a combinatorial optimization was performed to select the combination of sampled points which created the best overall trajectory. This method can be broken down into four steps (point sampling, clustering, combinatorial optimization, and post-processing and approach trajectory generation), each of which is described in detail in the following sections.

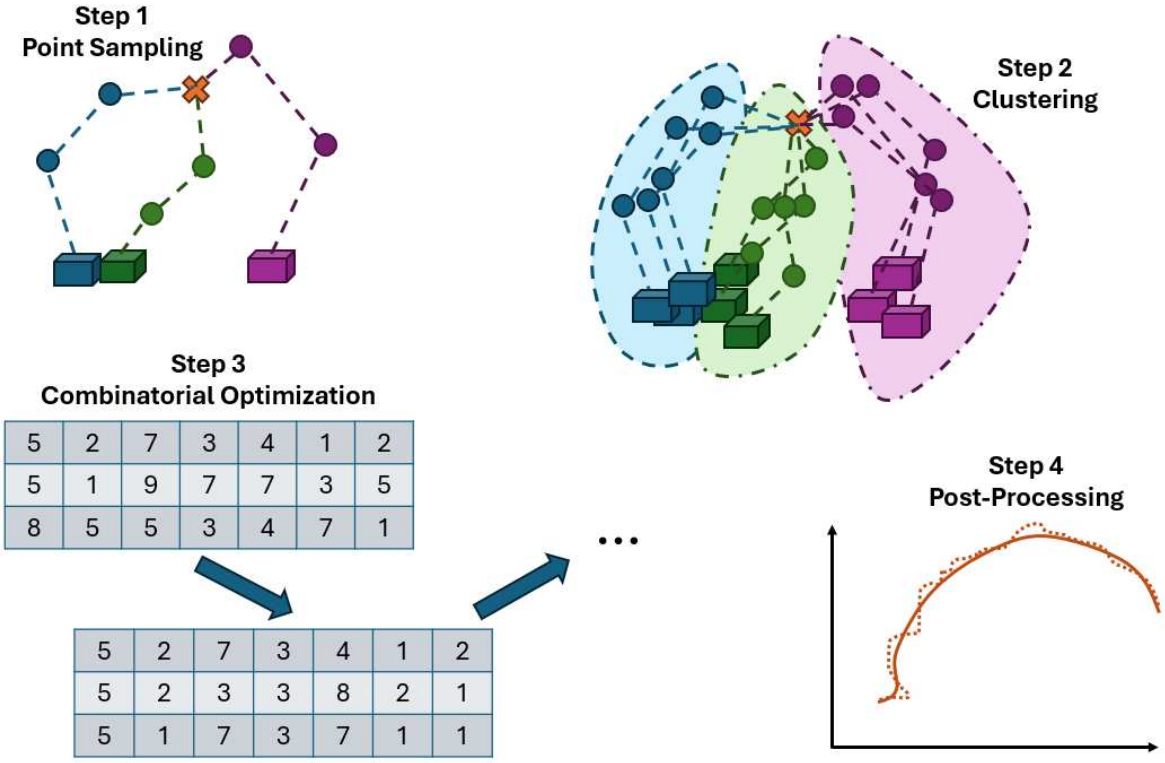


Figure 6.4. Illustration of the steps of the SBCO method.

6.3.3.1 - Point Sampling

The first step of the SBCO method is sampling possible configurations for each point on the desired trajectory. Each possible configuration point consists of the XY base position and five robot joint positions. Points were generated by sampling possible base positions for each point on the trajectory and then calculating the corresponding arm joint positions using the closed-form inverse kinematics solution, meaning that any possible base position could result in 0-8 possible configuration points, depending on the number of corresponding inverse kinematic solutions for the relative configuration determined by the trajectory point and base position. Possible points were then scored based on energy consumption (determined by the potential energy component of the Euler-Lagrange equations for the robotic arm joint configurations), joint positions (distance of

the robotic arm joints from their center positions), and proximity to the closest consecutive configuration on the trajectory (how close the base and joint positions were to any point generated for the previous time step). Sampled points were sorted based on these criteria, and the best 150 robot configurations were selected for each trajectory point for the final array of sampled configurations (dimensions N x 150, where N is the number of points on the desired trajectory).

6.3.3.2 - Clustering

Generating a reasonable number of sample configurations for each sample point creates an enormous number of trajectory combinations to optimize between (in this case, 150^N). Even using a global search algorithm, it is not feasible to search this entire space for the best trajectory combination, especially since the search space is discrete so that gradient search methods cannot be employed. To reduce the search space and ensure continuous trajectories (no large jumps in base or joint positions between consecutive time steps), two steps of clustering were performed prior to the combinatorial optimization search – point clustering and trajectory grouping.

Point clustering was performed in order to group similar configurations together at each trajectory point. Because it was unknown how many clusters would exist at each trajectory point, it did not make sense to use a clustering method like k-means. Instead, clustering was performed iteratively by adding points to a cluster if they were below a mean distance (L2 norm distance of base and joint positions) from the centroid of a cluster. A new cluster was created if the point was above this threshold distance from all existing clusters, until all points were assigned to a cluster. Figure 6.5 illustrates the point clustering.

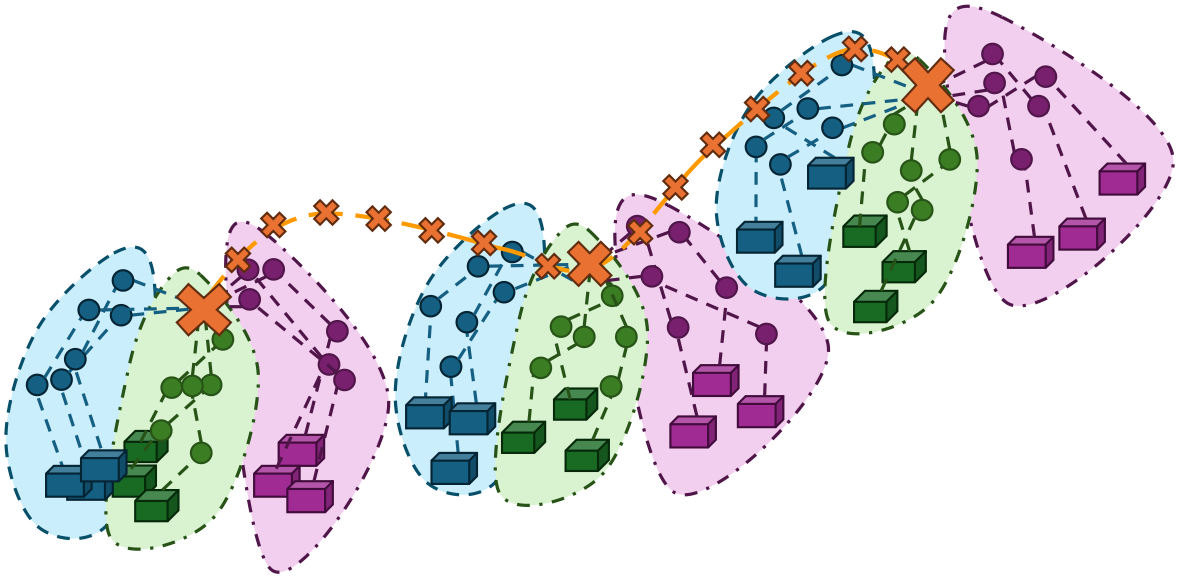


Figure 6.5. Point clustering and trajectory grouping process. Configurations in the same color show point clusters. Clusters of the same color illustrate trajectory groups for highlighted points along the trajectory.

Next, trajectory grouping was performed between the point clusters. The goal of the trajectory grouping was to group point clusters together which made up continuous trajectories. This was done in a forward-stepping method by selecting a cluster from the first trajectory point, then grouping it with all clusters below a mean distance (L2 norm) from the second trajectory point, and so on for the length of the trajectory. For the trajectory clustering, point clusters could be assigned to more than one trajectory group so long as they would form a continuous trajectory with any of the assigned trajectory groups. Any trajectory groups without a full continuous trajectory were eliminated. Once the trajectory groups were formed, the individual point clusters within each trajectory group were dissolved to create a singular configuration list for each point on the trajectory. In the case where no full trajectory groups were formed during the initial round of clustering, the full clustering process was performed again (point clustering and trajectory grouping) with an iteratively higher threshold distance until a continuous trajectory group was

found, or a maximum distance threshold was reached, in which case there was no continuous trajectory solution found. Figure 6.6 shows the entire clustering procedure flowchart.

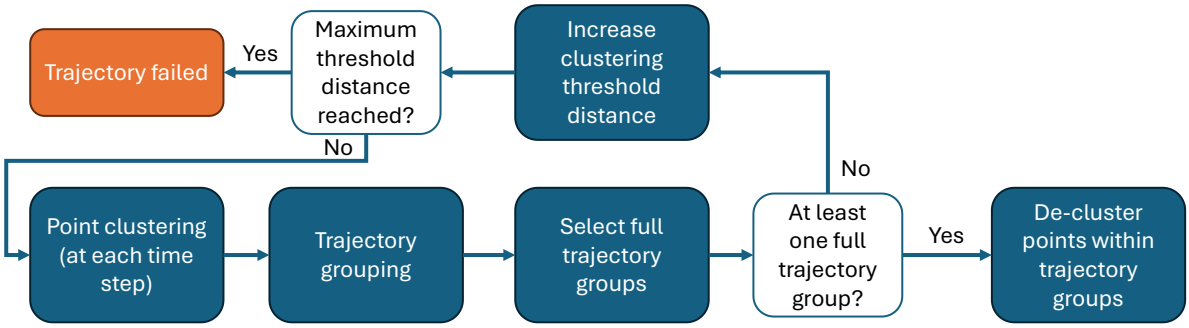


Figure 6.6. Flowchart of point clustering and trajectory grouping procedure.

6.3.3.3 - Combinatorial Optimization and Trajectory Selection

Once the trajectory groups are formed, the next step is performing combinatorial optimization within each one of the trajectory groups. The objectives for evaluating the quality of each solution focused on trajectory continuity (minimizing joint and base motion between consecutive time steps) and keeping the robotic arm joints closer to their center positions to reduce the chances of encountering joint limits. Table 6.4 shows the objective functions for the combinatorial optimization problem. One constraint function was employed, specifying the upper limit of joint motion between consecutive time steps, again to ensure trajectory continuity. An optimization was performed within each trajectory group using NSGA2 (population size of 100 for 100 generations). The final trajectory was selected between the combinatorial optimization within each trajectory group by choosing the trajectory with the lowest constraint function (i.e. lowest maximum joint motion between consecutive time steps).

Table 6.4. Objective functions for combinatorial optimization problem formulation

	Objective Function	Formula	Purpose
1	Minimize base motion	$f_1(\vec{x}) = \sum_{i=2}^{n} \left\| \vec{x}_{b,i} - \vec{x}_{b,i-1} \right\|$	Minimize total base motion of trajectory.
2	Minimize arm joint motion	$f_2(\vec{x}) = \sum_{i=2}^{n} \left\| \vec{x}_{a,i} - \vec{x}_{a,i-1} \right\|$	Minimize total arm joint motion of trajectory.
3	Minimize change in base linear velocity direction	$f_3(\vec{x}) = \sum_{i=2}^{n} cos^{-1}\left(\frac{\vec{v}_{b,i} \cdot \vec{v}_{b,\mathrm{i}-1}}{\left\| \vec{v}_{b,i} \right\| \left\| \vec{v}_{b,\mathrm{i}-1} \right\|} \right)$	Reduce abrupt changes in direction of base velocity to create smooth trajectory.
4	Minimize maximum joint motions	$f_4(\vec{x}) = \sum_{i=2}^{n} \max\left(\vec{x}_{a,i} - \vec{x}_{a,i-1} \right)$	Reduce large changes in position between time steps for arm joints.
5	Maximize manipulability of joints	$f_5(\vec{x}) = \sum_{i=1}^{n} \frac{\vec{x}_{a,i}}{\vec{q}_r}$	Keep joints closer to their center positions to maximize manipulability and prevent joint limit violations.

*$\vec{x}_b$ is the base position, $\vec{x}_a$ is the arm joint positions $\vec{v}_{,b}$ is the linear base velocity, and $\vec{q}_r$ is the joint range from center positions for the 2nd through 5th robotic arm joints.

6.3.3.4 - Trajectory Post-Processing

Because the combinatorial optimization was done in a discrete space (the sampled points) representing a limited subset of all the possible trajectories, the raw optimization output produces a generally 'noisy' result. This is suboptimal for control, and thus the trajectory output of the combinatorial optimization was post-processed to create a smoother, easier-to-track final trajectory. The post-processing was performed in two steps – smoothing the base trajectory and recalculating the arm joint trajectory. The base trajectory and arm joint trajectories were smoothed using a Gaussian filter with a sigma value of 3.0. Then, for each smoothed base trajectory point, the inverse kinematics for the robotic arm were recalculated using the adjusted relative position

between the target configuration and the smoothed base position. Because the inverse kinematics of the 6DOF arm configuration can have up to eight solutions, the solution closest to the smoothed trajectory joint positions (first point) or previous time step adjusted joint positions (any other point) was selected. Figure 6.7 shows an example raw trajectory and the final post-processed trajectory for a sample trajectory. Some special cases were handled as follows:

Case 1: Closest inverse kinematics solution of first trajectory point L2 norm distance is greater than 0.5 from original arm joint solution.

Solution: Smoothed base position is iteratively moved closer to the raw trajectory base position until the L2 norm distance is <0.5.

Case 2: Closest inverse kinematics solution of any other trajectory point L2 norm distance is greater than 0.75 from either previous time step point arm joint position (smoothed positions) or raw trajectory arm joint position.

Solution: Joint position was adjusted to be defined as the mean of the previous time step (adjusted position) and next time step (smoothed position) if the L2 norm of these 2 points was <0.75, or kept as the smoothed joint positions otherwise. If the point was the last trajectory point, it was adjusted to be equal to the previous time step adjusted joint positions. In this case the resulting end-effector position and orientation errors were calculated and recorded since the tracking was no longer exact.

Case 3: No inverse kinematic solutions exist for the smoothed base position.

Solution: The smoothed base position is iteratively adjusted closer to the original base position until there exists at least one inverse kinematic solution. At this point, if Case 1 or Case 2 applies, they are handled as described above.

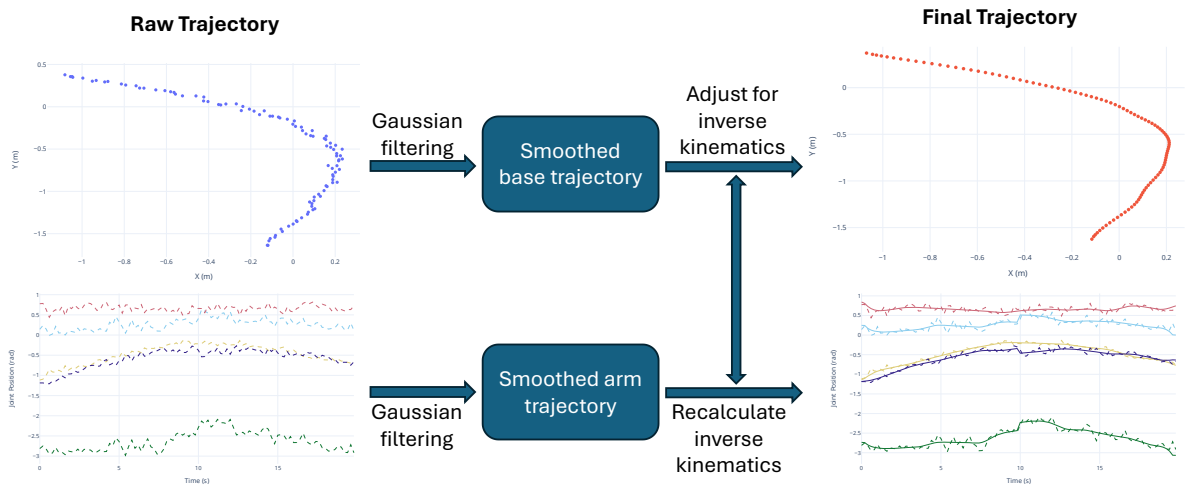


Figure 6.7. Post-processing of the SBCO trajectories.

6.3.3.5 - Approach Trajectory Generation

Since the SBCO method determines the starting configuration of the mobile robot arm, the approach trajectory must then be generated from the final task trajectory starting point. As such, a different method should be used than that presented in Chapter 5, which was utilized for the MOO trajectory generation method explored in Section 6.3.2 - MOO Method. In this case, the start (home position) and end (starting trajectory configuration) are known, so more traditional methods can be used to generate the trajectory between these points. Here, the base trajectory was generated as a Bezier curve with four reference points, where the two end points were the starting and ending base positions, and the two waypoints created a curve tangent to the starting orientation of the robot base and the initial direction of the base trajectory for path following (to create a smooth base trajectory transition). The arm joint trajectories were generated using zero-jerk motion equations between their relative start and end positions, given by (6.1), where $\vec{q}_r$ is the vector from the initial joint positions to the joint positions at the start of the trajectory. The duration of the approach trajectory was determined by the maximum specified base and arm joint velocities and

their respective travel distances, with the longest required travel time dictating the approach trajectory time, and all trajectories calculated from that for coordinated arrival at the starting configuration.

$$\vec{q}(t) = 10\vec{q}_r t^3 - 15\vec{q}_r t^4 + 6\vec{q}_r t^5 \tag{6.1}$$

6.3.4 - Testing and Validation

Both methods of trajectory generation were tested on the same set of sample trajectories. The test set consisted of 500 pseudo-randomly generated trajectories, generated as Bezier curves with time-varying end effector orientations. The Bezier curves were generated with the following constraints:

- Z-position within the achievable range for the corresponding end-effector orientation.
- All reference points are located within a 2m radius of the home position.
- Starting point is located within a 1m radius of the home position.
- Each waypoint is located between 0.5m and 1.5m of the previous waypoint.
- Each waypoint orientation within 30° (π/6 rad) of the previous waypoint orientation.
- All curves were generated with 3 waypoints.
- Each curve segment was divided into 50 time steps of 0.2s for trajectories consisting of 100 points, or lasting 20s.

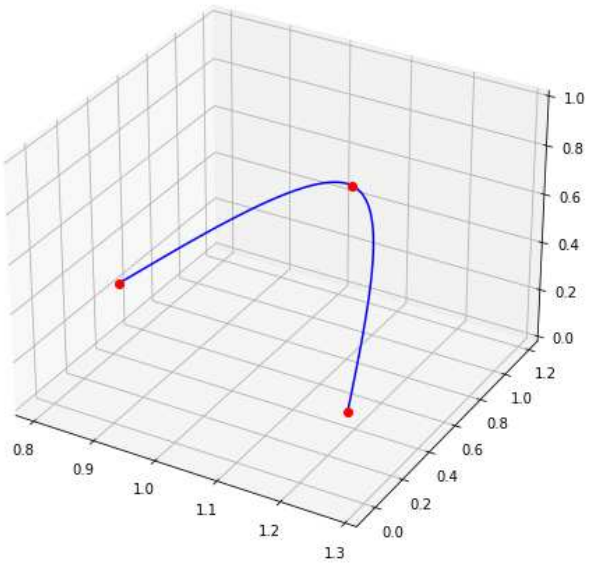


Figure 6.8. Sample test trajectory.

Each trajectory generation method was tested on all 500 sample trajectories, and the position and end effector tracking errors were recorded throughout each trajectory. For all experiments, acceptable tracking error thresholds were set to 0.005m and 0.1 rad (~5°).

6.4 - Results

For all experiments and trials, a trajectory was considered successful if the end-effector tracking error stayed below the specified error thresholds (0.005m position error and 0.1 rad orientation error) for the entire trajectory (excluding the approach trajectory). All other error metrics were calculated only over the unsuccessful trajectories. The relevant error metrics were defined as:

- Error time: Amount of trajectory time that the end-effector was above at least one of the error thresholds.

- Error percent: Percentage of the trajectory where the end-effector was above at least one of the error thresholds. Calculated as error time/trajectory time.
- Maximum position error: Maximum position error at any point in the trajectory.
- Maximum angle error: Maximum angle error at any point in the trajectory.
- Mean error time: Mean of the error times calculated over all failed trajectories.
- Mean error percentage: Mean of the error percentage calculated over all failed trajectories.
- Mean maximum position error: Mean of the maximum position errors calculated over all failed trajectories.
- Mean maximum angle error: Mean of the maximum angle errors calculated over all failed trajectories.

6.4.1 - MOO Results

Five different MOO problem formulations were tested for the mobile robot trajectory following problem – all the combinations of differential drive base vs omnidirectional base and using a solution space of all arm joint variables, versus a hybrid approach using inverse kinematics for the arm joint solutions when tracking the trajectory and the full solution space for recovery from tracking errors. The performance metrics for all of the experiments are given in Table 6.5, and the distribution of errors over the test trajectories are shown in Figure 6.9. Both omnidirectional problem formulations far outperformed the differential drive problem formulations. This is likely due to the nonholonomic kinematics of the differential drive base making it difficult for the MOO to find optimal solutions in the context of the surrounding trajectory (i.e. balancing the desired linear motion with facing in an optimal direction for the desired motion). This would result in the DDR facing in a direction where it was unable to move in the necessary direction to continue tracking the trajectory, a problem not encountered in the

omnidirectional problem formulation where there were no constraints on the achievable directions of linear motion.

The omnidirectional problem formulation using the full arm joint solution space (omni approx.) performed the best, fully tracking the trajectories in 93.4% of the trials, with an average error time of 2.7s in the cases where the solution did not fully track the trajectory. However, the lack of the base motion constraints in the omnidirectional problem formulation was observed to sometimes result in nontrackable base trajectories, either due to sudden changes in direction or noise (Figure 6.10). For these cases, the trajectories generated by the MOO would need post-processing similar to that done in the SBCO method to allow for proper tracking. This is a drawback, because one of the main appeals for the MOO approach is the possibility for it to be implemented in real-time, and the need for post-processing of the trajectories eliminates this as a possibility.

Table 6.5. MOO trajectory following experiment results.

Experiment	Success Rate	Mean Error Time (s)	Mean Error Percent	Mean Maximum Position Error (m)	Mean Maximum Angle Error (rad)	Overall Maximum Position Error (m)	Overall Maximum Angle Error (rad)
DD Approx	11.0%	9.09	45.47%	0.307	0.086	1.657	0.127
DD Hybrid	23.8%	8.80	43.98%	0.290	0.087	1.541	0.150
Omni Approx	93.4%	**2.69**	**13.45%**	**0.020**	0.084	0.132	0.136
Omni Hybrid	82.8%	2.94	14.72%	0.032	**0.075**	0.083	0.136
Omni Approx Restart	**98.0%**	3.70	18.5%	0.022	0.083	**0.042**	**0.090**

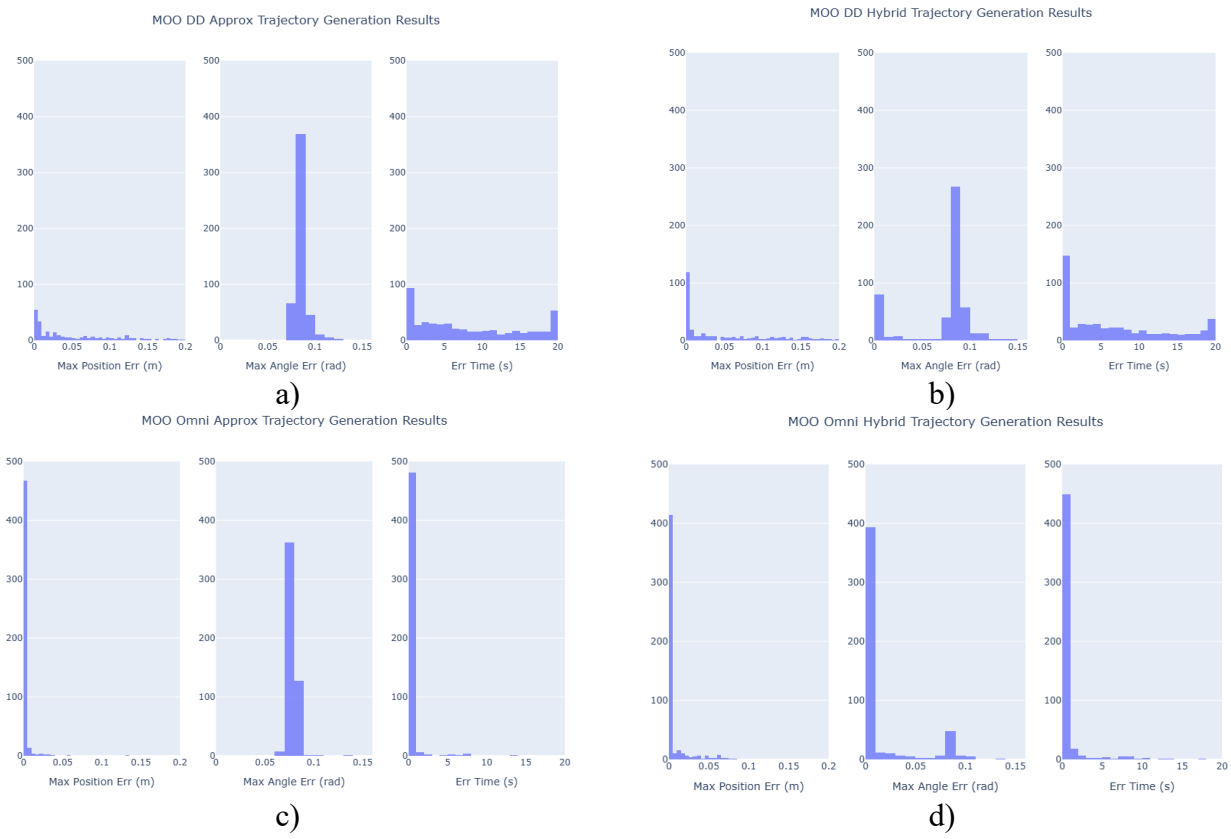


Figure 6.9. MOO trajectory generation results.

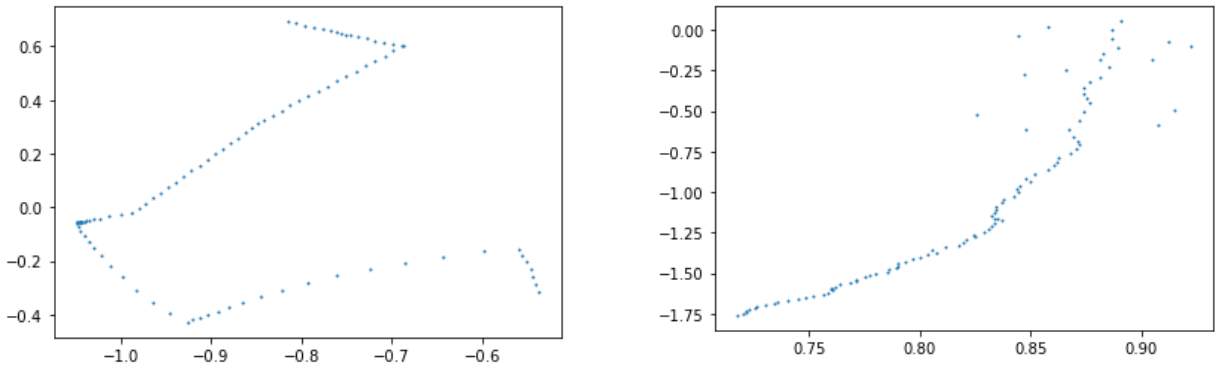


Figure 6.10. Example MOO Omni experiment base trajectories that would require postprocessing due to a) abrupt changes in direction and b) noise.

6.4.2 - SBCO Results

The SBCO method performed very well at generating trajectories that successfully followed the desired end-effector path. Of the 500 test trajectories, the end-effector went above the error thresholds in only four. All of the end-effector tracking errors were violations of the position threshold error, meaning that the end-effector was able to successfully track the desired orientation trajectory in all cases. Table 6.6 gives the performance metrics for the SBCO method for the 500 test trajectories. Figure 6.11 shows the distributions of the maximum position and angle errors over the trajectories.

Table 6.6. SBCO trajectory following experiment results.

Success Rate	Mean Error Time (s)	Mean Error Percent	Mean Maximum Position Error (m)	Mean Maximum Angle Error (rad)	Overall Maximum Position Error (m)	Overall Maximum Angle Error (rad)
99.2%	0.25	1.25%	0.021	0.022	0.033	0.032

Figure 6.11. SBCO method trajectory generation results.

The combination of the combinatorial optimization method to generate a rough feasible trajectory combined with the post-processing procedure to further optimize the trajectory proved to be a robust method for generating feasible mobile robotic arm trajectories for accurate path following. Due to the limited search space of the optimization based on the sampling, the raw trajectories generated from the optimization were often quite noisy and would be very difficult to track. However, the post-processing was able to smooth these raw results into much smoother trajectories for easier tracking, even in the more difficult cases. Figure 6.12 shows one such example, where the post-processing greatly smoothed the raw trajectory while still maintaining path tracking accuracy. Even in the cases where the post-processing was unable to achieve full tracking accuracy, the resulting errors were relatively small and quickly corrected (the maximum error time for any trajectory was only 0.4 s, with a maximum position error of 0.0236m).

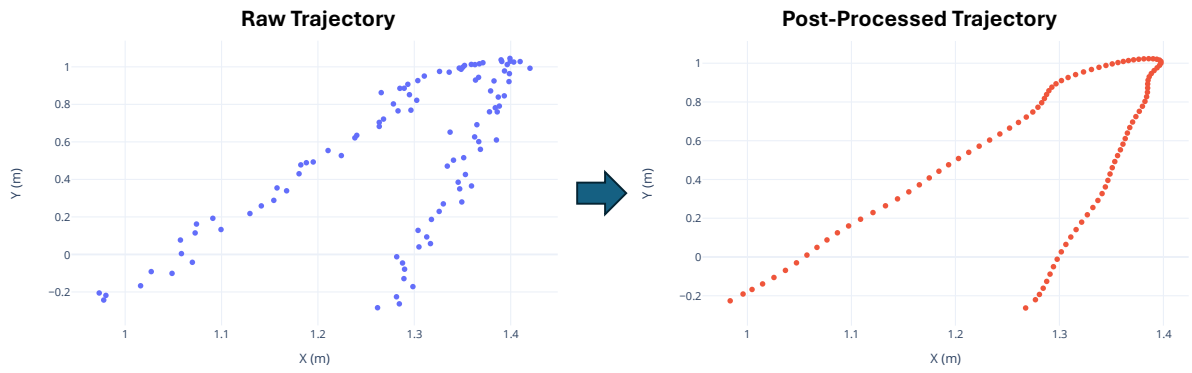


Figure 6.12. Example of post-processing smoothing a noisy raw base trajectory.

6.5 - Discussion

6.5.1 - MOO

The MOO problem formulations using the wheel joint velocities of the DDR as solution variables performed very poorly, with success rates of only 11.0% and 23.8% of tracking the trajectories within the specified error thresholds. As mentioned in the Results section, a large

component of the poor performance of this method is the nonholonomic constraints of the DDR motion. The success rate was drastically improved (to 93.4% and 82.8%) by solving for the XY base trajectory instead of the wheel motions. Formulating the solution space in this manner removes the nonholonomic constraints and creates a simpler search space. However, this comes at the cost of no longer guaranteeing trackable base trajectories for a DDR. Because of the motion constraints of a DDR, it is difficult for them to track noisy or zig-zagging paths. Much better tracking performance can be achieved following smooth, continuous trajectories. Due to this constraint, an objective function was added to the omnidirectional problem formulation to encourage reduction in sudden direction changes with the goal of creating easily trackable trajectories for a DDR. While this was effective in many of the test cases, there were numerous examples where the generated base trajectories would be nontrackable for a DDR and thus would require post-processing to be feasibly implemented.

Another major factor affecting the success of the MOO methods is the forward-stepping nature of this method. While this is attractive for real-time implementation, the complexity of the motion space of the robotic arm means that the feasibility of a trajectory can be highly dependent on the configuration of the robotic arm at the beginning of, and throughout, the trajectory. It is easy for the robotic arm to move into 'dead end' configurations, where it can no longer continue tracking the desired end-effector trajectory, even with the additional DOFs of the mobile base. It is difficult-to-impossible to predict these 'dead end' configurations *a priori* because they are dependent on both the starting configuration and the trajectory to be followed. Avoidance of such 'dead end' configurations was explored using the trajectory restarting experiment, which improved the success rate to 98%, much closer to the SBCO method. However, this method once again eliminates the possibility for real-time implementation of this method.

6.5.2 - SBCO

The SBCO method proved far more effective than the MOO-based methods at successfully generating feasible trajectories. The performance of this method is highly dependent on both the point sampling and post-processing to be able to produce these trajectories. A discussion on each of these steps and their respective effects on the algorithm follow.

6.5.2.1 - Point Sampling Method

The point sampling method defines the search space for the combinatorial optimization, and thus is possibly the most important factor on the performance of the algorithm. Using a sampling-based method allows us to shrink the search space to a manageable size, but this comes at the cost of reducing and simplifying the search space, thus reducing, and possibly eliminating, many high-quality solutions. Both the method of point sampling and the number of points sampled for each trajectory point affect the quality and feasibility of the search space.

There are myriad methods of point sampling that could be employed for this method. The simplest point sampling method is random point selection. This was tried during preliminary testing but often led to a search space with no continuous trajectories, and thus no feasible results. Instead, a forward-stepping of point selection was employed, generating points at each consecutive trajectory point based off of the sampled points at the previous trajectory point. As seen from the results, this proved to be a successful method of point sampling, being able to produce continuous trajectories for all of the test trajectories. However, this method limits the search space based on the points sampled at the first trajectory point and thus does not provide an optimal sampling of all feasible trajectories which would be required to find the overall most optimal solution. Other, more complex methods of point sampling could potentially provide a more comprehensive sampling of the entire search space and provide more optimal trajectories.

The other major factor of the point sampling performance is the number of points sampled at each trajectory point. This is a trade-off between the quality of the representation of the search space with the computational energy needed to do the optimization. A larger number of points will provide a more complete representation of the full search space, but at the cost of increased computation at the combinatorial optimization step. For the results presented in this chapter, a sampling of 150 points was chosen, which provided good results. However, it could be interesting to perform an analysis comparing the algorithm results to the required computational time for various numbers of sampling points to determine the optimal number based on the desired performance. It is also likely that the optimal number of sampling points will vary with the problem definition (i.e. robot configuration and trajectory length and complexity), with higher DOF configurations and more complex trajectories requiring more sampling points to provide satisfactory results.

6.5.2.2 - Dependence on Post-processing and Parameters

The performance of the SBCO method is also highly dependent on the post-processing of the trajectory generated from the combinatorial optimization. Because the search space is reduced to only the sampled points, this discretized search space yields only an approximation of the optimized trajectory. Searching in a discretized space creates noisy trajectories which must be smoothed to create a trackable trajectory. However, this smoothing process must balance preserving the original shape of the trajectory to ensure tracking accuracy with optimizing the joint trajectories to minimize accelerations and jerks and allow for smooth tracking. The method of using Gaussian filtering to perform trajectory smoothing proved to be effective in achieving both of these objectives. It was observed that the smoothing parameter, σ, was important for balancing these objectives. A low value of σ did not provide enough smoothing for good tracking

performance, while too large of a value smoothed the trajectory too much to maintain high tracking accuracy. For the given set of test trajectories, it was found that a σ-value of 3.0 provided the optimal performance over the test set. However, this value could likely be further optimized for a singular trajectory, and the optimal value is most likely dependent on a given trajectory and its complexity.

6.5.3 - Comparison of Methods

Both the MOO and SBCO methods of trajectory generation for a mobile robot have advantages and disadvantages. One of the main advantages of the MOO approach is the potential for it to be implemented in real-time. While in the experiments presented in this chapter the parameters were adjusted to provide better performance, thus increasing the computational time above the threshold for real-time implementation, it is possible that with code optimization, streamlining, and increased computational power this method could be implemented in real-time. This would provide numerous advantages, such as the ability to adapt to real-time changing trajectories and the potential to incorporate dynamic obstacle avoidance. However, the MOO approach also provided a much lower success rate at tracking trajectories, especially when the differential drive wheel velocities were used as solution variables. The performance was much improved by changing the solution space to be the X and Y base velocities, but this came at the cost of producing noisier base trajectories that would require often post-processing to allow for smooth tracking, thus again prohibiting implementation in real-time. Additionally, the method using inverse kinematics of the robotic arm to reduce the search space generally performed better, but this reduces the versatility of the algorithm to robot configurations with a closed-form inverse kinematics solution for the robotic arm. Using the full search space of all the robotic arm joint velocities increases the versatility of this algorithm to be applied to virtually any robot configuration but based on the

results presented in this chapter much more research would need to be done to improve the performance of this approach to a satisfactory level.

The more comprehensive approach of the SBCO method provided much better and more consistent results in generating feasible tracking trajectories. However, the approach of generating the trajectory as a whole, rather than using a forward-stepping method, prohibits real-time implementation. This limits the algorithm to applications where the full trajectory is known beforehand, and there is no need to avoid dynamic obstacles. It could be possible to combine the generated trajectory with a tracking algorithm including real-time obstacle avoidance to address this limitation. The other major drawback of the SBCO method is the computational effort required. Because the algorithm is performing a combinatorial optimization over the entire trajectory, the complexity of the computation increases very quickly with the length of the trajectory, since the number of combinations of trajectory points increases exponentially. This could be addressed by reducing the number of solution points along the trajectory and performing interpolation between the optimized points at the sacrifice of some level of tracking accuracy. As such, this may not be a feasible approach for long trajectories requiring high tracking accuracy. Finally, the SBCO approach was implemented using closed-form inverse kinematic solutions to directly calculate the required robotic arm joint positions needed to achieve a specific end-effector configuration given the relative base position. For robotic arms without closed-form inverse kinematic solutions, a more involved approach may be required to determine the appropriate robotic arm joint trajectories required to achieve high tracking accuracy.

6.5.4 - Including Obstacle Avoidance

Both the MOO and SBCO methods could be modified to include obstacle avoidance while generating trajectories. For the MOO approach, an objective function maximizing the distance

between the robot and obstacles could be added, along with a constraint function preventing collision with any obstacles. To implement this, the position of various points on the robot could be estimated using Jacobians (the same way the end-effector position is estimated), and then the distance between each of these points and any obstacles could be calculated. This approach would work for both avoidance of static obstacles known *a priori*, and dynamic obstacles if implemented in real-time in combination with sensor data providing the locations of moving or unexpected obstacles in the environment. For the SBCO method, obstacle avoidance could be incorporated by generating sample points where the robot does not collide or get within a certain distance of any obstacles in the environment. Provided a close enough spacing of points along the trajectory and a continuous solution is found, this would allow the robot to track the desired trajectory while avoiding any collisions with obstacles. It is likely that adding obstacle avoidance would increase the difficulty of finding a continuous trajectory, so it may be necessary to modify the point sampling method or sample a greater number of points to increase the size of the search space. Since this method is an offline trajectory generation method, it is not capable of providing dynamic obstacle avoidance, unless the trajectories of the obstacles are known *a priori*. However, it may be possible to combine the generated trajectory with a real-time obstacle avoidance control strategy to help mitigate this shortcoming.

Chapter 7 – Controller Design for Mobile Robot Arm with Differential Drive Base

7.1 - Introduction

In the previous chapter, several methods were explored for generating mobile robot arm trajectories that follow a specified time-space end-effector path. The most effective methods generated an XY-base trajectory in combination with a corresponding robotic arm joint trajectory for accurate path tracking. In order to accurately execute these generated trajectories, a controller must be designed which is able to effectively track both the XY-base trajectory and the arm joint trajectories in combination. The mobile robot configuration being explored in this research uses a differential drive mobile base, since this is the most common type of wheeled mobile robot. However, the nonholonomic constraints of a differential drive robot (DDR) configuration create challenges for accurate path tracking, especially for irregular (i.e. non-smooth) paths. In many cases, the trajectories generated for the mobile robot arm create base paths that, while smoothed to some extent, are more difficult to track than the regular polynomic or splined paths normally explored for DDR path tracking. As such, it becomes necessary to design a DDR controller which can accurately track more complex and irregular trajectories than those normally explored.

Additionally, a robotic arm controller must be designed which can track the desired joint trajectories in the context of the mobile base. Because the arm is mounted on the DDR, the orientation of the DDR will affect the end-effector position for a given joint configuration. To mitigate this effect, the arm controller must compensate for the changes in orientation of the DDR base to achieve accurate end-effector path tracking. This can be done by adjusting the trajectory of the waist joint of the robotic arm in real-time based on feedback of the DDR orientation.

Finally, due to the irregular nature of the base trajectories, it is likely there will be errors in the DDR tracking, creating errors in the end-effector tracking even if the arm joint trajectories are tracked perfectly. However, the additional degrees of freedom of the robotic arm present the possibility of compensating for these DDR tracking errors with the robotic arm to achieve more accurate end-effector tracking. In order to accomplish this, the DDR tracking errors must be predicted based on the velocity commands, and then appropriate compensatory control must be implemented in the robotic arm controller.

The fully integrated mobile robot arm controller is comprised of three main components – DDR controller, arm joint controller, and integrated arm error compensation. In this chapter, we present a fuzzy logic controller for DDR path tracking, a proportional arm joint controller with compensation for DDR base motions, and an error compensation method for mitigating base tracking errors by modifying the desired arm joint trajectories in real-time.

7.2 - Background

Both DDR and robotic arm control are widely studied topics with numerous validated approaches. In this section, an overview of relevant DDR and robotic arm control techniques will be provided. It should be noted this is not a comprehensive review of all related literature, but rather a summary of the most common approaches and the most relevant research.

7.2.1 - DDR Controllers

Many different approaches have been proposed for DDR control, including variations and combinations of kinematic control, feedforward control, PID control, NN and ML-based control, sliding-mode control, and more. Additionally, some DDR control research also addresses creating easily-trackable DDR trajectories using techniques such as NURBS [75], B-splines [76], and

Bezier curves [77]. In this section, we will focus on covering proposed DDR control methods, since the goal of this chapter is to design a controller capable of following pre-existing trajectories.

7.2.1.1 - PID-based Controllers

PID is a common and effective method of control for many applications, including DDR controller design. For DDR control, PID control loops can be applied in many configurations, including controlling tracking in XY space or controlling tracking of wheel velocity commands. In [78], a fuzzy PID controller was presented, using fuzzy control to tune the PID gains of a DDR kinematic controller. Thai et al. also used a PID control approach, where the PID gains were adjusted in real-time based on the tracking error [79]. A nonlinear PID control scheme is presented in [80], using 6 control parameters to control both the linear velocity of the DDR and the angular errors and velocity. The team in [81] proposed a cascaded control infrastructure with inner PID control loops for the wheel velocities cascaded with an external feedforward control loop controlling the robot position.

7.2.1.2 - NN and ML Controllers

Many other approaches use neural networks and machine learning algorithms to optimize parameter tuning. Mohamed et al. used a PID neural network (PIDNN) controller for DDR control, tuning the controller parameters using particle swarm optimization [82]. PIDNN is a controller structure using a forward multi-layer neural network where neuron functions represent the PID actions. Abed et al. proposed a neural network fractional order PID controller for DDR control, a variation of traditional PID control which includes powers of the integral and derivative control [83]. They used Enhanced Fruit Fly Swarm Optimization to tune the gains of this more complex controller, achieving the goal of accurate tracking of continuous trajectories while minimizing control effort. In [84], chaotic billiards optimization was used to optimize controller parameter

tuning. Khai et al. used a radial basis function neural network (RBFNN) to create an adaptive controller that operates by using the RBFNN to predict tracking errors and update the kinematic controller parameters accordingly [85].

7.2.1.3 - Other DDR Controllers

Various other DDR controllers have also been proposed, including variations of kinematic controllers, sliding-mode control, fuzzy logic controllers, and more. The research in [86] introduces an inverse kinematics-based proportional feedback controller using a point of interest located away from the axis of rotation to achieve omnidirectional control. Xie et al. propose an improved kinematic controller for better tracking performance [87]. Reference [88] proposes a fuzzy sliding-mode controller for DDR trajectory tracking. Various methods of feedback linearization have been proposed to improve DDR tracking performance, including input-output feedback linearization [89], backstepping-like feedback linearization [90], and set-based receding horizon tracking using feedback linearization [91].

7.2.2 - Robotic Arm Controllers

Many techniques exist for robotic arm control. They can be assigned to two general categories – joint space control and task space control. Joint space control strategies operate by following desired position or velocity trajectories in joint space. Such techniques include PID-based control strategies [92]-[95], feedback feedforward control [96][97], and machine learning-based controllers [98]-[100]. The advantage of joint space control is that low-level control outputs are normally joint torques, creating a linear feedback system in joint space. However, because the kinematics of robotic arms are generally non-linear and tasks are specified in Cartesian space, translating between joint space and task space is a nonlinear transformation that must be performed

outside of the joint space controllers to determine the desired robotic arm joint trajectories that will achieve the desired trajectory in Cartesian space.

For this reason, task space robotic arm controllers provide a desirable alternative to joint space controllers in some cases. Task space controllers operate in Cartesian task space rather than the robotic arm joint space, which eliminates the need for calculating the inverse kinematics of the robotic arm as part of the control procedure. However, the nonlinearity of the robotic arm kinematics creates a nonlinear feedback loop where errors in Cartesian space do not map linearly to joint movements, making feedback control more difficult. Feedback linearization can be used in task space controllers by using the Jacobian and robotic arm inverse dynamics to map task space control outputs to joint torque outputs [101]. By doing this, a linear feedback controller (such as a PID controller) can be designed to track trajectories in task space, and then the linear feedback control outputs are translated into joint control efforts as described. Other task space controllers include task space control with null-space compliance [102], adaptive task space control for uncertainties [103], and fuzzy task-space control [104].

7.3 - Methods

7.3.1 - DDR Controller Design

As evidenced by the background section of this chapter, many different approaches have been proposed to address the DDR control problem. These include PID-based controllers, fuzzy controllers, sliding controllers, and many others. There are 2 main approaches to controller design – kinematic controller design and dynamic controller design. Kinematic controller design considers only the kinematics of the DDR in the design of the control strategy. Dynamic controllers, on the other hand, consider the dynamics of the DDR in the controller design as well. Many proposed DDR controllers integrate both control approaches, implementing a kinematic

controller feeding into a dynamic controller in order to better approximate the real-world control response of the robot. While many of the already proposed controllers have been able to provide good performance tracking continuous, regular trajectories (such as circular, sinusoidal, or spline-based paths), they are not optimized for tracking the irregular trajectories such as those generated in the mobile robot trajectory generation method presented in Chapter 6.

We propose a combined kinematic and dynamic DDR controller infrastructure designed for tracking irregular trajectories. The kinematic controller uses fuzzy controllers to output linear and angular velocity commands based on the next 2 trajectory points. These velocities are then fed into a dynamic controller to approximate the actual control response of the DDR. In the following sections we will present the kinematic and dynamic models of a DDR, followed by the proposed fuzzy kinematic controller design and dynamic controller implementation.

7.3.1.1 - Kinematic Model of DDR

In this section, we will derive the kinematic model of the simple, 3-wheeled configuration of a differential drive robot, consisting of two drive wheels (left and right) and a supporting caster wheel for balance (Figure 7.1). The kinematic model of the DDR assumes minimal wheel slippage, and thus the linear and angular velocities can be determined as functions of the wheel velocities using (7.1) and (7.2), respectively.

$$v = r\left(\frac{\dot{\theta}_r + \dot{\theta}_l}{2}\right) \tag{7.1}$$

$$\omega = r\left(\frac{\dot{\theta}_r - \dot{\theta}_l}{w}\right) \tag{7.2}$$

Alternatively, the required wheel velocity commands to achieve a given linear and angular velocity of the DRR can be computed with (7.3) and (7.4).

$$\dot{\theta}_r = v\frac{1}{r} + \omega\frac{w}{2r} \tag{7.3}$$

$$\dot{\theta}_l = v\frac{1}{r} - \omega\frac{w}{2r} \tag{7.4}$$

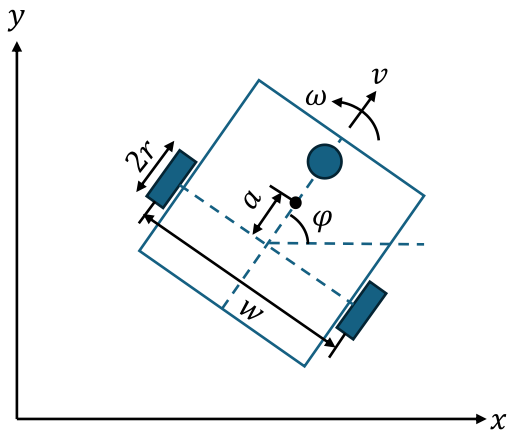


Figure 7.1. Two-wheeled differential drive robot model.

This kinematic model is sufficient for the fuzzy kinematic controller presented in this research since it outputs linear and angular velocity commands. However, some other DDR controller designs operate in XY space, including the controller presented in [81] which is used as one of the benchmark comparison controllers in the controller testing. Typically, the XY kinematic model is derived for a point on the robot offset from the center of rotation along the axis of linear motion (Figure 7.1). The kinematic model can then be derived as a function of v, ω, and φ (the current orientation of the DDR).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} cos\varphi & -asin\varphi \\ sin\varphi & acos\varphi \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{7.5}$$

7.3.1.2 - Dynamic Model of DDR

Including the dynamic model of the DDR in the controller design helps to ensure more accurate tracking of the desired trajectory by taking the dynamics of the robot into account, rather than

solely relying on the kinematic model. The full derivation of the dynamic model from the Euler-Lagrange equations can be found in 0. Here, we use the dynamic model which consists of the linear and angular velocities of the DDR as variables, given by (7.6) and (7.7).

$$\left(m + \frac{2I_w}{r^2}\right)\dot{v} - m_c d\omega^2 = \frac{1}{r}(\tau_r + \tau_l) \tag{7.6}$$

$$\left(I + \frac{w^2}{2r^2}I_w\right)\dot{\omega} + m_c d\omega v = \frac{w}{2r}(\tau_r - \tau_l) \tag{7.7}$$

Table 7.1. DDR dynamic equation variables

Variable	Meaning
m	Total mass of DDR
m_c	Mass of DDR without driving wheels and actuators
d	Distance from center of rotation to center of mass
I_w	Moment of inertia of driving wheel plus motor about wheel axis
I	Total moment of inertia of DDR
τ_r	Right driving wheel torque
τ_l	Left driving wheel torque

7.3.1.3 - Fuzzy Kinematic Controller

A kinematic DDR controller was designed for tracking a trajectory specified by a list of time-space points. The goal of the controller design was to achieve accurate trajectory tracking for irregular trajectories. To this end, the proposed controller takes the next 2 trajectory points following the current position to smooth the path and accomplish better overall tracking performance and less oscillations. Because of the nonholonomic nature of the DDR, it can often be difficult for the robot to recover from tracking errors, with error corrections often resulting in large oscillations in the orientation of the robot around the desired trajectory as it tries to correct

onto the desired path (Figure 7.2). The design of the proposed controller seeks to minimize this oscillatory response while still achieving fast error correction.

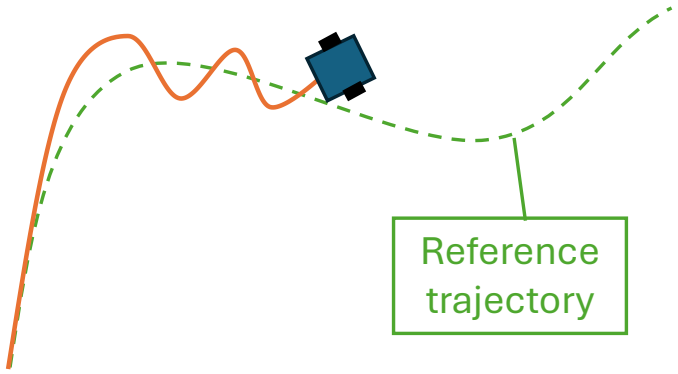


Figure 7.2. DDR oscillations in response to tracking error

It was decided that a fuzzy logic-based controller would be best able to accomplish the desired DDR trajectory tracking objectives. The proposed controller takes the current position and the next two trajectory points as inputs and outputs linear and angular velocity commands, which are then fed into the dynamic controller. The overall kinematic controller Simulink diagram is shown in Figure 7.5. The kinematic controller uses two Mandani type-II fuzzy logic controllers, one for each of the linear and angular velocity commands. First, the raw desired linear and angular velocities are calculated for each of the next two trajectory points using (7.8) and (7.9), where θ is the direction of the vector from the current trajectory point to the desired trajectory point, $\vec{x}_t$ is the location of the desired trajectory point, $\vec{x}$ is the current robot location, and φ is the current robot heading.

$$v_r = \max(0, \|\vec{x}_t - \vec{x}\| \cos(\theta - \varphi)) \tag{7.8}$$

$$\omega_r = \frac{\theta - \varphi}{\Delta t} \tag{7.9}$$

The angular velocity fuzzy logic controller is a two-input, one-output configuration, where the inputs are the raw desired angular velocities for the next two trajectory points, and the output is the final angular velocity command. Both inputs have five membership functions (Z, Guassian, and S-shaped) – big left, small left, zero, small right, and big right. The output has five triangular membership functions – big left, small left, zero, small right, and big right. The membership functions and rule table for the angular velocity fuzzy logic controller are given in Figure 7.3 and Table 7.2, respectively. This fuzzy logic controller was designed to minimize angular oscillations of the DDR to achieve smooth tracking, while still allowing a quick response to large angular tracking errors.

Table 7.2. Rule table for fuzzy angular velocity controller

wRef\wNext	BL	SL	Z	SR	BR
BL	BL	SL	SL	SL	Z
SL	SL	SL	SL	SL	Z
Z	Z	Z	Z	Z	SR
SR	Z	SR	SR	SR	SR
BR	Z	SR	SR	SR	BR

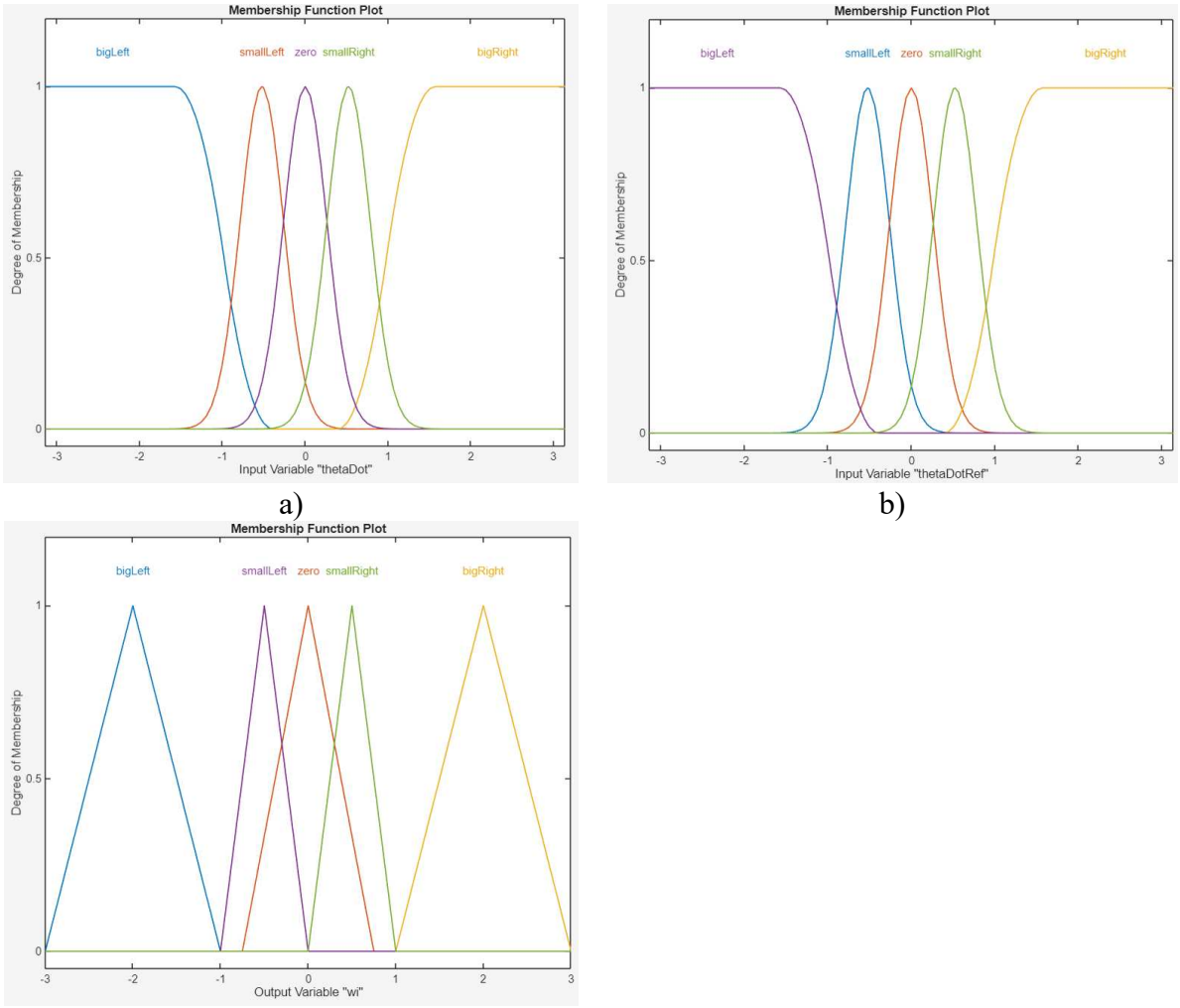


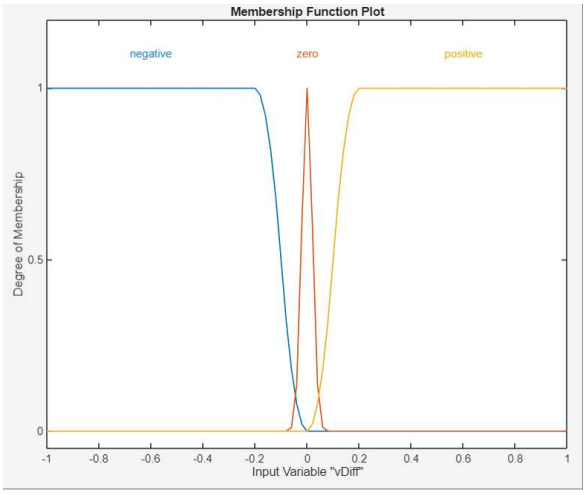
Figure 7.3. Membership functions for the fuzzy angular velocity controller for a) next time step angular velocity input, b) current time step angular velocity input, and c) angular velocity command output.

The linear fuzzy logic controller sets the gain of the linear velocity command. It is a two-input, one-output design, where the inputs are the change in raw linear velocity between the next two trajectory points and the final angular velocity command, and the output is the gain applied to the raw linear velocity command. Both inputs have three membership functions (Z, Gaussian, and S-shaped) – negative, zero, and positive for the change in linear velocity command, and zero, small, and large for the angular velocity command (using the absolute value of the angular velocity

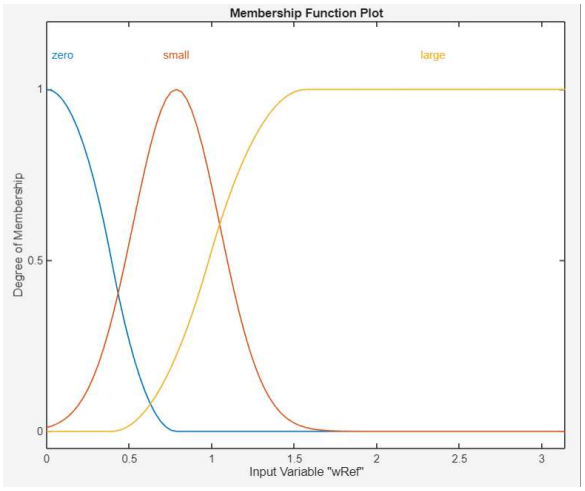
output). The output of the linear velocity gain parameter has three triangular membership functions – zero, small and large. The membership functions for the linear fuzzy logic controller and the rule mapping are shown in Figure 7.4 and Table 7.3, respectively. The linear fuzzy logic controller was designed to optimize the raw linear velocity command based on the future desired linear velocity for smooth tracking and angular errors (i.e. reducing the linear velocity for large angular errors since the DDR is facing in the wrong direction for the desired trajectory, and thus linear motion would increase the trajectory tracking error). The final output linear velocity command from the fuzzy kinematic controller is the product of the raw linear velocity command and the gain output from the fuzzy linear gain controller.

Table 7.3. Rule table for fuzzy linear gain controller

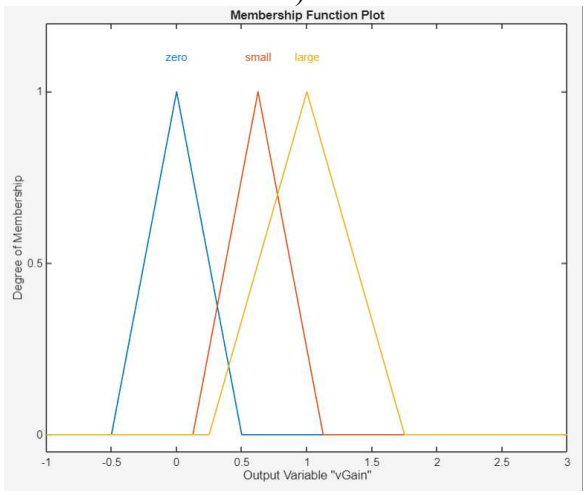
wRef\vDiff	N	Z	P
Z	S	L	L
S	S	L	L
L	Z	S	S



a)



b)



c)

Figure 7.4. Membership functions for the fuzzy linear gain controller for a) change in linear velocity input, b) final angular velocity command input, and c) linear velocity gain output.

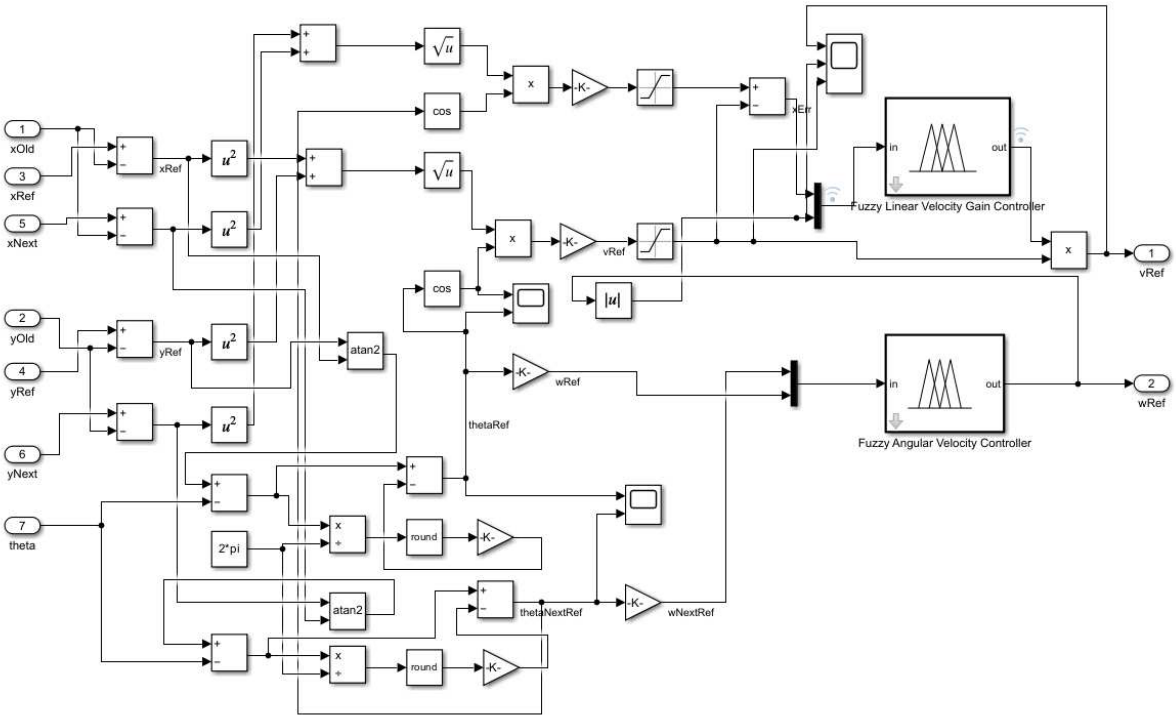


Figure 7.5. Simulink diagram of fuzzy kinematic controller.

7.3.1.4 - Dynamic Controller

The DDR dynamic controller is implemented as a feedback feed-forward controller. This design was chosen because of the discrete nature of the input trajectories. Since the DDR controller is designed to track trajectories specified by discrete time-space points rather than a continuous function in the time domain, the kinematic controller only updates the desired velocities once per time step of the trajectory. However, for accurate tracking of the desired velocities, it is necessary for the dynamic controller to have a much higher update rate. In this design, the feed-forward input is the desired acceleration required to achieve the desired velocity by the next time step. This control input is added to a proportional feedback controller tracking the desired velocity. The feedback feed-forward controller design is applied to both the linear and angular velocity commands output from the fuzzy kinematic controller, which are then fed into the dynamic

equations of the DDR ((7.6) and (7.7)) to calculate the actual DDR velocities. These actual velocities are then used to update the DDR state using the kinematic equations. Figure 7.6 shows the Simulink block diagram of the dynamic controller, and Figure 7.7 shows the Simulink diagram of the overall DDR controller structure.

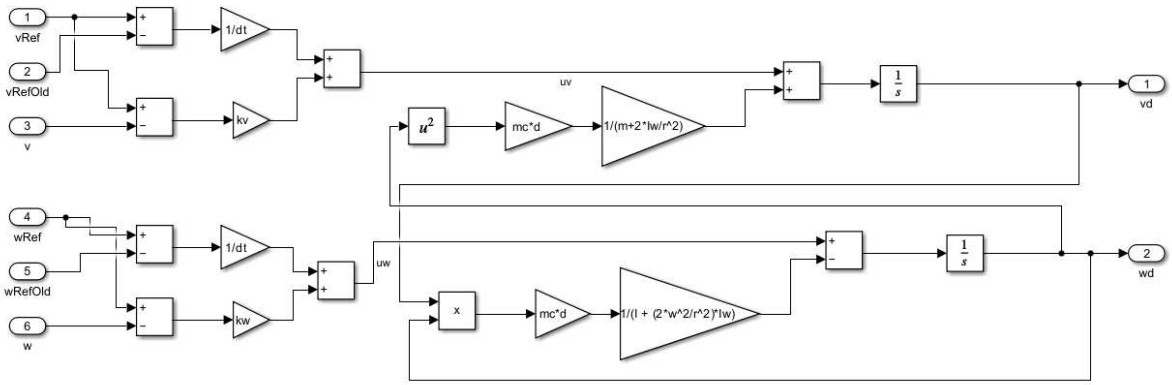


Figure 7.6. Simulink diagram of DDR dynamic controller.

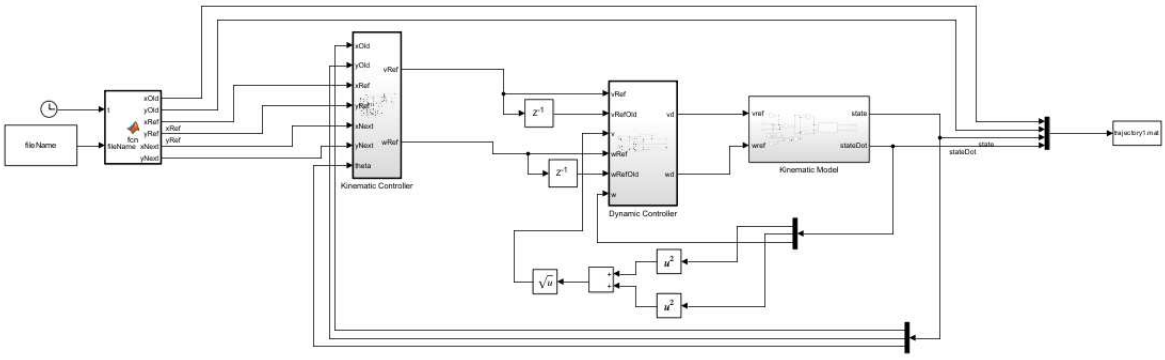


Figure 7.7. Simulink diagram of overall proposed fuzzy DDR controller structure

7.3.1.5 - DDR Controller Testing vs Other Controllers

The proposed DDR controller design was tested on the 500 base trajectories generated using the SBCO method presented in Chapter 6 via implementation in Simulink. These trajectories were generally relatively smooth but often followed irregular shapes and contained rather abrupt changes in direction, making them more difficult for DDR tracking than the regular types of

trajectories usually used for DDR controller testing (circular, sinusoidal, etc.). All of these trajectories are specified by discrete points rather than time-space functions, with points being specified at a rate of 5 Hz, or every 0.2s.

To benchmark the proposed DDR controller, two other DDR controller designs in the literature were selected for testing on the same trajectory set – a fuzzy-PID controller [78] and a proportional kinematic controller [85]. In [78], PID controllers were implemented to track the desired X and Y positions of the DDR, and fuzzy logic was used to set the gains of these X and Y PID controllers based on the tracking error and error rate of change. These X and Y velocity commands were then translated into DDR linear and angular velocities using the robot kinematics. The proportional kinematic controller used a similar approach, adjusting the control inputs based on the tracking errors in X and Y via a proportional controller. These adjusted inputs were then translated into linear and angular velocity commands. Both of these controllers were implemented as kinematic controllers outputting desired velocity commands, which were then fed into the same dynamic controller used for the proposed fuzzy kinematic controller. For all controllers, the kinematic controller update rate was 5 Hz (corresponding to the time step of the discrete trajectory points) and the dynamic controller update rate was 50 Hz.

7.3.2 - Robotic Arm Control

7.3.2.1 - Robotic Arm Kinematics

In this work, the robot arm kinematic model was derived using the Denavit-Hartenberg (DH) parameter model. With this model, a transformation matrix can be constructed for each arm link using the DH parameters, and the final transformation from the world frame to the end-effector frame is accomplished by multiplying these transformation matrices together (7.10).

$$T_0^n = T_0^1 T_1^2 \dots T_{n-1}^n \tag{7.10}$$

7.3.2.2 - Robotic Arm Dynamics

A dynamic model of the robot arm can be constructed using the Euler-Lagrange dynamic equations, given by (7.11).

$$\tau = D(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) \tag{7.11}$$

Where $D(q)$ is the inertial matrix, $C(q, \dot{q})$ is the Coriolis matrix, $g(q)$ is the potential energy vector, and τ is the vector of joint torques.

7.3.2.3 - Arm Controller Design

As outlined in the background (Section 7.2.2 - Robotic Arm Controllers) there are many existing approaches to robotic arm control, from traditional control methods such as PID control to more advanced methods such as NN-based control and others. As such, it was deemed unnecessary to design a novel controller for the robotic arm when existing control methodologies provide more than sufficient tracking performance. The provided trajectory was given in joint space, so the arm controller was also designed to operate in joint space. The chosen robotic arm controller is an inverse dynamics PD controller, where the control input is given by (7.12) as the desired joint acceleration vector.

$$u = k_p e + k_d \dot{e} \tag{7.12}$$

For validation in Simulink, the control input was specified as an acceleration vector. However, for implementation into a real robotic arm, the input should be given in terms of torque, and would thus be computed from the control input vector using (7.11). To calculate the new joint positions, this control input would then be used to solve the dynamic equations of the robotic arm to determine the joint accelerations. However, comparing equations (7.11) and (7.13), we can see

that this has the result of $u = \ddot{q}$, thus making these steps redundant for simulating the tracking performance of the robotic arm controller.

$$\tau = D(q)u + C(q, \dot{q})\dot{q} + g(q) \tag{7.13}$$

7.3.2.4 - Adjusting for Base Rotation

Because the generated arm trajectories do not factor in the differential drive base rotation, this must be compensated for in real-time based on the current configuration of the DDR base. This compensation is done by adjusting the robotic arm waist joint command based on the current orientation of the DDR. Since the arm controller takes inputs of the next two time step joint positions, the predicted orientation at the 2nd time step is approximated using the current angular velocity command of the DDR and the current orientation. The orientation at the active time step is observed from the current DDR state. The adjusted desired waist joint angles are calculated by subtracting the DDR base orientation from the original desired value (7.14). It should be noted that, while the arm trajectories were generated adhering to the waist joint position limit ranging from -π to π radians, this compensation can cause the desired joint position to go outside of this joint range. Technically, the waist joint is a continuous joint so this may not be a concern; however, there are practical/structural limitations on its rotations due to concerns such as wiring wrapping depending on the mechanical design of the robotic arm. In many cases, compensating for the DDR base orientation caused the waist joint to need to perform multiple revolutions where this could become an issue, and should thus be considered when implementing this control scheme.

$$q_{0,adj} = q_{0,d} - \varphi \tag{7.14}$$

7.3.2.5 - Error Compensation

Due to the irregular nature of the test trajectories for the DDR and its tracking limitations due to the nonholonomic constraints, it is expected that there will be a level of tracking error for the

DDR base of the mobile robot arm. To help mitigate these effects at the end-effector, a method was proposed to use the robotic arm to compensate for the DDR base tracking errors during trajectory execution. The proposed method of error compensation consists of two steps – base tracking error prediction, and adjusting the desired arm trajectory to compensate for these errors. The base tracking errors were predicted by simply computing the DDR kinematics for the linear and angular velocity commands output from the fuzzy kinematic controller at each time step. While there are some inaccuracies in this error prediction method due to imperfect tracking of the trajectory commands due to robot dynamics, the magnitude of error in the estimated DDR position was observed to be much lower than the tracking errors. Thus, this served as a simple, efficient, and reasonable means of tracking error prediction.

The error compensation command of the robotic arm was determined using Jacobian-based inverse dynamics. The error vector was computed using (7.15), transforming the error vector into the coordinate frame of the robotic arm by rotating it by the reverse of the DDR orientation angle about the Z-axis. This error vector was then used to calculate the required joint motions using the pseudo-inverse Jacobian (7.16).

$$\vec{e}_{base} = R_{base}\left(\vec{x}_d - \vec{x}_{pred}\right) \tag{7.15}$$

$$\overrightarrow{\Delta q_{comp}} = J^{+}\vec{e}_{base} \tag{7.16}$$

Note that here the standard equation of $\dot{x} = J\dot{q}$ was transformed into position space from velocity space by multiplying each side by the time step, Δt. Using $\Delta x = \dot{x}\Delta t$ and $\Delta q = \dot{q}\Delta t$, we can directly calculate the necessary joint motions from the error vector in position space. Finally, the error compensation vector of joint position changes was added to the original desired joint position vector (7.17) and this adjusted joint position command was then fed into the arm controller.

$$\vec{q}_c = \vec{q}_d + \overrightarrow{\Delta q_{comp}} \tag{7.17}$$

While the inverse Jacobian method can be an effective method for computing the inverse kinematics of a robotic arm, it does not perform well when the robotic arm is in a configuration at or near a singularity. In this case, the element of the Jacobian is close or equal to zero, and thus the inverse Jacobian approaches infinity, leading to unreasonably large joint velocity commands to move in certain directions. Obviously, such large joint commands for error compensation would result in too much disturbance to the tracking trajectory and lead to larger end-effector tracking errors than with no error compensation. To minimize these Jacobian-based singularity effects on the error compensation control outputs, the pseudo-inverse Jacobian elements were capped at a maximum absolute value of 5, and the error compensation change in joint position commands were capped at an absolute value of 0.1 rad. While this limits the maximum achievable error compensation for the robotic arm, it was deemed sufficient for achieving the desired error compensating behavior while limiting the additional errors created from near-singular configurations where the robotic arm was unable to move in the necessary direction to compensate for the base tracking errors.

7.3.2.6 - Integrated Controller Testing

The integrated mobile robot controller was also implemented in Simulink and tested on the 500 SBCO-generated trajectories (base + arm joint trajectories). Testing was performed on the integrated controller both with and without the base tracking error compensation. The goal of the integrated controller testing was to evaluate the end-effector tracking performance for both position and orientation.

7.4 - Results

7.4.1 - DDR Controller Results

The tracking performance of the proposed fuzzy kinematic controller was compared to two other controllers presented in the literature, a proportional kinematic controller [85] and a fuzzy-PID controller [78], over 500 test trajectories. The proposed controller had the best tracking performance of the 3 controllers, with a mean RMS position error of 0.0021m and a mean maximum position error of 0.0136m. This was a 4.8X and 2.4X improvement over the fuzzy-PID controller for the RMS error and maximum error, respectively, and an 18.3X and 6.8X improvement over the proportional kinematic controller for the RMS error and maximum error, respectively. Figure 7.8 and Figure 7.9 show the distribution of RMS and maximum tracking errors, respectively, for each of the controllers over the 500 test trajectories.

Table 7.4. Differential drive controller tracking results

	Proportional Kinematic Controller	**Fuzzy-PID Controller**	**Proposed Fuzzy Kinematic Controller**
Mean RMS Position Err (m)	0.0385	0.0101	**0.0021**
Mean Max Position Err (m)	0.0924	0.0330	**0.0136**

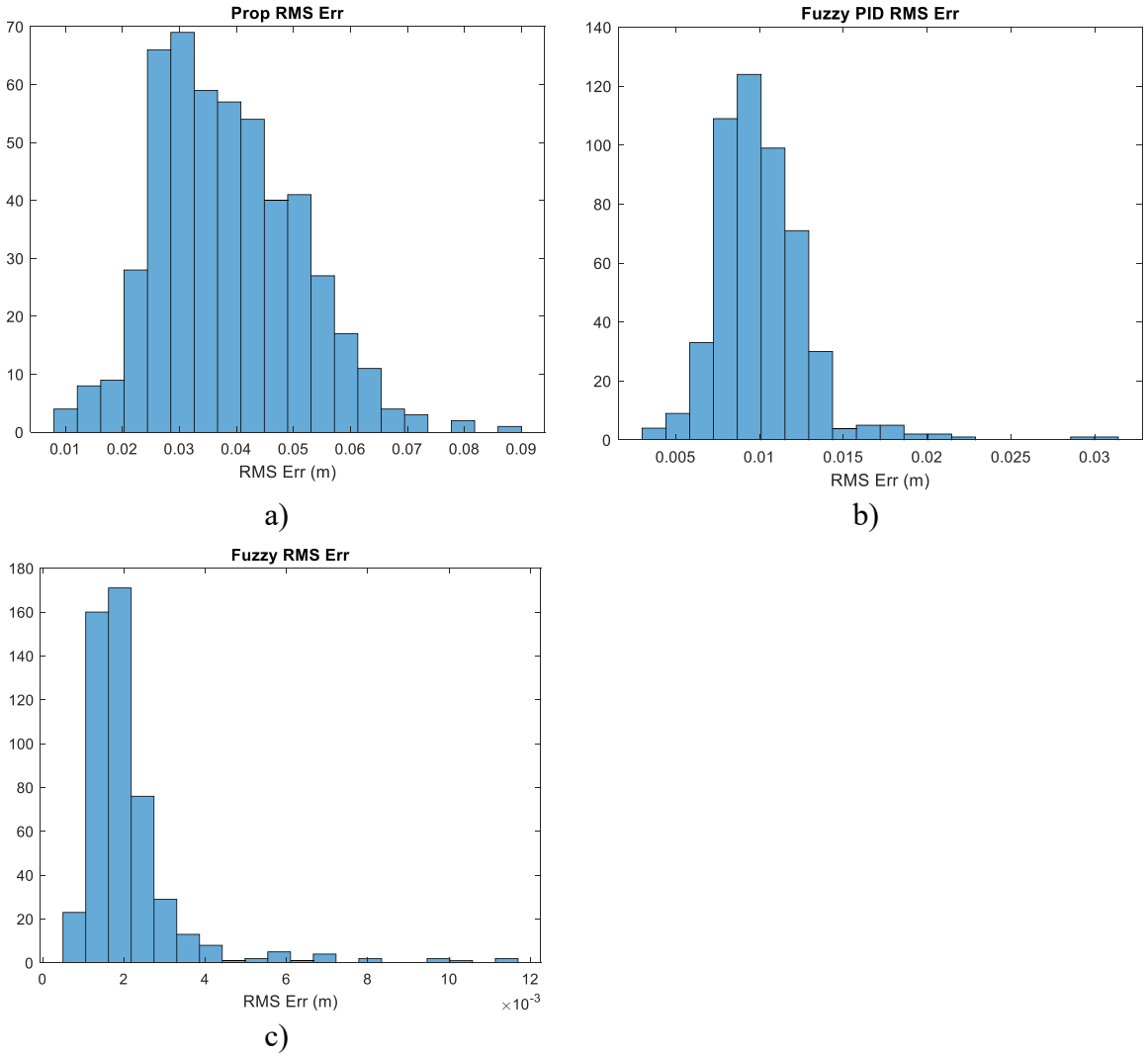


Figure 7.8. Distribution of RMS tracking errors for a) proportional kinematic controller [85], b) fuzzy PID controller [78], and c) proposed fuzzy controller

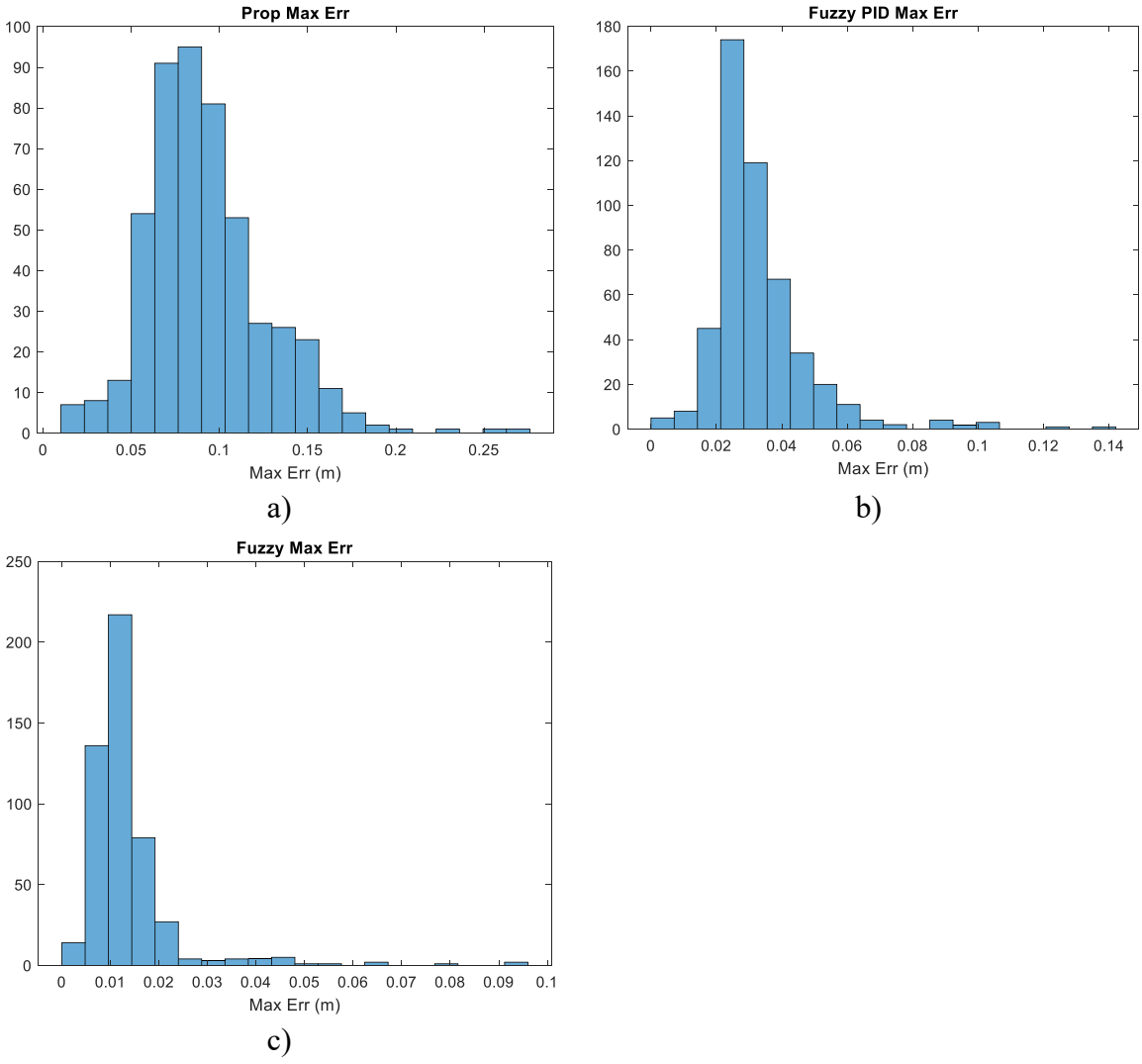


Figure 7.9. Distribution of maximum tracking errors for a) proportional kinematic controller [85], b) fuzzy PID controller [78], and c) proposed fuzzy controller

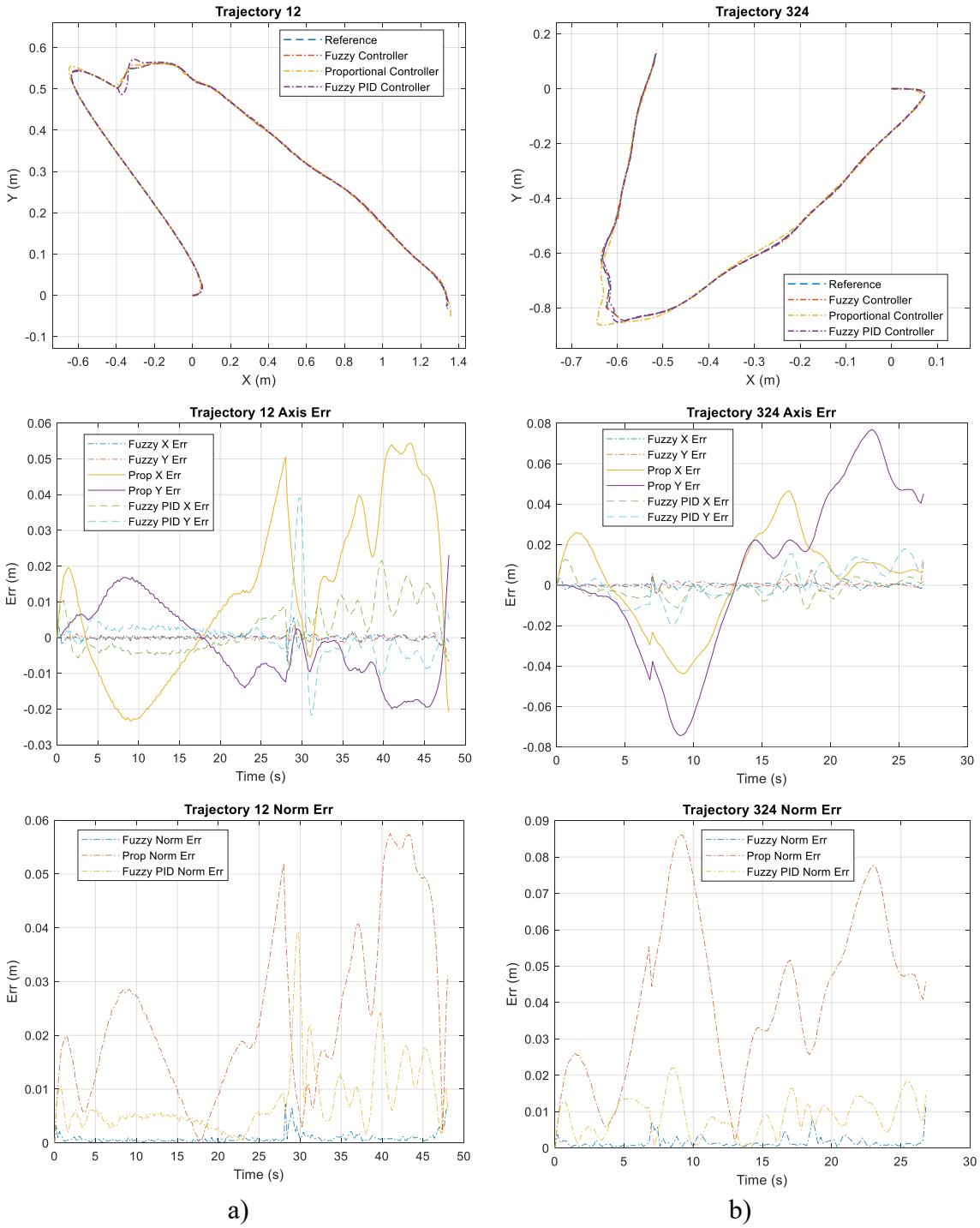


Figure 7.10. Comparison of tracking performance between selected controllers for 2 test trajectories.

Additionally, the angular velocity of the DDR was recorded throughout the test trajectories to analyze the oscillations of the DDR as it tracked the trajectories. The fuzzy PID controller generally had higher angular velocity spikes while the proportional and proposed fuzzy controller typically had smoother angular velocity profiles, although all still had some oscillations as the controllers responded to tracking errors. An example comparing the angular velocity profiles between the three controllers for a sample trajectory is shown in Figure 7.11.

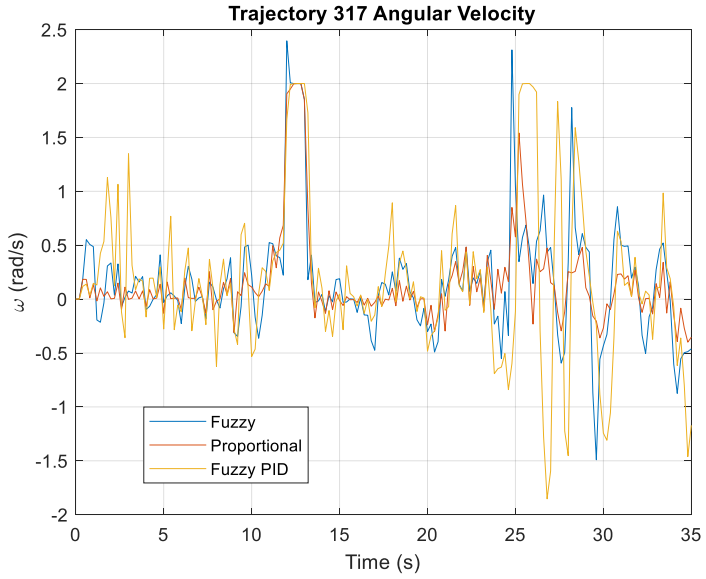


Figure 7.11. Comparison of angular velocity profiles for a sample trajectory

7.4.2 - Integrated Controller Results

The integrated mobile robot arm controller was tested over the same 500 trajectories both with and without the base tracking error compensation. The standard controller achieved mean RMS position and angle errors of 0.0038m and 0.016 rad, respectively, and mean maximum position and angle errors of 0.0106m and 0.0301 rad, respectively. The addition of the error compensation decreased the mean maximum position error to 0.0087m, a decrease of 18%, with minimal effects on the other tracking performance metrics. This indicates that the primary result of including the

error compensation was to reduce the maximum position tracking error. An interesting result is that the error compensation did not have a significant effect on the RMS position error, likely because this error is due to a combination of tracking errors, including joint tracking errors, making it more difficult to compensate for. However, reducing the maximum position tracking error is a valuable result, as this ensures an overall more stable and accurate end-effector tracking. It is expected that the error compensation did not affect the angular tracking errors, since the error compensation was not compensating for angular errors, and this error is fully due to joint tracking errors and variations in the DDR base orientation. However, it should be noted that the base tracking error compensation could have resulted in increased angle errors due to the position-orientation coupling of the robotic arm kinematics, and thus it is an encouraging result that the error compensation showed no significant effect on the angular tracking accuracy.

Table 7.5. Integrated controller tracking results

	Mean RMS Position Err (m)	**Mean Max Position Err (m)**	**Mean RMS Angle Err (rad)**	**Mean Max Angle Err (rad)**
Integrated Controller Only	0.0038	0.0106	0.0116	0.0301
Integrated Controller with Error Compensation	**0.0034**	**0.0087**	0.0117	0.0303

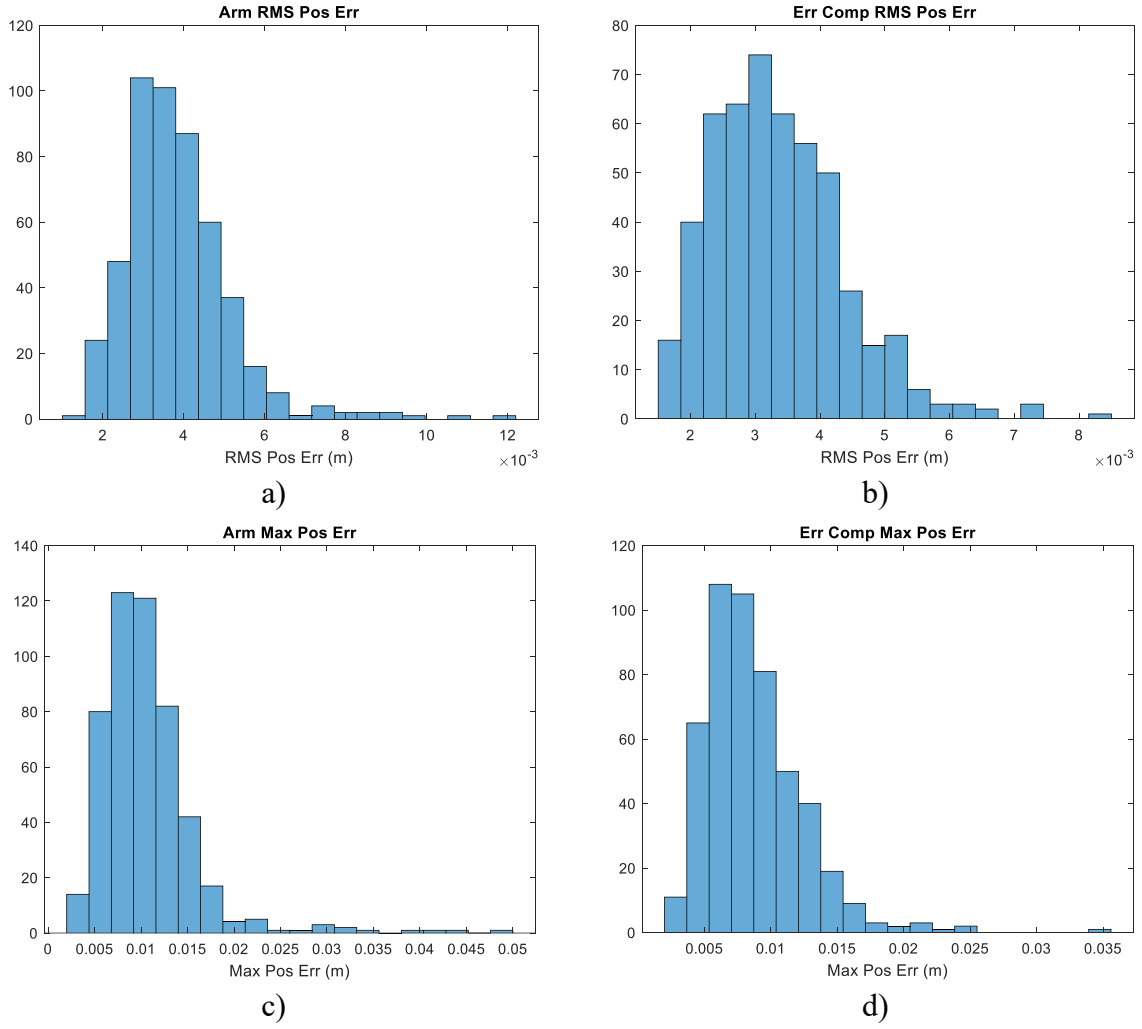


Figure 7.12. Distribution of end-effector position tracking errors for integrated mobile robot controller (a, c) and with added error compensation (b, d).

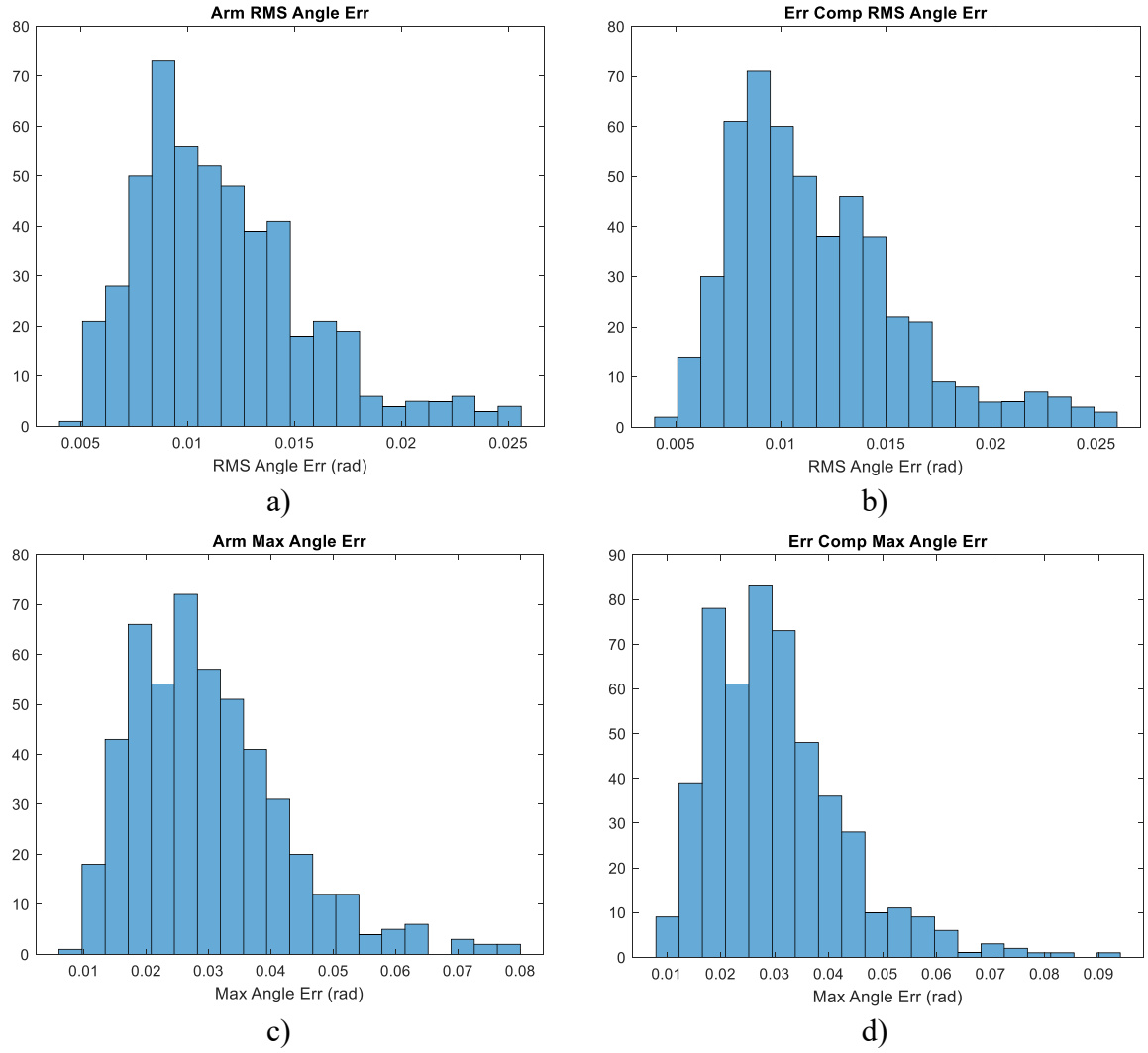


Figure 7.13. Distribution of end-effector orientation tracking errors for integrated mobile robot controller (a, c) and with added error compensation (b, d).

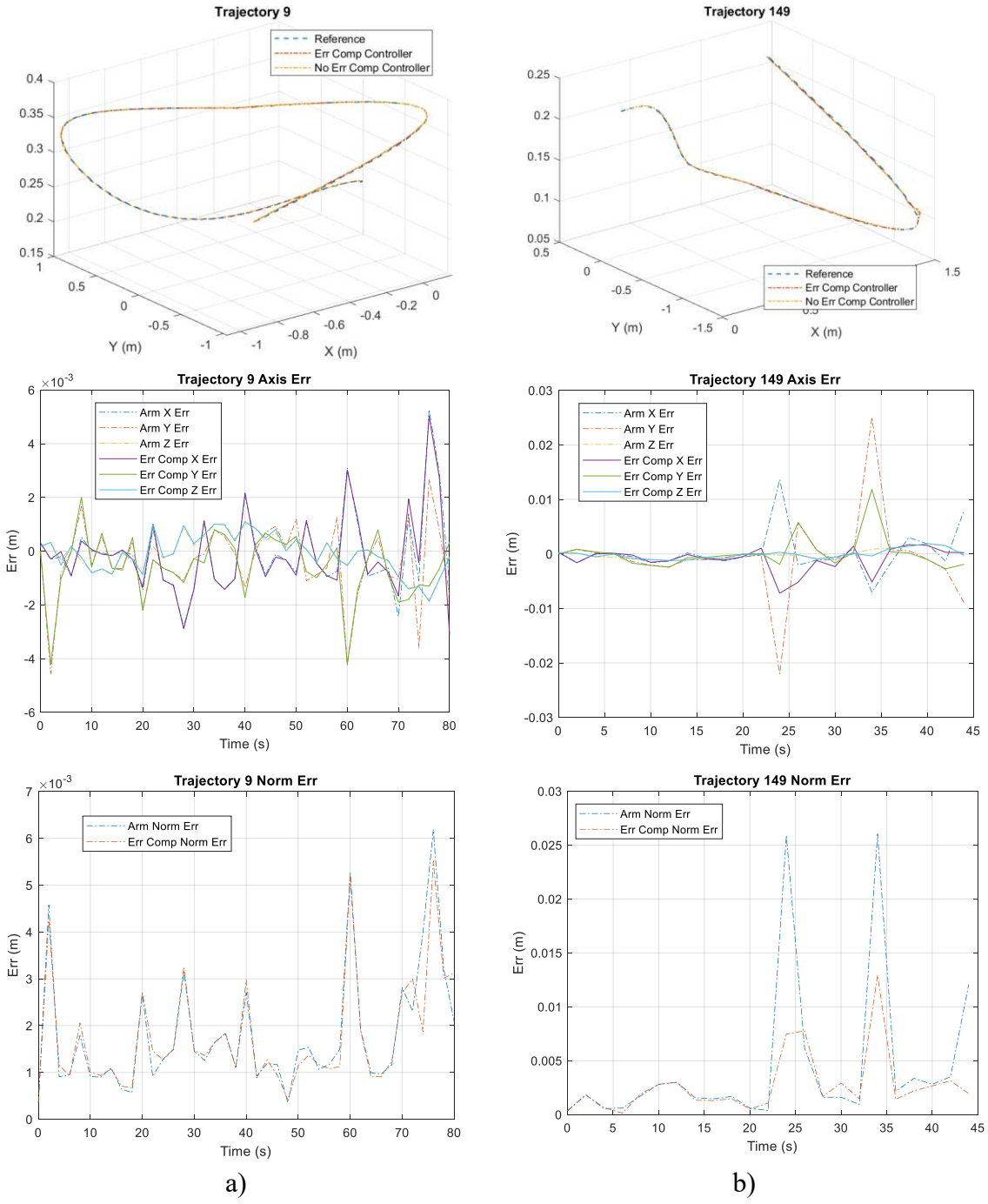


Figure 7.14. Comparison of tracking performance between integrated controller with and without error compensation for 2 test trajectories.

7.5 - Discussion

7.5.1 - DDR Controller

Irregular path tracking is a difficult problem for the DDR due to its nonholonomic constraints, and thus it has not been studied much in the literature. In many cases, DDR trajectories can be designed as continuous and regular paths, and as such more emphasis is made on generating trackable paths instead of designing controllers to track more difficult paths. For the mobile robot control problem, the integrated control of the base and the arm for end-effector trajectory tracking often necessitates an irregular base trajectory to ensure accurate end-effector tracking. To this end, the goal of this research was to design a controller optimized for irregular path tracking. The proposed controller far outperformed other benchmarked DDR controllers in this application, most likely at least partly due to the fact that the other controllers were not optimized for this specific application. Additionally, while basic tuning of the other controllers was performed, it is likely they could have been further optimized for tracking of the test trajectory set. While it is unlikely that additional tuning of the other controllers would have allowed them to exceed the performance of the proposed controller, the performance improvement of the proposed controller over the other controllers may have been decreased.

The proposed DDR controller was able to achieve satisfactory tracking performance of the test trajectories, with a mean RMS error of approximately 2 mm and a mean maximum error of less than 1.5 cm. While this is a very good tracking performance, there were still isolated cases where the controller exhibited quite high tracking errors. Additionally, observation of the angular velocity of the DDR showed significant oscillations throughout the trajectory which, while a common effect of DDR behavior when trying to achieve high accuracy and quick response trajectory tracking, indicates inefficient control effort which could be further minimized by reducing the

angular velocity oscillations. However, it is likely that trying to achieve smoother tracking of trajectories would also result in higher tracking errors, as the DDR would be less responsive in correcting tracking errors. This problem becomes a trade-off between DDR trajectory smoothness and tracking accuracy, and in this case the tracking accuracy was prioritized.

Another point to consider is the additional tracking error that would likely be incurred in real-world implementations. In simulations, the kinematics of the DDR are computed assuming no wheel slippage. However, this is not a realistic assumption for real-world implementations, and thus kinematic-only calculations based on wheel velocities to determine the DDR configuration will result in state estimation errors. This has been addressed in the literature, and integration of sensor-fusion methods using IMUs and other sensor data can greatly improve the accuracy of DDR state estimation in real-world deployments. Such techniques should be considered and adopted where appropriate for any DDR controller implementation in hardware.

7.5.2 - Error Compensation

The configuration of the mobile robot arm setup gives the advantage of higher DOFs (degrees of freedom), which can be leveraged to compensate for the base tracking error. Using the robotic arm to compensate for the DDR base tracking error presents two main challenges – accurately predicting the base tracking error in order to compensate in real-time, and the ability of the robotic arm to move in the desired directions to compensate for the error without introducing other tracking errors, primarily in the end-effector orientation. The method for predicting DDR base tracking error presented in this chapter simply used the velocity commands of the DDR along with the DDR kinematics to predict the DDR position at the next time step. This proved to be fairly accurate, but there were still some errors in the tracking error prediction. The two main sources of error are the linearization of the velocity commands in the prediction (i.e. treating them separately

when the angular velocity command will affect the linear velocity over time by changing its direction) and the imperfect tracking of the velocity commands due to the DDR dynamics. Accounting for these sources of error to improve the DDR tracking error prediction accuracy would require much more complex models for error prediction, which could prove difficult to train and would certainly increase computational intensity. The method presented in this chapter proved accurate enough to reduce the maximum tracking errors, which is one of the most important metrics for trajectory tracking since the goal of trajectory tracking is usually to stay below specified error thresholds.

The other difficulty with using the robotic arm to compensate for the base tracking errors is the ability of the robotic arm to actually be able to move in the required directions to compensate for the predicted error. Robotic arms suffer from singularities in certain configurations, and thus the current robot arm configuration determines the arm's error compensation ability. In some cases, the arm is simply unable to move in the required direction for error compensation due to its current configuration. Additionally, the coupling of position and orientation of a robotic arm also increases the difficulty of performing position compensation without affecting the orientation tracking. Again, this will be determined by the configuration of the robotic arm and whether it is able to move in the desired directions without changing the orientation of the end-effector. Both of these factors, along with joint tracking errors in the arm controller, help explain why the addition of error compensation does not fully eliminate the end-effector tracking error.

Chapter 8 – Conclusion

Mobile robots and robotic arms are some of the most commonly-used robots due to their utilitarian functionalities. Mobile robots have the potential to navigate autonomously and can be used in a variety of applications ranging from delivery to exploring and operating in dangerous environments. Robotic arms can perform manipulation tasks, making them highly useful in manufacturing operations, surgical applications, and more. Separately, both mobile robots and robotic arms are able to perform a wide range of tasks. Together, the range of potential applications greatly increases, creating a much more powerful robotic platform.

The problem of controlling a mobile robot arm becomes more difficult than either control problem individually, especially when considering integrated control where the mobile base and robotic arm are controlled synchronously. The different stages of task execution for a mobile robot arm must also be considered. Generally, this can be divided into 2 general stages: navigation and task execution. In this dissertation, solutions were presented to both parts of the mobile robot control problem. First, navigation strategies were explored, starting with Bluetooth-based target localization, and followed by APF-based navigation strategies optimized for a mobile robot arm application. Then, task execution strategies were addressed with real-time MOO-based coordinated trajectory generation and sampling-based combinatorial optimization for path following trajectory generation. Finally, an integrated DDR and robotic arm controller design was proposed for following these generated trajectories with minimal tracking error. A summary of the contributions of this dissertation follows.

8.1 - Bluetooth-based Target Localization (Chapter 2)

A novel Bluetooth AoA homing controller was developed for a mobile robot, enabling navigation to a target object of unknown location using the AoA and RSSI of the Bluetooth signal. To our knowledge, this was the first work to address the problem of autonomous navigation to a target of previously unknown location. Such a capability has a wide range of potential applications, from object retrieval in dangerous environments to search and rescue applications. The use of Bluetooth technology for the target localization integrates easily in most scenarios since Bluetooth technology is already relatively ubiquitous, and thus additional sensors or technology infrastructure are not required to implement the proposed target localization strategy. Controller testing via simulations showed accurate convergence to the target location, even in the presence of measurement noise, with an MAE of 0.12m or less, depending on the level of noise. The versatility of the controller was proven through successful navigation to target locations in the presence of obstacles, indicating its compatibility with state-of the-art autonomous navigation algorithms.

While the control algorithm was proven successful during simulation testing, real-world hardware implementation failed to achieve as successful of results. This was mainly due to two factors – inaccuracies in the AoA sensor readings (especially for angles outside of the front 180°) and errors in pose estimation of the differential drive robot due to wheel slippage. As a relatively new technology, it is expected that the accuracy of Bluetooth AoA sensor algorithms will be improved in the future. Additionally, an off-the-shelf AoA antenna array with embedded software was used, so it is likely better results could be achieved by optimizing the angle predictions using the raw data, as existing research has shown. There is much existing research on robot localization, and techniques such as sensor fusion and SLAM could mitigate errors due to incorrect robot pose

estimation. Future work could investigate the potential challenges with implementation of the controller framework into hardware, especially regarding the accuracy and range limitations of Bluetooth AoA.

8.2 - APF-based Navigation Strategies for Mobile Robot Arm (Chapters 3 and 4)

A modified artificial potential field (APF)- based navigation strategy was developed in the context of mobile robot arm control. The main objective of this research was to develop a navigation strategy capable of guiding the robot along a near-optimal path to the vicinity of the desired task, considering both the optimal location of the robot for performing the task, and the difficulty of navigation to a given location. A modified potential field was proposed using a grid-based mapping and weighting different grid points based on their optimality for performing a task in a given location. This creates a distributed attractive force with higher attraction at more optimal locations while also creating a potential field through the environment which considers navigation difficulty, thus balancing the two objectives.

While APF-based navigation could be modified to address this more complex navigational problem, it still suffers drawbacks such as local minima which can prevent the robot from reaching the target location. In addition to the modified potential function, the APF algorithm was further optimized to include two other modifications: (i) a modified obstacle influence distance to reduce repulsive potentials preventing passage through narrow areas in the environment; and (ii) creation of a skeleton highway map throughout the environment which is engaged to guide the robot along near-optimal paths to the target. The result of these modifications was far more successful and efficient navigation than using the original APF algorithm, especially in more complex environments. Experiments proved the effectiveness of both the optimized APF-HW algorithm and the ability of the modified potential function to navigate the mobile robot arm to an optimal

starting position for a manipulation task while considering both the optimality of the location for task execution and the difficulty of navigating to that location.

8.3 - Trajectory Generation for Mobile Robot Arm Integrated Control (Chapters 5 and 6)

In Chapters 5 and 6, the problem of trajectory generation for integrated control of a mobile robot arm was addressed. Specifically, the goal was to develop a method to generate trajectories utilizing coordinated movements of the mobile base and robotic arm together to achieve better performance and range of applicability than would be possible by controlling the mobile base and robotic arm separately. These advantages include a virtually infinite working field in XY and improved energy efficiency among others. The contribution of this section is a trajectory generation method which effectively produces smooth, coordinated mobile robot arm trajectories capable of following longer and more complex end-effector trajectories than by controlling the mobile base and robotic arm in isolation.

In the context of performing a manipulation task, the trajectory generation problem was divided into two steps: (i) approach trajectory (moving from the robotic arm home position to the task trajectory starting configuration), and (ii) task trajectory following (tracking the specific end-effector trajectory for the task). The approach trajectory generation method presented utilizes a real-time MOO problem formulation to move the end-effector into the desired starting configuration for the task trajectory. The presented method was effective at reaching the desired starting configuration in 100% of the tested cases, reaching the desired configuration in less than 30s 77% of the time, demonstrating the efficiency of the method.

The original goal was to formulate a real-time method for task trajectory following as well, using a similar MOO-based approach. However, the complexity of the control space of the mobile robot arm proved difficult to implement a real-time trajectory generation method. This was mainly

due to the constraints of the control space imposed by robotic arm joint limits, which resulted in the arm sometimes reaching 'stopper' configurations where it could no longer track the desired end-effector trajectory when generating the trajectory in a forward-stepping method in real-time. It was also observed that the success of trajectory tracking was highly dependent on the starting configuration of the mobile robot arm, where successful starting configuration were dependent on the trajectory to be tracked, and thus not easily predictable. Based on this limitation, an alternate approach was developed, generating the trajectory as a whole rather than using a real-time forward-stepping method. This successful method uses point sampling at each trajectory point and then optimizes the trajectory using a combinatorial optimization strategy. Post-processing of the raw trajectories produced smooth, trackable trajectories with accurate end-effector tracking for over 99% of the test trajectories. Additionally, generating the mobile base path in XY space proved far more effective than generating wheel joint velocities due to the nonholonomic constraints of a differential drive configuration. This also makes accurate tracking easier to implement because wheel slippage will not affect the desired base trajectory when tracking in XY space, but it would cause errors if tracking was in wheel joint space. While this method cannot be implemented in real-time, it presents a feasible method of tracking arbitrary end-effector trajectories using coordinated mobile robot arm motion – a novel contribution to robot control.

A reinforcement learning-based approach was also explored as a solution to the mobile robot arm trajectory generation problem, since this is another increasingly common solution to robotic control problems. While the reinforcement learning method was able to generate trajectories to a specific end-effector configuration, it was unable to learn a generalized control strategy that could be implemented in real-time. Additionally, the learning process for successfully reaching a given end-effector configuration was much more time-intensive than any of the other trajectory

generation methods presented in this dissertation, and thus reinforcement learning was not deemed a suitable solution at this time.

Ideally, an integrated control strategy could be implemented in real-time for both the approach trajectory and task trajectory following problems, as this would allow for real-time adjustments in the desired trajectory and easier integration of obstacle avoidance. In this dissertation, only the approach trajectory generation can be implemented in real-time, and the task trajectory must be generated offline due to the limitations discussed. Future work could address this shortcoming and potentially explore different methods of online trajectory generation for task trajectory tracking. Additionally, the presented methods do not consider obstacle avoidance as part of the trajectory generation. Some methods for incorporating obstacle avoidance were discussed, but none were tested or implemented. This is another avenue of future research which would greatly expand the capabilities of the proposed methods.

8.4 - Controller Design for Differential Drive Mobile Robot Arm (Chapter 7)

The final topic of this dissertation was designing a controller to track the integrated mobile robotic arm trajectories. The major contributions of this chapter were a novel differential drive robot controller design optimized for tracking irregular trajectories and an overall integrated mobile robot arm controller infrastructure including end-effector error compensation. The proposed DDR controller used two fuzzy logic controllers to command the desired linear and angular velocities of the DDR, respectively, based on the next two trajectory points. This controller design achieved far superior tracking of irregular trajectories compared to other proposed DDR controllers in literature, reducing maximum tracking errors by up to 2.4X and RMS tracking errors by up to 4.8X.

The integrated controller architecture combined the proposed DDR controller to track a given XY base trajectory with an inverse dynamics PD arm joint controller to achieve accurate end-effector tracking. This control architecture included base orientation compensation in the robotic arm to mitigate the effects of the base orientation on the end-effector tracking. Additionally, an error compensation method was proposed using the inverse Jacobian of the robotic arm to compensate for predicted base tracking errors from the DDR controller velocity commands. The addition of this error compensation was able to reduce the mean maximum end-effector tracking errors by 18%.

The integrated controller design was able to provide satisfactory end-effector tracking results in almost all cases. Due to the nonholonomic constraints of the DDR, some large tracking errors were still observed from base tracking errors when the base trajectory had abrupt changes in direction. For the given robot configuration, the tracking performance was very good. Some potential solutions to this problem include modifications to the base trajectory generation to further reduce the difficulty of tracking or using a different mobile base configuration, such as an omnidirectional wheeled robot. Future research could explore these or other options to further improve the end-effector trajectory tracking control of a mobile robot arm.

8.5 - The Greater Context

Overall, this dissertation provides proposed methods for all stages of coordinated mobile robot arm control in task execution, which can be used as a starting point for even more advanced methods of mobile robot control. As technology progresses and robots become more capable, reliable, and autonomous, robots will become even more ubiquitous in society. The mobile robotic arm platform has great potential to be applied to a wide range of applications due to the complementary capabilities of mobile robots and robotic arms. These applications could range

from manufacturing to service robots to space exploration and more. However, more complex applications for robotics also require more sophisticated and advanced control strategies.

A major factor contributing to the advancement of autonomous robots is the growing capability repertoire of machine learning and artificial intelligence. Already, much research is being done in this space, exploring how to integrate AI and ML into robotics to create better autonomy and more developed capabilities than can be achieved with more traditional control strategies alone. Hybrid control strategies combining proven and efficient traditional control strategies with higher-level ML and AI-based control architectures have the potential to create robust systems capable of high levels of autonomy. However, ML models and more generalized AI systems can be very complex, highly computationally expensive, energy-intensive to train, and difficult to explain – sometimes outputting unexpected results which could be highly dangerous in safety-critical applications. At minimum, these systems likely require intelligent ablation approaches to prevent hallucinations, performance issues, and susceptibility to malware: all of this can be guided by expertise-driven approaches such as those described in this dissertation. These current AI and ML shortcomings, therefore, highlight the need for expertise in developing these systems with an awareness of the limitations, and the continued need for more traditional control approaches such as those presented in this dissertation. These control systems can provide a foundation for building more sophisticated, hybrid systems that are capable of more autonomy and able to perform more complex tasks.

References

[1] D. Zhang, J. Han, G. Cheng, and M. Yang, "Weakly supervised object localization and detection: a survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5866-5885, Sep. 2022.

[2] R. Lui, G. Huskic, and A. Zell, "Dynamic objects tracking with a mobile robot using passive UHF RFID tags," in 2014 IEEE/RSJ Int. Conference on Intelligent Robots and Systems, Chicago, IL, USA, 2014, PP. 4247-4252.

[3] C. Duan, X. Rao, L. Yang, and Y. Liu, "Fusing RFID and computer vision for fine-grained object tracking," in IEEE Conference on Computer Communications, Atlanta, GA, USA, 2017.

[4] Y. Xianjia, L. Qingqing, J. P. Queralta, J. Heikkonen, and T. Westerlund, "Applications of UWB networks and positioning to autonomous robots and industrial systems," in 2021 10th Mediterranean Conference on Embedded Computing, Budva, Bondenegro, 2021.

[5] A. Ledergerber, M. Hamer, and R. D'Andrea, "A robot self-localization system using one-way ultra-wideband communication," in 2015 IEEE/RSJ Int. Conference on Intelligent Robots and Systems, Hamburg, Germany, 2015, pp. 3131-3137.

[6] Y. Cao, C. Yang, R. Li, A. Knoll, and G. Beltrame, "Accurate position tracking with a single UWB anchor," in 2020 IEEE Int. Conference on Robotics and Automation, Paris, France, 2020, pp. 2344-2350.

[7] W. Shule, C. M. Almansa, J. P. Queralta, Z. Zou, and T. Westerlund, "UWB-based localization for multi-UAV systems and collaborative heterogenous multi-robot systems," in 15th Int. Conference on Future Networks and Communications, Leuven, Belgium, 2020.

[8] S. Zhou, and J. K. Pollard, "Position measurement using Bluetooth," *IEEE Trans. on Consumer Electronics*, vol. 52, no. 2, pp. 555-558, May 2006.

[9] U. Bandara, M. Hasegawa, M. Inoue, H. Morikawa, and T. Aoyama, "Design and implementation of a Bluetooth signal strength based location sensing system," in IEEE Radio and Wireless Conference, Atlanta, GA, USA, 2004, pp. 319-322.

[10] Y. Wang, Q. Ye, J. Cheng, and L. Wang, "RSSI-based Bluetooth indoor localization," in Int. Conf. on Mobile Ad-hoc and Sensor Networks, Shenzhen, China, 2015, pp. 165-171.

[11] G. Kumar, R. Gupta, and R. Tank, "Phase-based angle estimation approach in indoor localization system using Bluetooth low energy," in Int. Conf. on Smart Elect. and Comm., Trichy, India, 2020, pp. 904-912.

[12] Z. Hajiakhondi-Meybodi, M. Salimibeni, K. N. Plantaniotis, and A. Mohammadi, "Bluetooth low energy-based angle of arrival estimation via switch antenna array for indoor localization," in IEEE Int. Conf. on Information Fusion, Rustenburg, South Africa, 2020, pp. 1-6.

[13] A. Khan, S. Wang, and Z. Zhu, "Angle-of-arrival estimation using an adaptive machine learning framework," *IEEE Communications Letters*, vol. 23, no. 2, pp. 294-297, Feb. 2019.

[14] G. Pau, F. Arena, Y. E. Gebremariam, and I. You, "Bluetooth 5.1: an analysis of direction finding capability for high-precision location services," *Sensors*, vol. 21, no. 3589, 2021.

[15] M. Cominelli, P. Patras, and F. Gringoli, "Dead on arrival: an empirical study of the Bluetooth 5.1 positioning system," in WiNTECH, Los Cabos, Mexico, 2019, pp. 13-20.

[16] P. Boccadoro, D. Colucci, V. Dentamaro, L. Notarangelo, and G. Tateo, “Indoor localization with Bluetooth: a framework for modelling errors in AoA and RSSI,” in IPIN WiP Proceedings, Lloret de Mar, Spain, 2021.
[17] V. Vasilopoulos, O. Arslan, A. De, and D. E. Koditschek, “Sensor-based legged robot homing using range-only target localization,” in Int. Conf. on Robotics and Biomimetrics, Macau SAR, China, 2017, pp. 2630-2637.
[18] S. Haykin, “Self-Organizing Maps,” in Neural Networks and Learning Machines, 3rd ed., Upper Saddle River, NJ, USA: Pearson Ed. Inc., 2009, pp. 456-465.
[19] C. Warren, "Global path planning using artificial potential fields," in 1989 IEEE International Conference on Robotics and Automation, Scottsdale, AZ, 1989 pp. 316-321.
[20] M. Weigl, B. Siemiątkowska, K. A. Sikorski, and A. Borkowski, “Grid-based mapping for autonomous mobile robot,” *Robotics and Autonomous Systems*, vol. 11, pp. 13-21, 1993.
[21] C. Cai, and S. Ferrari, “Information-driven sensor path planning by approximate cell decomposition,” *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 39, no. 3, pp. 672-689, June 2009.
[22] O. Takahashi, and R. J. Schilling, “Motion planning in a plane using generalized Voronoi Diagrams,” *IEEE Trans. on Robotics and Automation*, vol. 5, no. 2, pp. 143-150, April 1989.
[23] P. Bhattacharya, and M. L. Gavrilova, “Roadmap-based path planning,” *IEEE Robotics & Automation Magazine*, pp. 58-66, June 2008.
[24] M. G. Park, and M. C. Lee, “Artificial potential field based path planning for mobile robots using a virtual obstacle concept,” in IEEE/ASME Int. Conf. on Adv. Intelligent Mechatronics, Kobe, Japan, 2003, pp. 735-740.
[25] F. Bounini, D. Gingras, H. Pollart, and D. Gruyer, “Modified artificial potential field method for online path planning applications,” in IEEE Intelligent Vehicles Symp., Redondo Beach, CA, USA, 2017, pp. 180-185.
[26] I. Al-Taharwa, A. Sheta, and M. Al-Weshah, “A mobile robot path planning using genetic algorithm in static environment,” *Journal of Comp. Sci*., vol 4, no. 4, pp. 341-344, 2008.
[27] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright, H. M. Tai, in Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004 (2004), vol. 2, pp. 1338–1345.
[28] A. Saffiotti, “The uses of fuzzy logic in autonomous robot navigation,” *Soft Computing*, vol. 1, pp. 180-197, 1997.
[29] H. Omrane, M. S. Masmoudi, and M. Masmoudi, “Fuzzy logic based control for autonomous mobile robot navigation,” *Computational Intelligence and Neuroscience*, vol. 2016, 2016, doi:10.1155/2016/9548482.
[30] N. H. Singh, and K. Thongam, “Neural network-based approaches for mobile robot navigation in static and moving obstacles environments,” *Intelligent Service Robotics*, vol. 12, pp. 55-67, 2019.
[31] C. Luo, and S. X. Yang, “A bioinspired neural network for real-time concurrent map building and complete coverage robot navigation in unknown environments,” *IEEE Trans. on Neural Networks*, vol. 19, no. 7, pp. 1279-1298, July 2008.
[32] B. K. Patle, A. Pandey, A. Jagadeesh, and D. R. Parhi, “Path planning in uncertain environment by using firefly algorithm,” *Defence Technology*, vol. 14, pp. 691-701, 2018.
[33] M. Brand, and X. Yu, “Autonomous robot path optimization using firefly algorithm,” in Int. Conf. on Machine Learning and Cybernetics, Tianjin, 2013, pp. 1028-1032.

[34] Y. Qin, D. Sun, N. Li, and Y. Cen, "Path planning for mobile robot using the particle swarm optimization with mutation operator," in Int. Conf. on Machine Learning and Cybernetics, Shanghai, 2004, pp. 2473-2478.
[35] H. S. Dewang, P. K. Mohanty, S. Dundu, "A robust path planning for mobile robot using smart particle swarm optimization," *Procedia Computer Science*, vol. 133, pp. 290-297, 2018.
[36] B. K. Patle, G. Babu L, A. Pandey, D. R. K. Parhi, and A. Jagadeesh, "A review: on path planning strategies for navigation of mobile robot," *Defence Technology*, vol. 15, pp. 582-606, 2019.
[37] Z. Lin, M. Yue, G. Chen, and J. Sun, "Path planning of mobile robot with PSO-based APF and fuzzy-based DWA subject to moving obstacles," *Trans. of the Institute of Measurement and Control*, vol. 44, no.1, pp. 121-132, 2022.
[38] U. Orozco-Rosas, O. Montiel, and R. Sepulveda, "Mobile robot path planning using membrane evolutionary artificial potential field," *Applied Soft Computing Journal*, vol. 77, pp. 236-251, 2019.
[39] A. Azzabi, and K. Nouri, "An advanced potential field method proposed for mobile robot path planning," *Trans. of the Institute of Measurement and Control*, vol. 41, no. 11, pp. 3132-3144, 2019.
[40] F. Bounini, D. Gingras, H. Pollart, and D. Gruer, "Modified artificial potential field method for online path planning applications," in 2017 IEEE Intelligent Vehicles Symposium (IV), Redondo Beach, CA, USA, June 2017, pp. 180-185.
[41] Q. Yao, Z. Zheng, L. Qi, H. Yuan, X. Guo, M. Zhao, Z. Liu, and T. Yang, "Path planning method with improved artificial field – a reinforcement learning perspective," *IEEE Access*, pp. 135513-135523, 2020.
[42] X. Fan, Y. Guo, H. Liu, B. Wei, and W. Lyu, "Improved artificial potential method applied for AUV path planning," *Mathematical Problems in Engineering*, 2020.
[43] R. Szczepanski, T. Tarczewski, and K. Erwinski, "Energy efficient local path planning algorithm based on predictive artificial potential field," *IEEE Access*, pp. 39729-39742, 2020.
[44] A. Mohammed, B. Scmidt, L. Wang, and L. Gao, "Minimizing energy consumption for robot arm movement," *Procedia CIRP*, vol. 25, pp. 400-405, 2014.
[45] M. Mahdavian, A. Yousefi-Komi, M. Shariat-Panahi, and A. Ghasemi-Toudeshki, "Optimal trajectory generation for energy consumption minimization and moving obstacle avoidance of a 4DOF robot arm," in 3rd RSI Int. Conf. on Robotics and Mechatronics, Tehran, Iran, Oct. 7-9, 2015, pp. 353-358.
[46] M. Siar, and A. Fakharian, "Energy efficiency in the robot arm using genetic algorithm," in 2018 Artificial Intelligence and Robotics, IRANOPEN 2018 and the 8th Conference on Artificial Intelligence and Robotics, 2018, pp. 14–20.
[47] K. Belda, and O. Rovny, "Predictive control of 5 DOF robot arm of autonomous mobile robotic system," in 21st Int. Conf. on Process Control, Strbske Pleso, Slovakia, June 6-9, 2017, pp. 339-344.
[48] W. Widihiada, I. G. N. Santhiarsa, and C. G. I. Partha, "Design of motion control of mobile robot manipulator," *Int. Journal of Mech. Eng. And Robotics Research*, vol. 9, no. 11, pp. 1509-1514, 2020.
[49] A. Seddaoui, C. M. Saaj, "Collision-free optimal trajectory generation for a space robot using genetic algorithm," *Acta Astronautica*, vol. 179, pp. 311-321, 2021.

[50] R. Fotouhi, H. Salmasi, I. Waheed, and M. Vakil, “Trajectory planning for a wheeled mobile robot and its robotic arm,” in ASME 2010 Int. Mech. Eng. Congress & Exposition, Bancouver, BC, Canada, Nov. 12-18, 2010.
[51] F. Ramos, M. Gajamohan, N. Huebel, R. D’Andrea, “Time-optimal trajectory generator for robotic manipulators,” 2013.
[52] R. Katzschmann, T. Kroger, T. Asfour, O. Khatib, “Towards online trajectory generation considering robot dynamics and torque limits,” in 2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Tokyo, Japan, Nov. 3-7, 2013, pp. 5644-5651.
[53] R. Zhao, D. Sidobre, and W. He, “Online via-points trajectory generation for reactive manipulations,” in 2014 IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics, Besancon, France, July 8-11, 2014, pp. 1243-1248.
[54] T. Kroger, and J. Padial, “Simple and robust visual servo control of robot arms using an on-line trajectory generator,” in 2012 IEEE Int. Conf. on Robotics and Automation, Saint Paul, MN, USA, May 14-18, 2012, pp. 4862-4869.
[55] Y. Quinoniz, O. Zatarain, C. Lizarraga, and J. Mejia, “Proposal for a new method to improve the trajectory generation of a robotic arm using a distribution function,” in New Perspectives in Software Engineering: Proceedings of the 9th International Conference on Software Process Improvement (CIMPS 2020), pp. 213-231.
[56] M. Wang, J. Xiao, F. Zeng, and G. Wang, “Research on optimized time-synchronous online trajectory generation method for a robot arm,” *Robotics and Autonomous Systems*, vol. 126, 2020.
[57] D. G. Wall, J. Economou, and K. Knowles, “Quasi-real-time confined environment path generation for mobile robotic manipulator arms,” *Proceedings of the Institution of Mech. Engineers Part I: Journal of Systems and Control Engineering*, vol. 232, no. 3, pp. 270-284, 2018.
[58] O. Koc, G. Maeda, J. Peters, “Online optimal trajectory generation for robot table tennis,” *Robotics and Autonomous Systems*, vol. 305, pp. 121-137, 2018.
[59] J. Colan, J. Nakanishi, T. Aoyama, and Y. Hasegawa, “Optimization-based constrained trajectory generation for robot-assisted stitching in endonasal surgery,” *Robotics*, vol.10, no. 27, 2021.
[60] C. E. Cortes, D. Saez, F. Milla, A. Nunez, and M. Riquelme, “Hybrid predictive control for real-time optimization of public transport systems’ operations based on evolutionary multi-objective optimization,” *Transportation Research Part C*, vol. 18, pp. 757-769, 2010.
[61] X. Gong, T. De Pessemier, L. Martens, and W. Joseph, “Energy- and labor-aware flexible job scheduling under dynamic electricity pricing: a many-objective optimization investigation,” *Journal of Cleaner Production*, vol. 209, pp. 1078-1094, 2019.
[62] A. Malik, T. Henderson, R. Prazencia, “Multi-objective swarm intelligence trajectory generation for a 7 degree of freedom robotic manipulator,” *Robotics*, vol. 10, no. 127, 2021.
[63] K. Li, R. Wang, T. Zhang, and H. Ishibuchi, “Evolutionary many-objective optimization: a comparative study of the state-of-the-art,” *IEEE Access*, vol. 6, pp. 26194-26214, 2018.
[64] J. L. Ringuest. "Compromise programming." Multi-objective optimization: behavioral and computational considerations (1992): 51-59.
[65] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multi-objective genetic algorithm: NSGA-II,” *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, 2002.

[66] J. Blank, and K. Deb, "Pymoo: multi-objective optimization in Python," *IEEE Access*, vol. 8, pp. 89497-89509, 2020.
[67] X. Zhang, H. Dou, Y. Zhang, K. Jan, and B. Ivanovic, "Trajectory planning and control of handling robot manipulator," *Journal of Physics: Conference Series*, vol. 1802, 2021.
[68] A. A. Raji, O. S. Asaolu, and T. T. Akano, "Joint space robot arm trajectory planning using septic function," *ABUAD Journal of Engineering Research and Development*, vol. 5, no. 1, pp. 110-123, June 2022.
[69] M.A.N. Huda, S. H. Susilo, and P. M. Adhi, "Implementation of inverse kinematic and trajectory planning on 6-DOF robotic arm for straight-flat welding movement," *Journal of Engineering Design and Technology*, vol. 22, no. 1, pp. 51-61, March 2022.
[70] D. Zhao, Z. Ding, W. Li, S. Zhao, and Y. Du, "Robotic arm trajectory planning method using deep deterministic policy gradient with hierarchical memory structure," *IEEE Access*, vol. 11, pp. 140801-140814, 2023.
[71] X. Miao, H. Fu, and X. Song, "Research on motion trajectory planning of the robotic arm of a robot," *Artificial Life and Robotics*, vol. 27, pp. 561-567, 2022.
[72] S. D. Das, V. Bain, and P. Rakshit, "Energy optimized robot arm path planning using differential evolution in dynamic environment,," in Proceeding of the Second Int. Conf. on Intelligent Computing and Control Systems, Madurai, India, June 14-15, 2018, pp. 1267-1272.
[73] A. Sengupta, T. Chakroborti, A. Konar, and A. Nagar, "Energy efficient trajectory planning by a robot arm using invasive weed optimization technique" in Proceedings of the 2011 3rd World Congress on Nature and Biologically Inspired Computing, Salamanca, Spain, Oct. 19-21, 2011, pp. 311-316.
[74] S. Zhang, Z. Xia, M. Chen, and S. Cheng, "Multi-objective optimal trajectory planning for robotic arms using deep reinforcement learning," *Sensors*, vol. 23, 2023.
[75] H. Belaidi and H. Bentarzi, "NURBs trajectory generation and following by an autonomous mobile robot navigating in 3D environment," in the 4th Annual IEEE Int. Conf. on Cyber Technology in Automation, Control and Intelligent Systems, Hong Kong, China, June 4-7, 2014, pp. 168-173.
[76] M. Elbanhawi, M. Simic, and R. N. Jazar, "Continuous path smoothing for car-like robots using B-spline curves," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 80, pp. 23-56, 2015.
[77] K. R. Simba, N. Uchiyama, S. Sano, "Real-time smooth trajectory generation for nonholonomic mobile robots using Bezier curves," *Robotics and Computer-Integrated Manufacturing*, vol. 41, pp. 31-42, 2016.
[78] D. K. Tiep, K. Lee, D, Im, B. Kwak, and Y. Ryoo, "Design of fuzzy-PID controller for path tracking of mobile robot with differential drive," *Int. Jornal of Fuzzy Logic and Intelligent Systems*, vol. 18, no. 3, pp. 220-228, Sep. 2018.
[79] N. H. Thai, T. T. K. Ly, H. Thien, and L. Q. Dzung, "Trajectory tracking control for differential-drive mobile robot by a variable parameter PID controller," *Int. Journal of Mech. Eng. And Robotics Research*, vol. 11, no. 8, pp. 614-621, Aug. 2022.
[80] U. Zangina, S. Buyamin, M. S. Z. Abidin, M. S. A. Mahmud, and J. S. Hasan, "Non-linear PID controller for trajectory tracking of a differential drive mobile robot," *Journal of Mech. Eng. Research and Developments*, vol. 43, no. 7, pp. 255-270, 2020.

[81] R. L. S. Sousa, M. Forte, F. G. Nogueira, and B. C. Torrico, “Trajectory tracking control of a nonholonomic mobile robot with differential drive,” in 2016 IEEE Biennial Congress of Argentina, Buenos Aires, Argentina, June 15-17, 2016, pp. 1-6.
[82] M. J. Mohamed, and M. D. Hamza, “Design PID neural network controller for trajectory tracking of differential drive mobile robot based on PSO,” *Engineering and Technology Journal*, vol. 27, part A, no. 12, pp. 574-583, 2019.
[83] A. M. et al, “Trajectory tracking of differential drive mobile robots using fractional-order proportional-integral-derivative controller design tuned by an enhanced fruit fly optimization,” *Measurement and Control*, vol. 55, no. 3-4, pp. 209-226, 2022.
[84] R. H. Mohammed, M. E. Dessouki, and B. E. Elnaghi, “Path tracking control of differential drive mobile robot based on chaotic-billiards optimization algorithm,” *Int. Journal of Electrical and Computer Engineering*, vol. 13, no. 2, pp. 1449-1462, April 2023.
[85] T. Q. Khai and Y Ryoo, “Design of adaptive kinematic controller using radial basis function neural network for trajectory tracking control of differential-drive mobile robot,” *Int. Journal of Fuzzy Logic and Intelligent Systems*, vol. 19, no. 4, pp. 349-359, Dec. 2019.
[86] C. Shih, and L. Lin, “Trajectory planning and tracking control of a differential-drive mobile robot in a picture drawing application,” *Robotics*, vol. 6, no. 17, 2017.
[87] D. Xie, S. Wang, and Y. Wang, “Trajectory tracking control of differential drive mobile robot based on improved kinematics controller algorithm,” in 2018 Chinese Automation Congress, Xi’an, China, Nov. 30 – Dec. 2, 2018.
[88] H. Wu, and M. Q. Zaman, “LiDAR based trajectory-tracking of an autonomous differential drive mobile robot using fuzzy sliding mode controller,” *IEEE Access*, vol. 10, pp. 33713-33722, 2022.
[89] K. Shojaei, A. M. Shahri, A. Tarakameh, and B. Tabibian, “Adaptive trajectory tracking control of a differential drive wheeled mobile robot,” *Robotica*, vol. 29, pp. 391-402, 2011.
[90] D. Chwa, “Tracking control of differential-drive wheeled mobile robots using a backstepping-like feedback linearization,” *IEEE Trans. on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 40, no. 6, pp. 1285-1295, Nov. 2010.
[91] C. Tiriolo, G. Franzè, and W. Lucia, “A receding horizon trajectory tracking strategy for input-constrained differential-drive robots via feedback linearization,” *IEEE Trans. on Control Systems Technology*, vol. 31, no. 3, pp. 1460-1467, May 2023.
[92] H. B. Kazemian, “The SOF-PID controller for the control of a MIMO robot arm,” *IEEE Trans. on Fuzzy Systems*, vol. 10, no. 4, pp. 523-532, Aug. 2002.
[93] E. M. Jafarov, M. N. A. Parlakçi, and Y. Istefanopulos, “A new variable structure PID-controller design for robot manipulators,” *IEEE Trans. on Control Systems Technology*, vol. 13, no. 1, pp. 122-130, Jan. 2005.
[94] P. Rocco, “Stability of PID control for industrial robot arms,” *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 606-614, Aug. 1996.
[95] D. P. Kwok, and F. Sheng, “Genetic algorithm and simulated annealing for optimal robot arm PID control,” in Proceedings of the First IEEE Conf. on Evolutionary Computation, Orlando, FL, USA, June 27-29, 1994, pp. 707-713.
[96] M. Plooij, W. Wolfslag, and M. Wisse, “Robust feedforward control of robotic arms with friction model uncertainty,” *Robotics and Autonomous Systems,* vol. 70, pp. 83-91, 2015.
[97] R. L. Wells, J. K. Schueller, and J. Tlusty, “Feedforward and feedback control of a flexible robotic arm,” *IEEE Control Systems Magazine*, vol. 10, no. 1, pp. 9-15, Jan. 1990.

[98] S. Tolu, M. Vanegas, J. A. Garrido, N. R. Luque, and E. Ros, "Adaptive and predictive control of a simulated robot arm," *Int. Journal of Neural Systems*, vol. 23, no. 3, 2013.

[99] Y. H. Kim and F. L. Lewis, "Neural network output feedback control of robot manipulators," *IEEE Trans. on Robotics and Automation,* vol. 15, no. 2, pp. 301-309, April 1999.

[100] K. Xu and Z. Wang, "The design of a neural network-based adaptive control method for robotic arm trajectory tracking," *Neural Computing and Applications*, vol. 35, pp. 8785-8795, 2023.

[101] M. W. Apong, S. Hutchinson, and M. Vidyasagar, "Robot Modeling and Control," John Wiley & Sons, Inc., 2006, Chapter 8.

[102] H. Sadeghian, L. Villani, M. Keshmiri, and Bruno Siciliano, "Task-space control of robot manipulators with null-space compliance," *IEEE Trans. on Robotics*, vol. 30, no. 2, pp. 493-506, April 2014.

[103] H. Yazarel, and C. C. Cheah, "Task-space adaptive control of robotic manipulators with uncertainties in gravity regressor matrix and kinematics," *IEEE Trans. On Automatic Control*, vol. 47, no. 9, pp. 1580-1585, Sep. 2002.

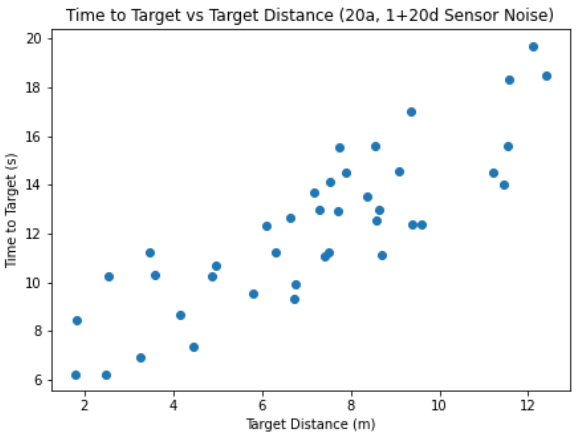
[104] H. Su, W. Qi, J. Chen, and D. Zhang, "Fuzzy approximation-based task-space control of robot manipulators with remote center of motion constraint," *IEEE Trans. on Fuzzy Systems*, vol. 30, no. 6, pp. 1564-1573, June 2022.

R. Dhaouadi and A. A. Hatab, "Dynamic modelling of differential-drive mobile robots using Lagrange and Newton-Euler methodologies: a unified framework," *Advanced in Robotics & Automation*, vol. 2, no. 2, 2013.

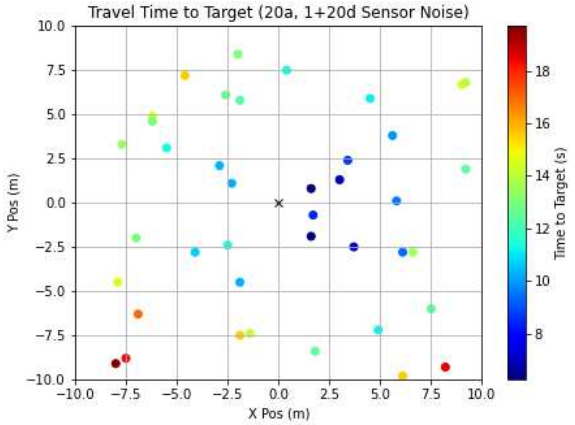
Appendix A - Additional AoA Homing Controller Results

This appendix presents additional results from the AoA homing controller testing.

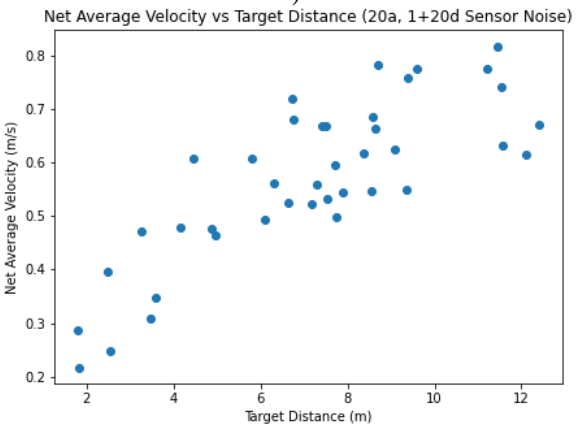
A.1 - Obstacle-free Simulation Results



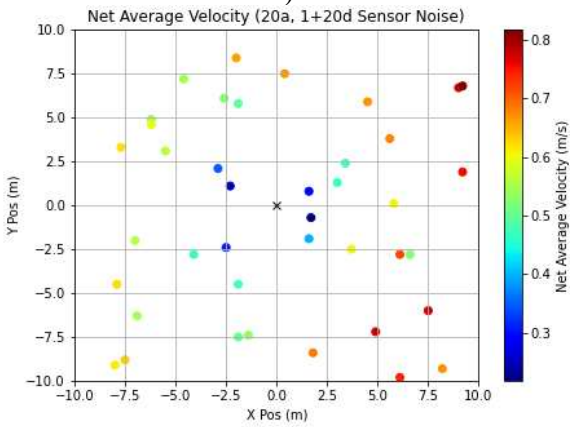
a)



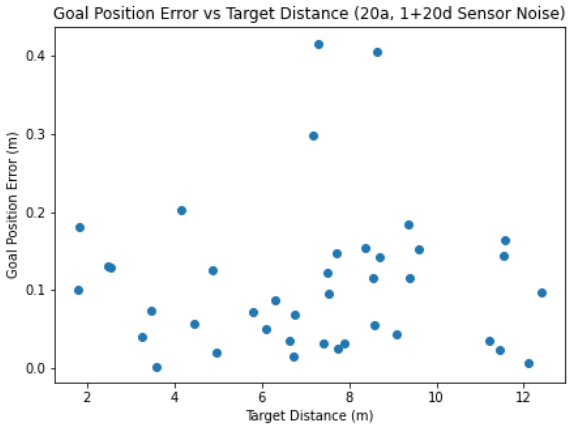
b)



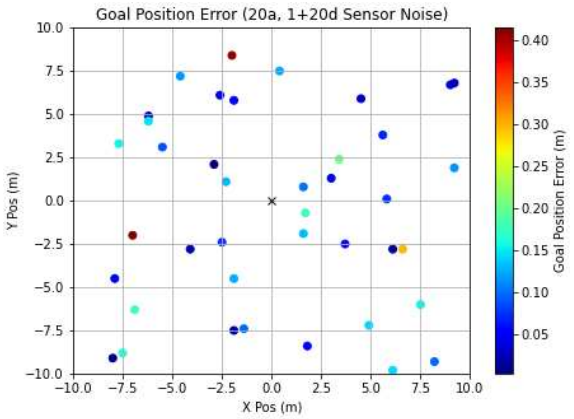
c)



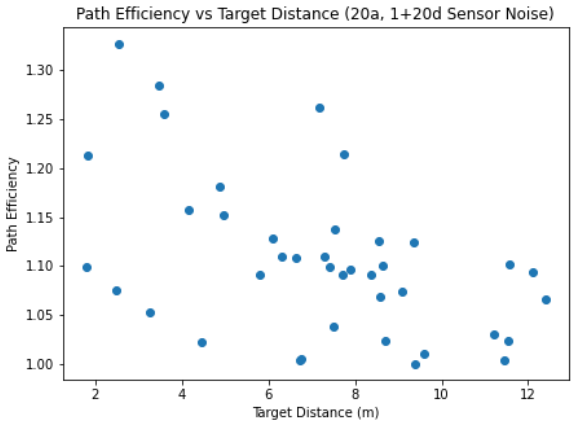
d)



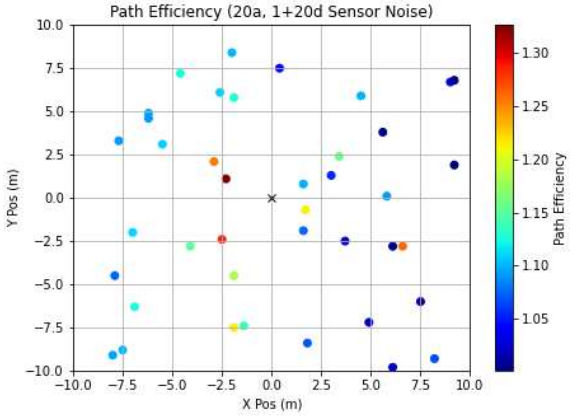
e)



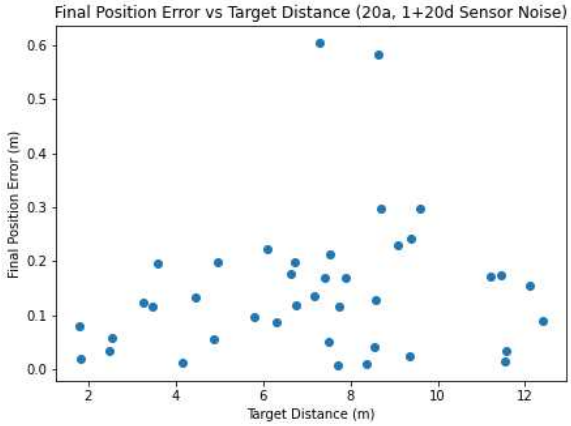
f)



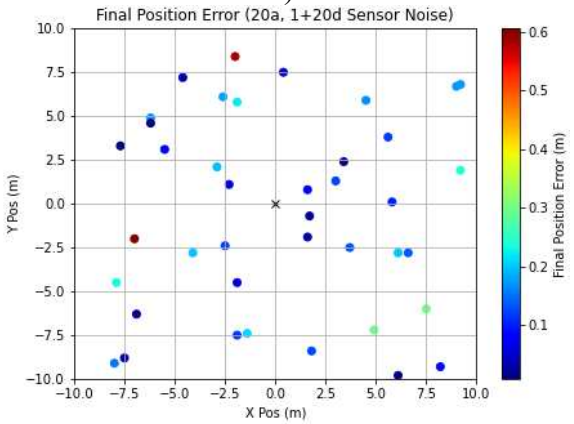
g)



h)



i)



j)

Figure A.1. Performance metrics for obstacle-free homing controller testing by target location.

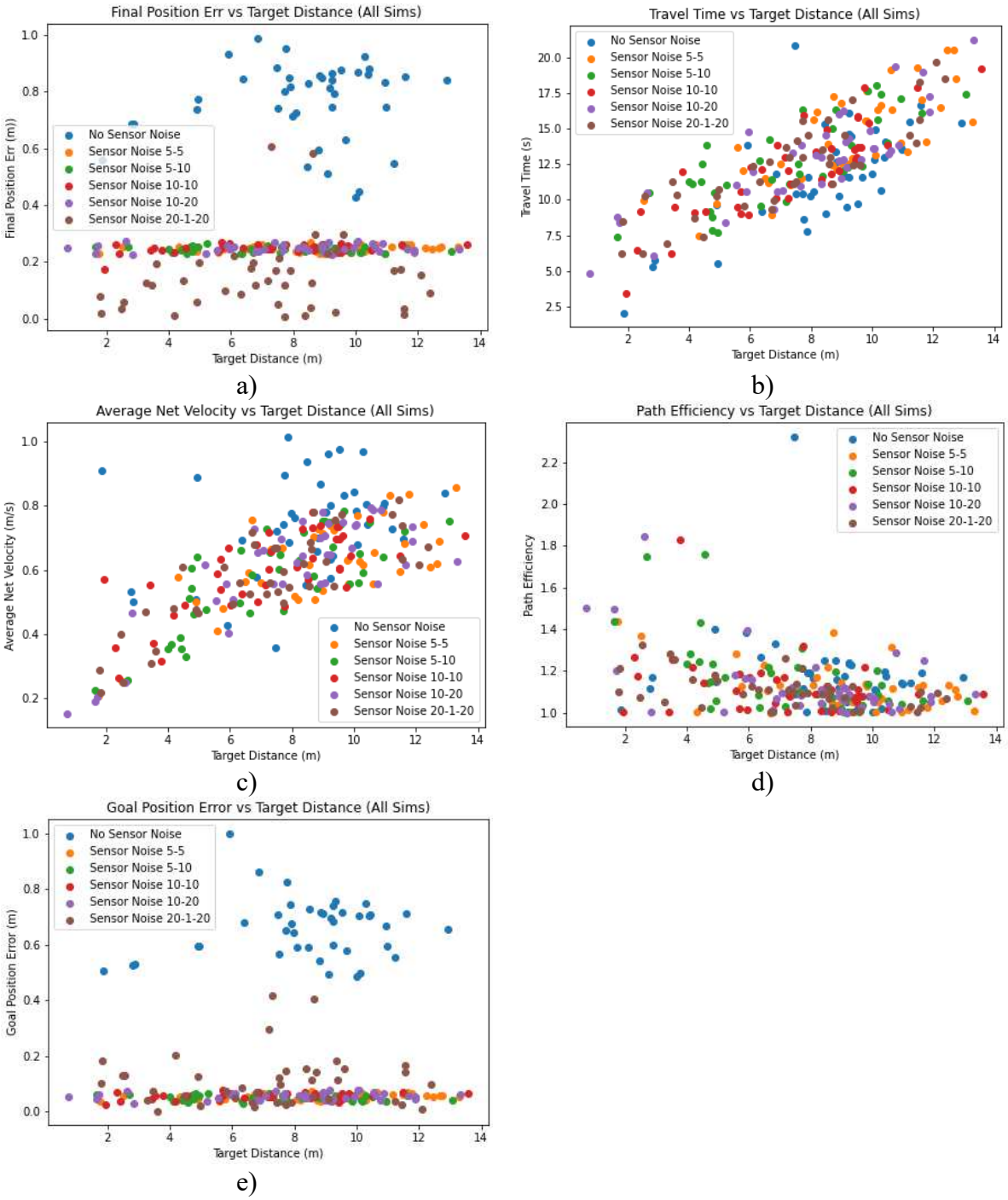


Figure A.2. Distribution of performance metrics versus distance to target for obstacle-free homing controller simulations for all sensor noise levels.

A.2 - Obstacle Avoidance Simulation Results

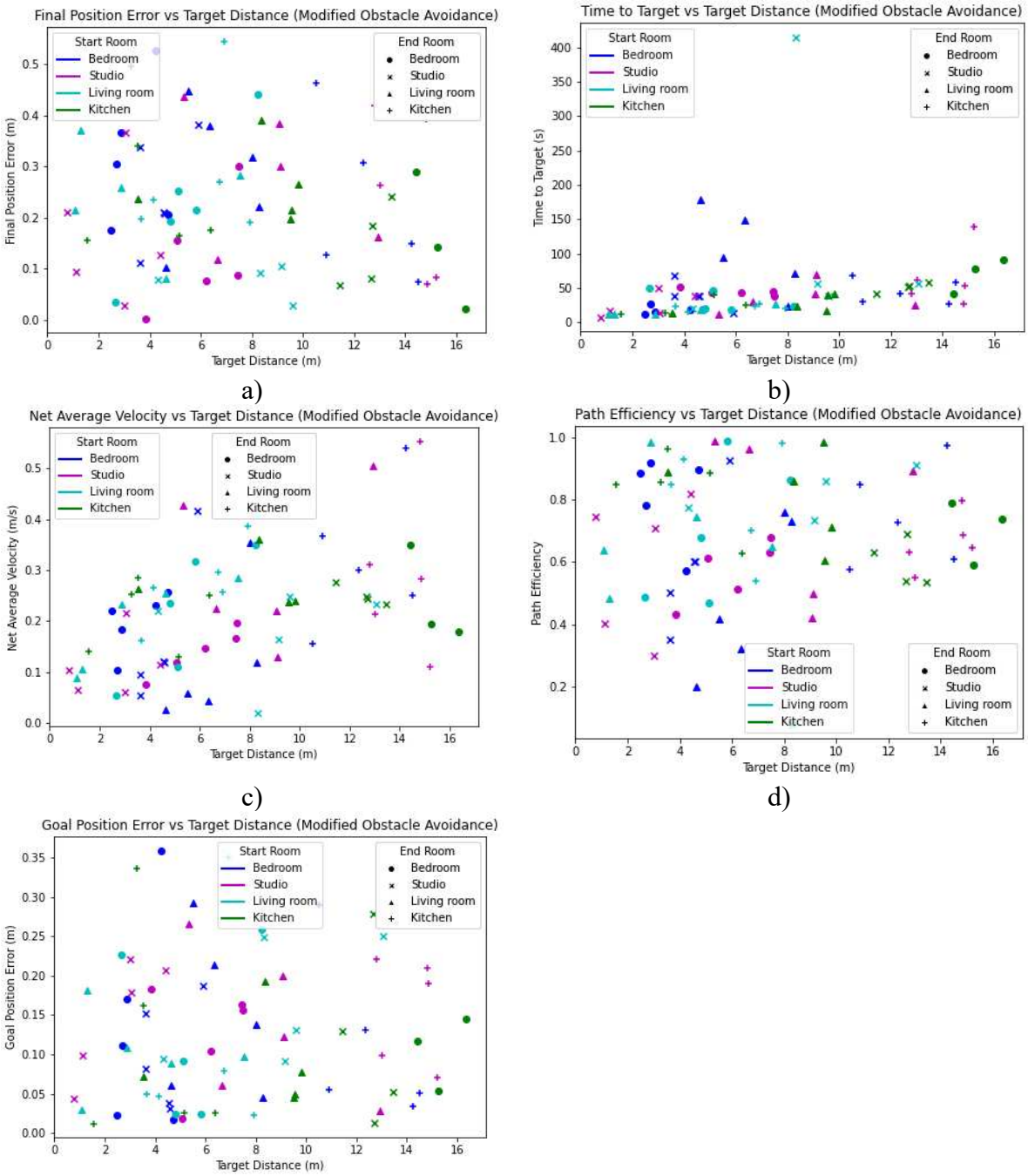


Figure A.3. Full results for obstacle avoidance homing controller simulations.

Appendix B - Additional APF-HW Results

This appendix presents additional results from the testing comparing the APF-HW method and other APF-based navigation methods.

B.1 - Small House World Results

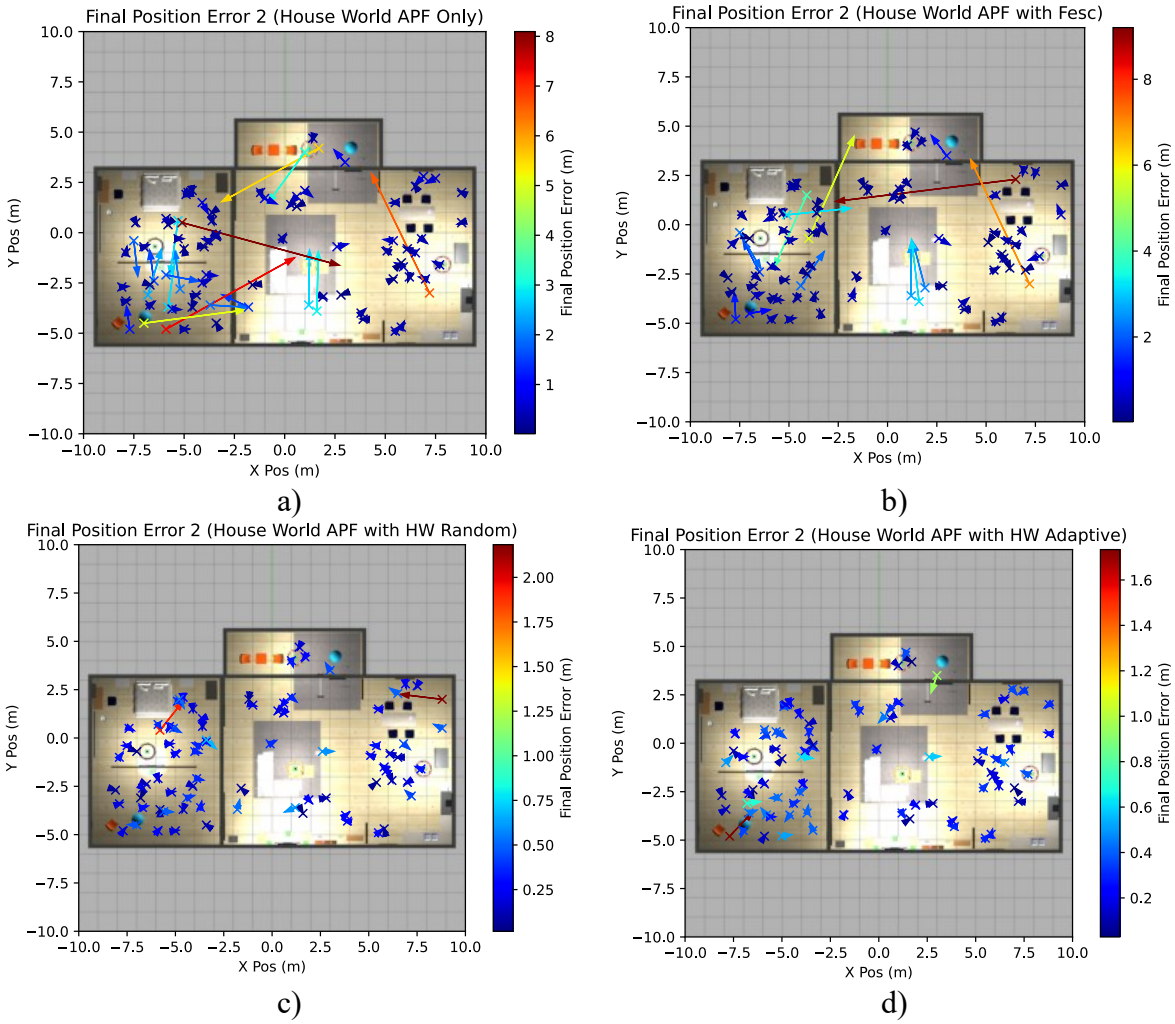


Figure B.1. Final position error results for AWS Small House World APF simulations.

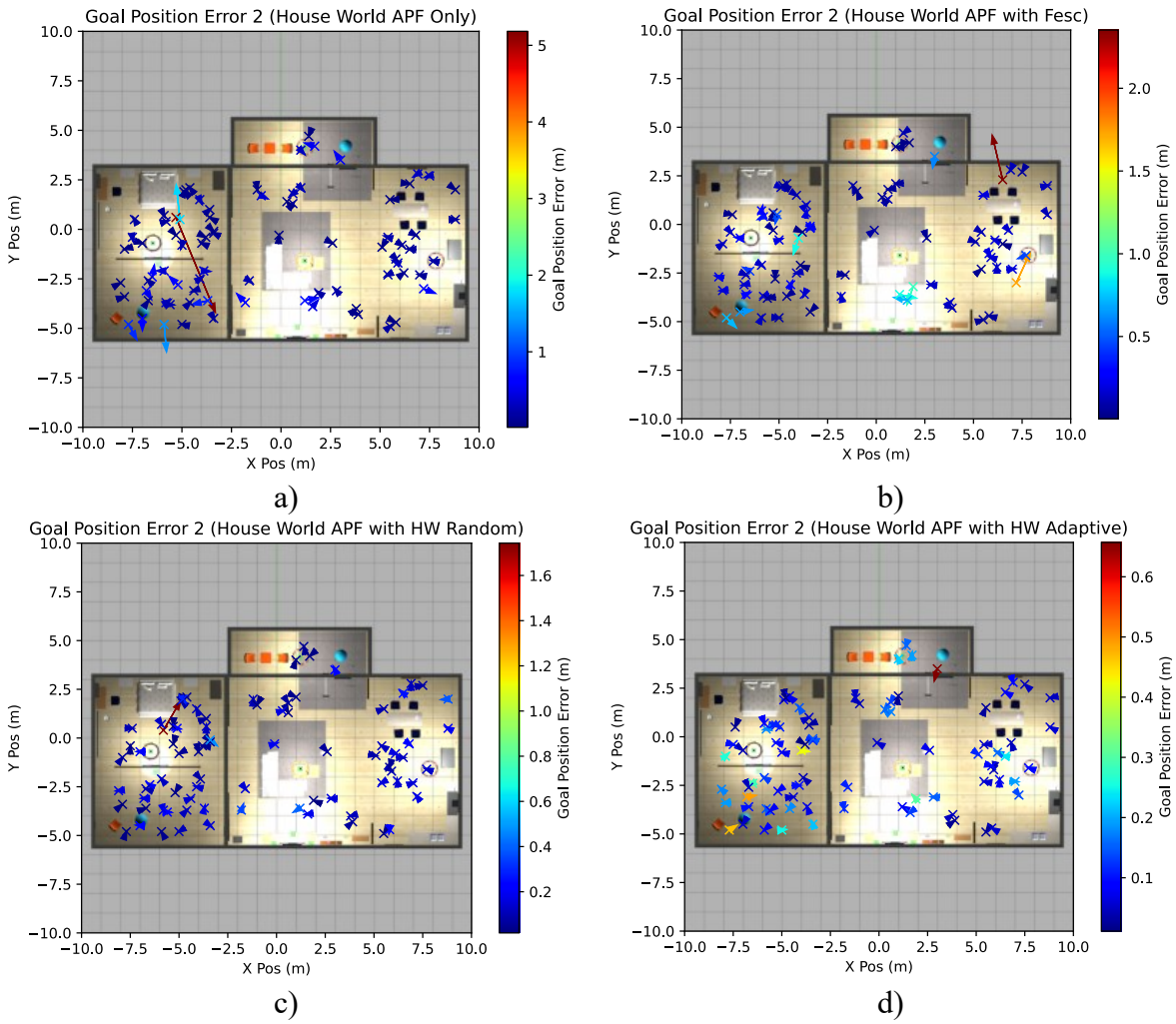


Figure B.2. Final goal position error for AWS Small House World APF simulations.

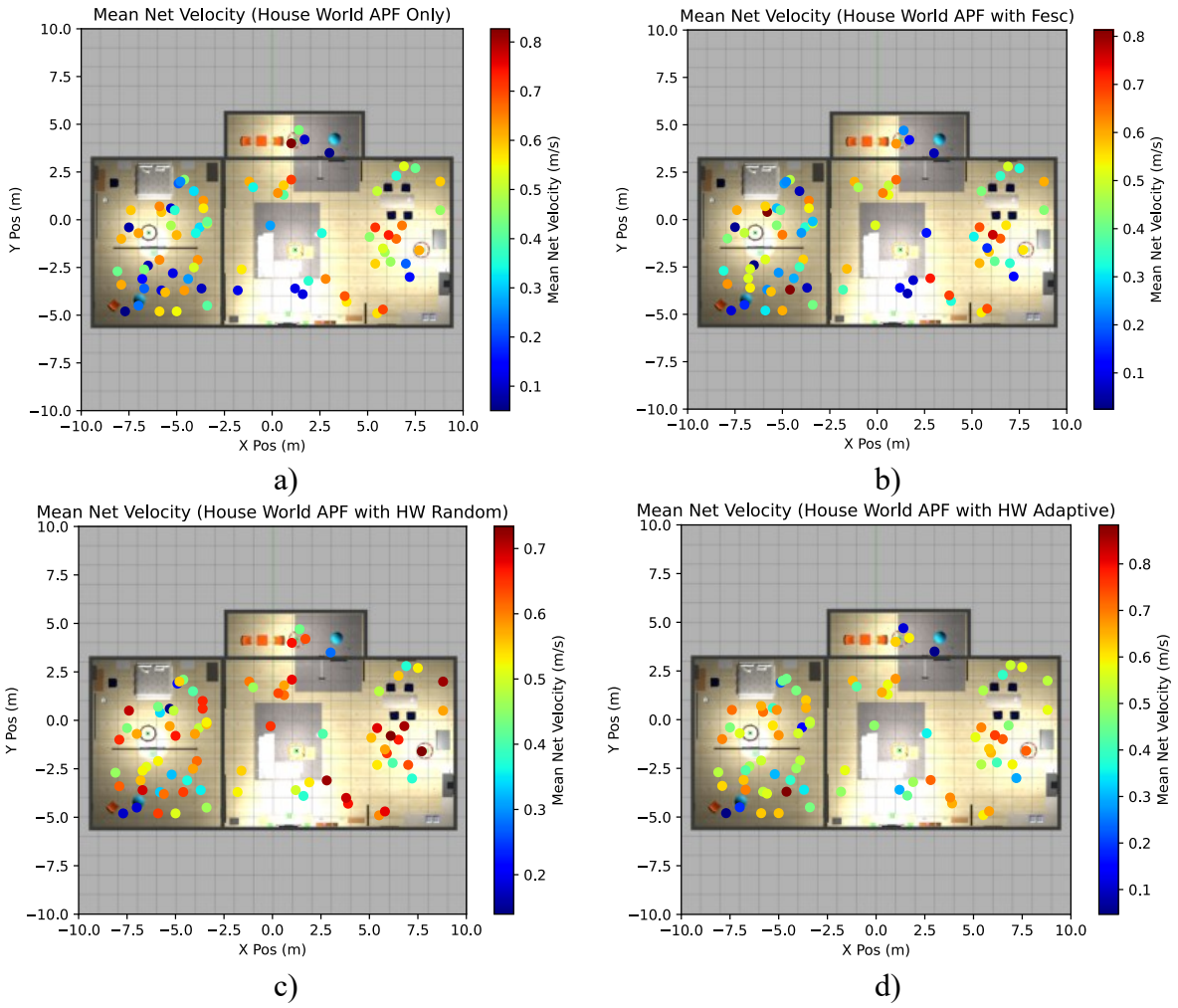


Figure B.3. Net velocity results for AWS Small House World APF simulations.

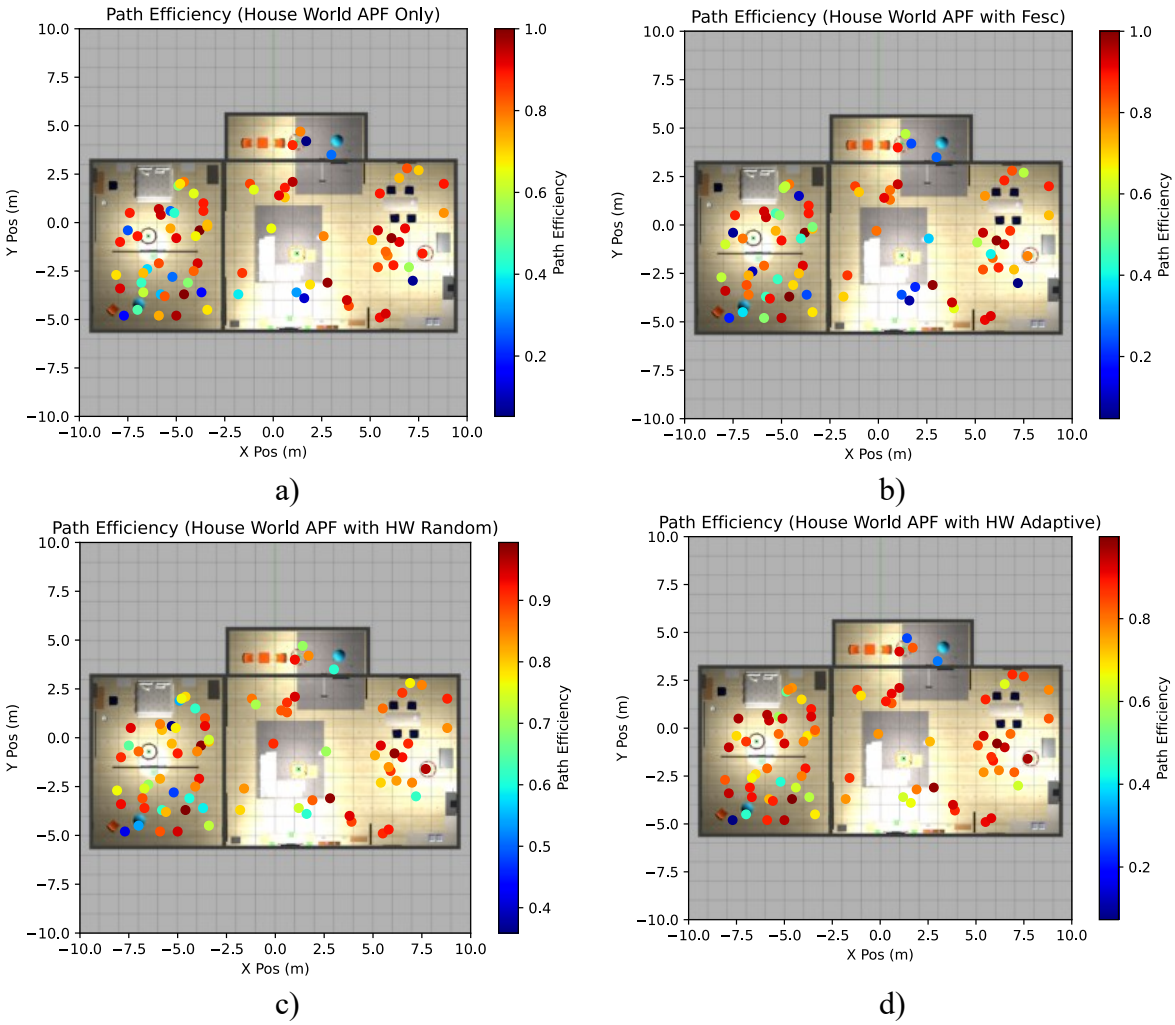


Figure B.4. Path efficiency results for AWS Small House World APF simulations.

B.2 - Bookstore World Results

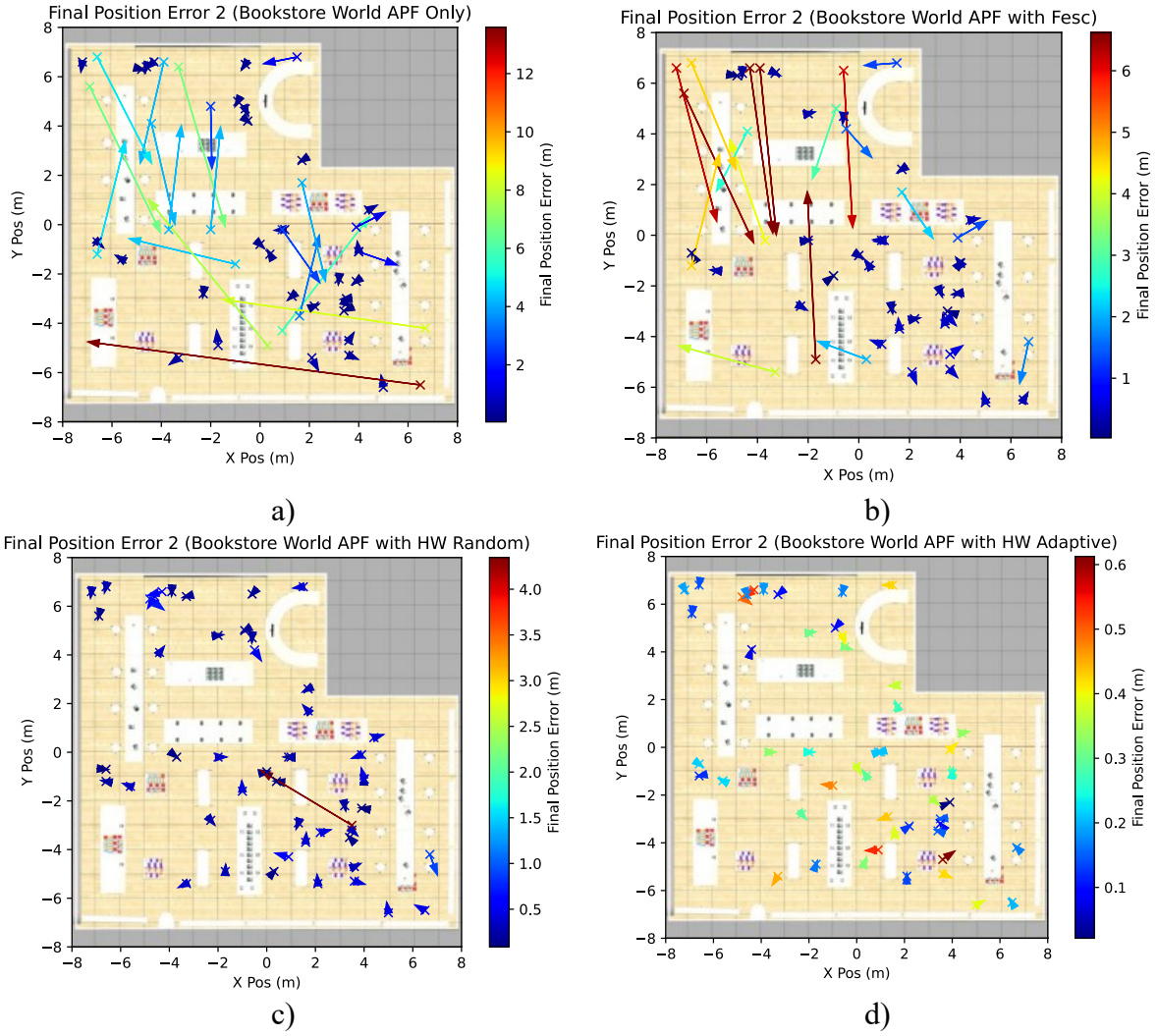


Figure B.5. Final position error results for AWS Bookstore World APF simulations.

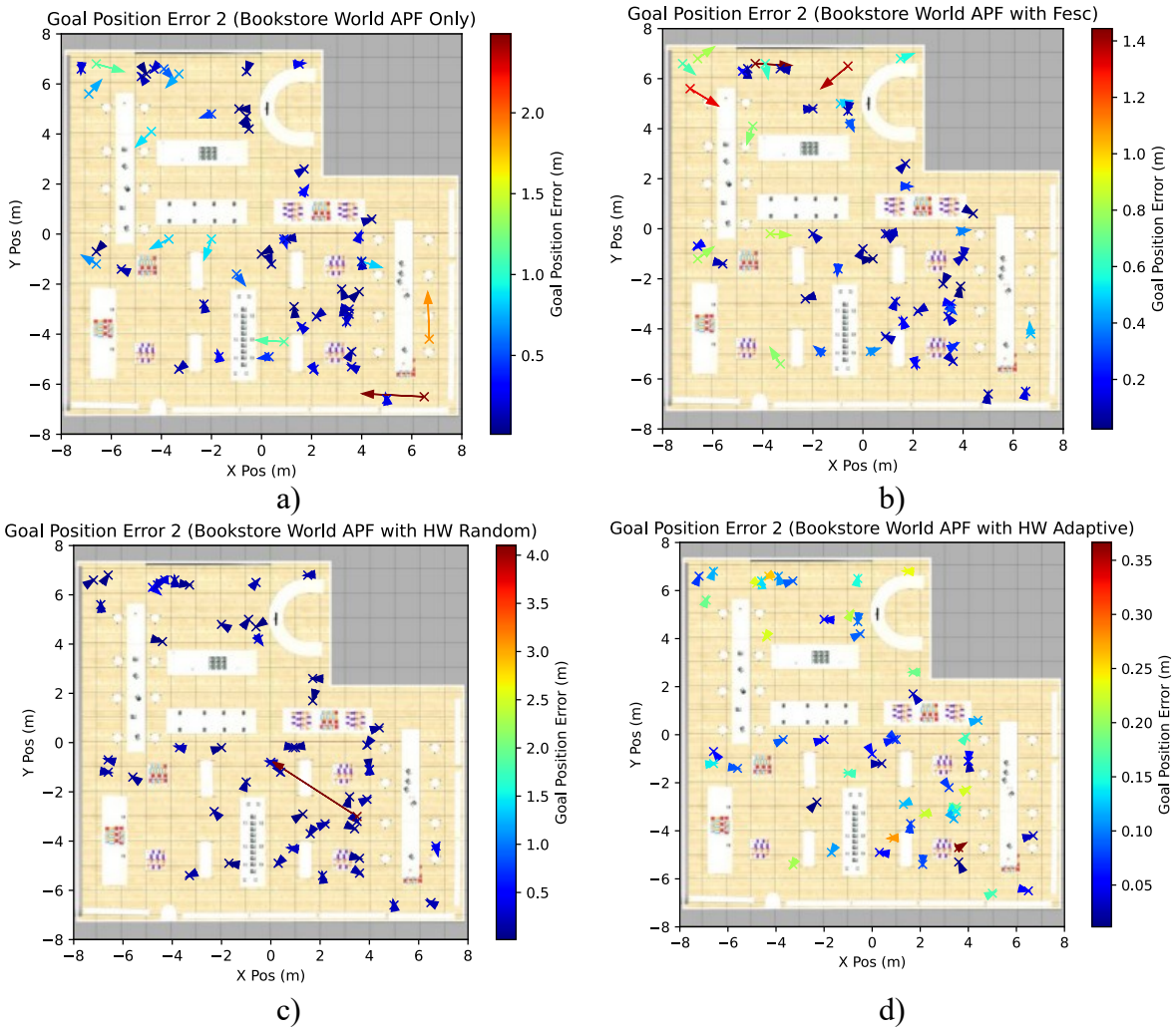


Figure B.6. Final goal position error results for AWS Bookstore World APF simulations.

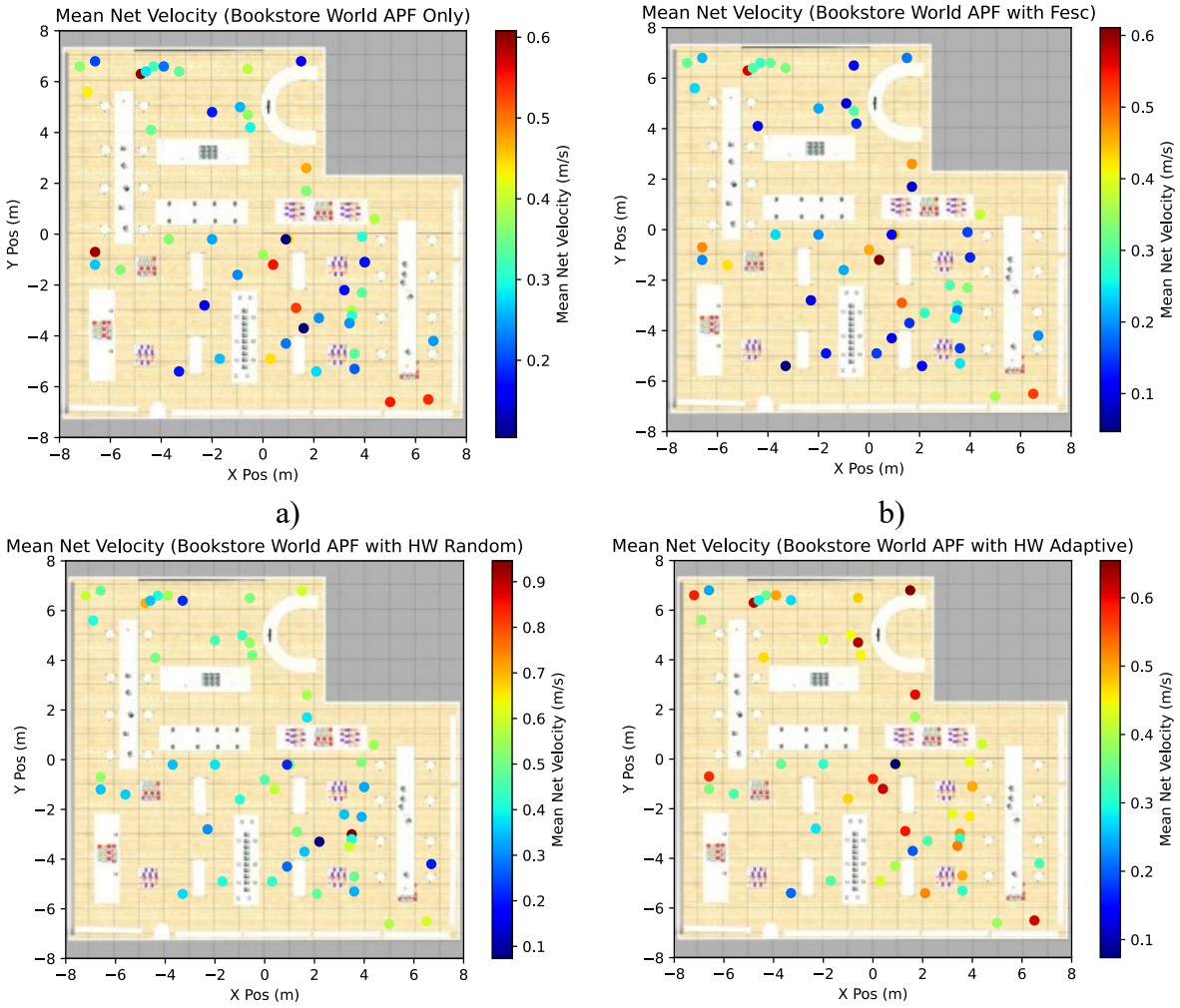


Figure B.7. Net velocity results for AWS Bookstore World APF simulations.

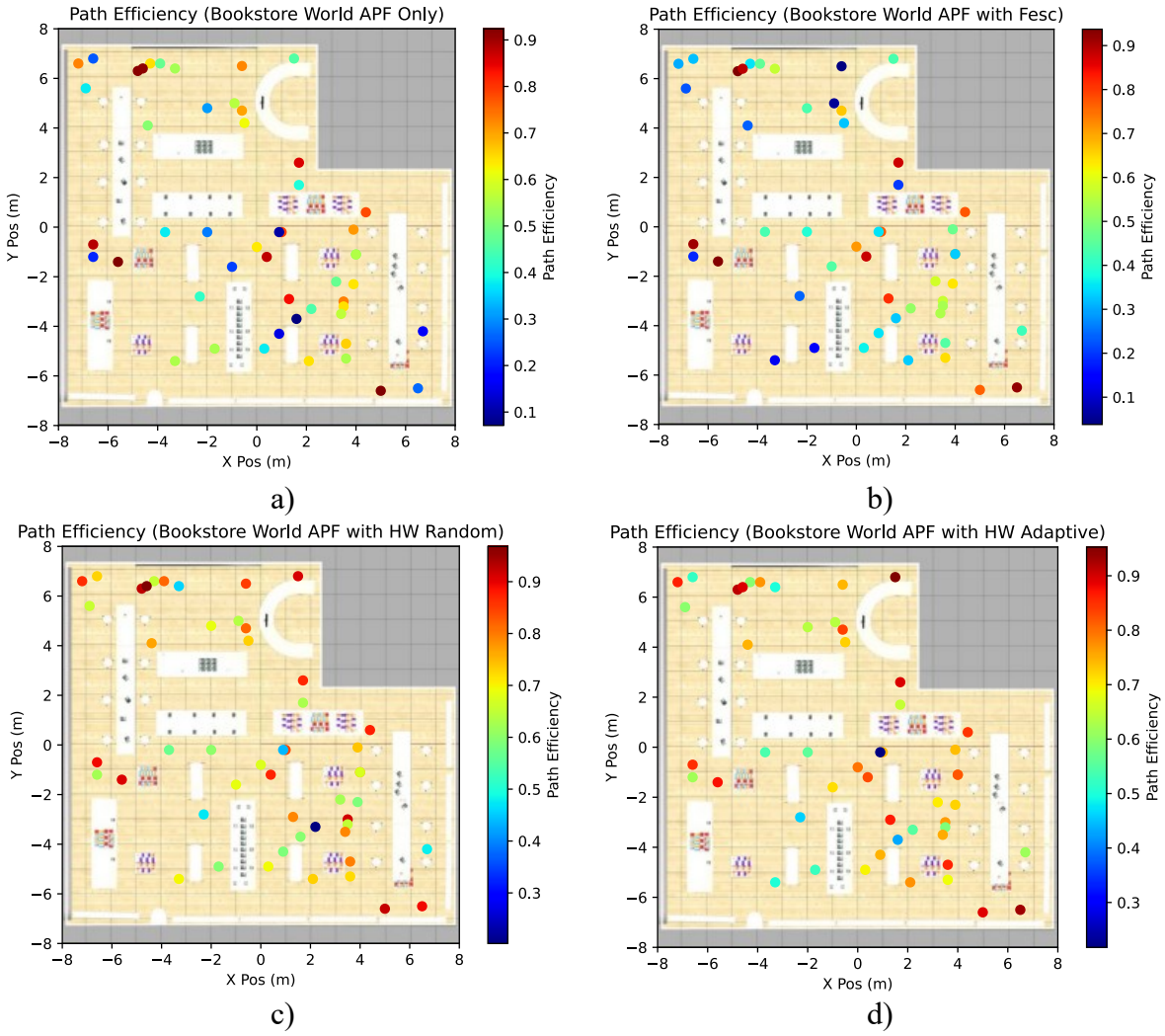


Figure B.8. Path efficiency results for AWS Bookstore World APF simulations.

B.3 - Café World Results

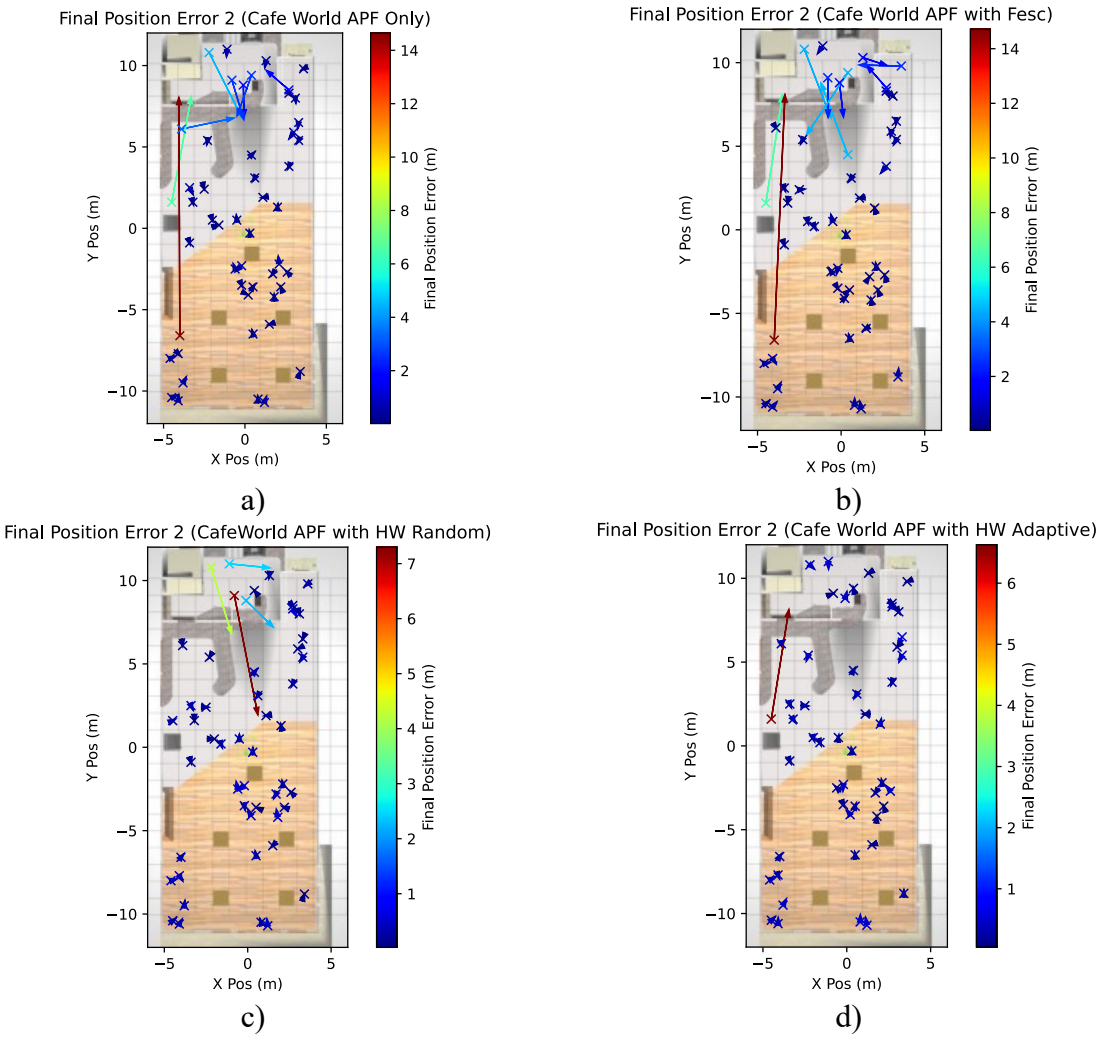


Figure B.9. Final position error results for Café World APF simulations.

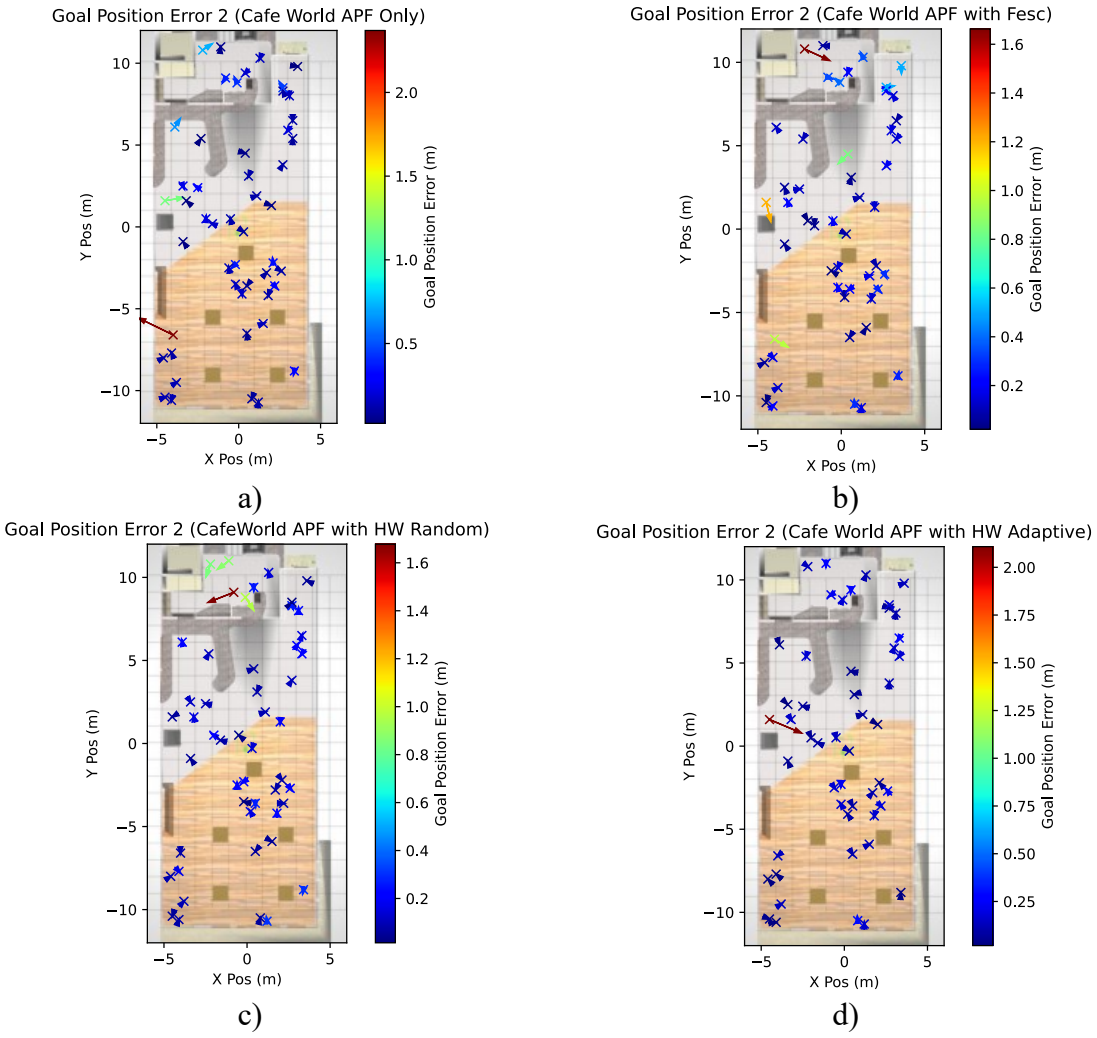


Figure B.10. Final goal position error for Café World APF simulations.

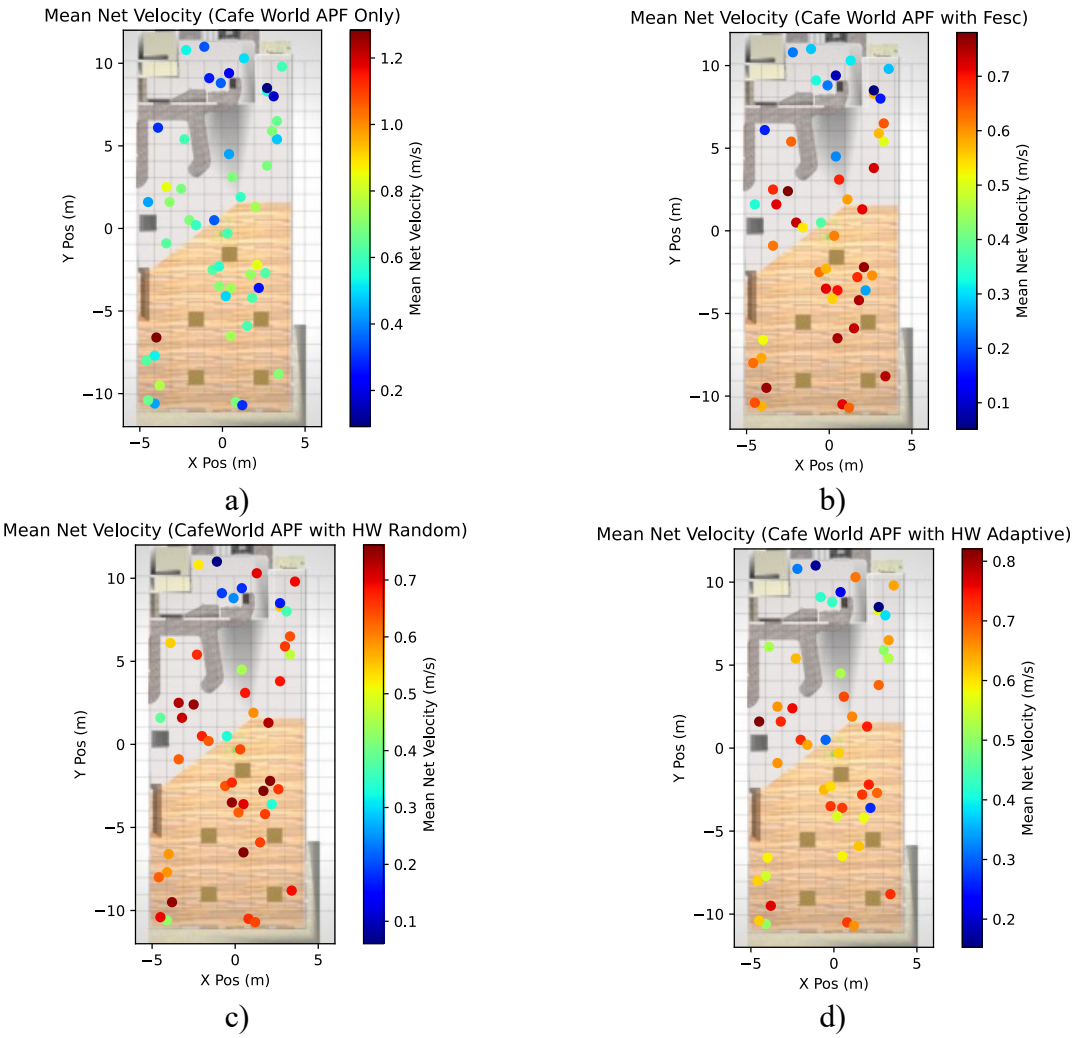


Figure B.11. Net velocity results for Café World APF simulations.

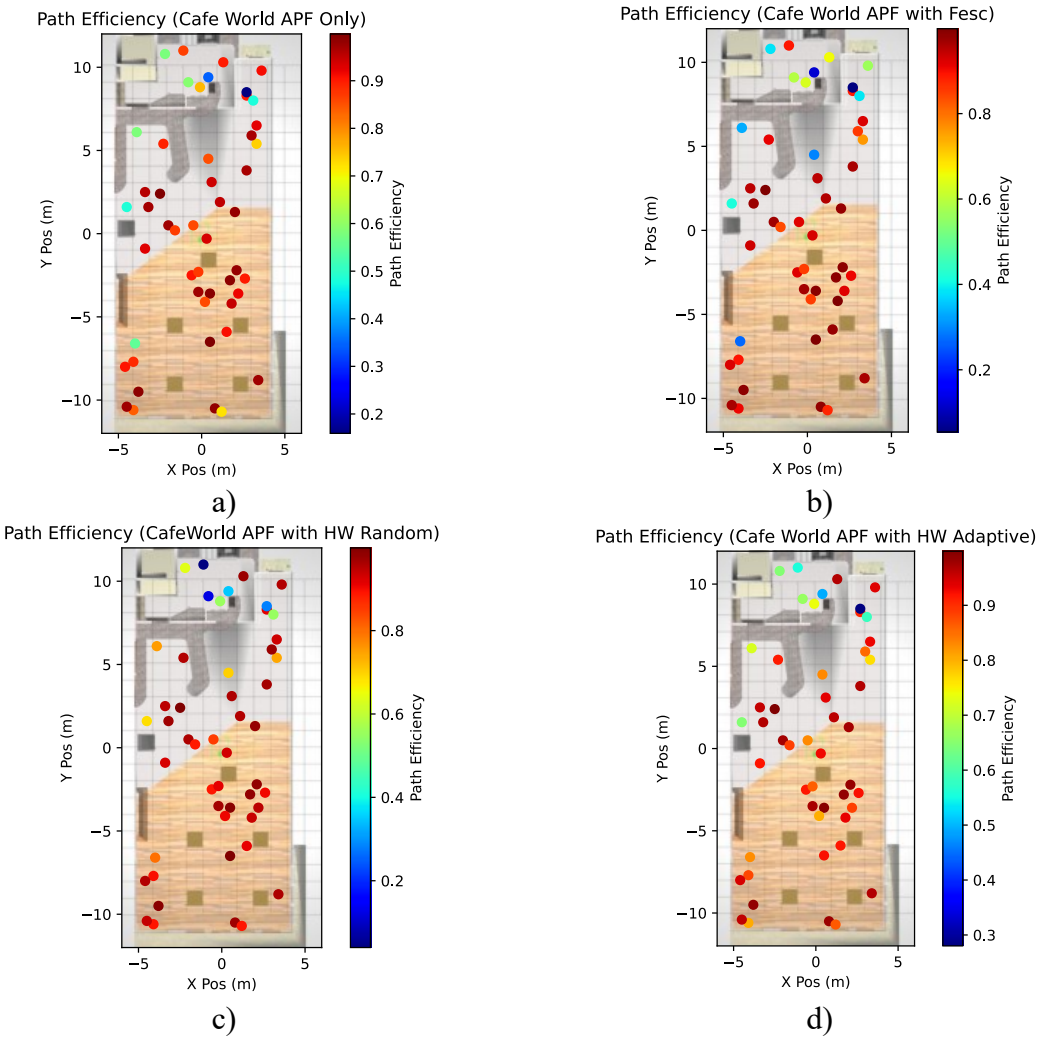


Figure B.12. Path efficiency results for Café World APF simulations.

Appendix C - 6DOF Robotic Arm Inverse Kinematics Solution

This appendix presents the inverse kinematics solution for a 6DOF robotic arm with a decoupled wrist and arm configuration. The robot arm is parameterized with Denavit-Hartenberg parameters. For a desired end-effector configuration given by the position vector x_{tt} and rotation matrix, R_{tt}, the inverse kinematics solution can be calculated as follows.

$$\vec{x}_{tt} = \begin{bmatrix} x_{tt} \\ y_{tt} \\ z_{tt} \end{bmatrix}$$

$$R_{tt} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Start by finding the position of the wrist center, $\vec{x}_w$.

$$\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \begin{bmatrix} x_{tt} \\ y_{tt} \\ z_{tt} \end{bmatrix} - d_6 \begin{bmatrix} r_{13} \\ r_{23} \\ r_{33} \end{bmatrix}$$

From this, calculate the first 3 joint positions (4 possible combinations):

$$q_{11} = atan2(x_w, y_w)$$

$$q_{12} = \pi + atan2(x_w, y_w)$$

Next, define:

$$D = \frac{x_c^2 + y_w^2 - d_2^2 + (z_c - d_1)^2 - a_2^2 - a_3^2}{2a_2a_3}$$

The 2 solutions of q_3 are then given by:

$$q_{31} = atan2\left(D, \sqrt{1 - D^2}\right)$$

$$q_{32} = atan2\left(D, -\sqrt{1 - D^2}\right)$$

And the solutions of q_2 are calculated as:

$$q_{21} = atan2\left(\sqrt{x_w^2 + y_w^2 - d_2^2}, z_w - d_1\right) - atan2\left(a_2 + a_3\cos(q_{31}), a_3\sin(q_{31})\right)$$

$$q_{22} = atan2\left(\sqrt{x_w^2 + y_w^2 - d_2^2}, z_w - d_1\right) - atan2\left(a_2 + a_3\cos(q_{32}), a_3\sin(q_{32})\right)$$

This gives a total of 4 possible solutions for the arm joint angles:

$$(q_{11}, q_{21}, q_{31})$$

$$(q_{11}, q_{22}, q_{32})$$

$$(q_{12}, q_{21}, q_{31})$$

$$(q_{12}, q_{22}, q_{32})$$

For each arm joint solution, there are 2 possible wrist joint solutions. For each solution, there are 2 possible cases. The first case is if one of the following conditions is true:

$$\cos(q_1)\cos(q_2 + q_3)\, r_{13} + \sin(q_1)\cos(q_2 + q_3)\, r_{23} + \sin(q_2 + q_3)\, r_{33} \neq 0$$

$$-\cos(q_1)\sin(q_2 + q_3)\, r_{13} - \sin(q_1)\sin(q_2 + q_3)\, r_{23} + \cos(q_2 + q_3)\, r_{33} \neq 0$$

In this case, the wrist joint solutions are as follows:

$$q_{51} = atan2\left(\sin(q_1)\, r_{13} - \cos(q_1)\, r_{23}, \sqrt{1 - \left(\sin(q_1)\, r_{13} - \cos(q_1)\, r_{23}\right)^2}\right)$$

$$q_{41} = atan2(\cos(q_1)\cos(q_2 + q_3)\, r_{13} + \sin(q_1)\cos(q_2 + q_3)\, r_{23} + \sin(q_2 + q_3)\, r_{33},$$
$$-\cos(q_1)\sin(q_2 + q_3)\, r_{13} - \sin(q_1)\sin(q_2 + q_3)\, r_{23} + \cos(q_2 + q_3)\, r_{33})$$

$$q_{61} = atan2(-\sin(q_1)\, r_{11} + \cos(q_1)\, r_{21}, \sin(q_1)\, r_{12} - \cos\,(q_1) r_{22})$$

And

$$q_{52} = atan2\left(\sin(q_1)\, r_{13} - \cos(q_1)\, r_{23}, -\sqrt{1 - (\sin(q_1)\, r_{13} - \cos(q_1)\, r_{23})^2}\right)$$

$$\begin{aligned} q_{42} = atan2(&-\cos(q_1)\cos(q_2 + q_3)\, r_{13} - \sin(q_1)\cos(q_2 + q_3)\, r_{23} \\ &- \sin(q_2 + q_3)\, r_{33}), \cos(q_1)\sin(q_2 + q_3)\, r_{13} + \sin(q_1)\sin(q_2 + q_3)\, r_{23} \\ &- \cos(q_2 + q_3)\, r_{33}) \end{aligned}$$

$$q_{61} = atan2(\sin(q_1)\, r_{11} - \cos(q_1)\, r_{21}, -\sin(q_1)\, r_{12} + \cos\,(q_1) r_{22})$$

Otherwise, the solutions are:

$$q_{51} = 0$$

$$\begin{aligned} q_{41} + q_{61} = atan2(&\cos(q_1)\cos(q_2 + q_3)\, r_{11} + \sin(q_1)\cos(q_2 + q_3)\, r_{21} \\ &+ \sin(q_2 + q_3)\, r_{31}, -\cos(q_1)\sin(q_2 + q_3)\, r_{11} - \sin(q_1)\sin(q_2 + q_3)\, r_{21} \\ &+ \cos(q_2 + q_3)\, r_{31}) \end{aligned}$$

Where q_{41} and q_{61} can be chosen arbitrarily to satisfy this equation.

The other solution is

$$q_{52} = \pi$$

$$\begin{aligned} q_{42} - q_{62} = atan2(&-\cos(q_1)\cos(q_2 + q_3)\, r_{11} - \sin(q_1)\cos(q_2 + q_3)\, r_{21} \\ &- \sin(q_2 + q_3)\, r_{31}, -\cos(q_1)\cos(q_2 + q_3)\, r_{12} - \sin(q_1)\cos(q_2 + q_3)\, r_{22} \\ &- \sin(q_2 + q_3)\, r_{32}) \end{aligned}$$

Where q_{42} and q_{62} can be chosen arbitrarily to satisfy this equation.

Two wrist joint solutions can be calculated for each of the four sets of arm joint solutions for a total of eight possible inverse kinematics solutions.