

**OPTIMISED MULTI-ROBOT PATH PLANNING  
VIA SMOOTH TRAJECTORY GENERATION**

**LOKE ZHI YU**

**UNIVERSITI TUNKU ABDUL RAHMAN**

**OPTIMISED MULTI-ROBOT PATH PLANNING  
VIA SMOOTH TRAJECTORY GENERATION**

**LOKE ZHI YU**

**A project report submitted in partial fulfilment of the  
requirements for the award of Bachelor of Mechatronics  
Engineering with Honours**

**Lee Kong Chian Faculty of Engineering and Science  
Universiti Tunku Abdul Rahman**

**May 2024**

## DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.



Signature : \_\_\_\_\_

Name : Loke Zhi Yu


ID No. : 1903026


Date : 24 April 2024

**APPROVAL FOR SUBMISSION**

I certify that this project report entitled “**OPTIMISED MULTI-ROBOT PATH PLANNING VIA SMOOTH TRAJECTORY GENERATION**” was prepared by **LOKE ZHI YU** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Mechatronics Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature :   
\_\_\_\_\_  
Supervisor : Dr Shalini a/p Darmaraju  
Date : 14 May 2024  
\_\_\_\_\_

Signature :   
\_\_\_\_\_  
Co-Supervisor : Kwan Ban Hoe  
Date : 14 May 2024  
\_\_\_\_\_

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2024, Loke Zhi Yu. All right reserved.

## ACKNOWLEDGEMENTS

I would like to express my sincere thanks to everyone who helped me with this project. First, I am deeply grateful to my supervisor, Dr. Shalini Darmaraju and co-supervisor, Dr. Kwan Ban Hoe, for their valuable guidance, patience, and support throughout the project. Their advice and weekly meetings were extremely helpful in overcoming challenges and completing the project successfully.

I am also thankful to my parents and friends for their constant encouragement and support during this entire journey. Their belief in me kept me motivated and determined. The completion of this project would not have been possible without the collective efforts and contributions of these individuals. I am truly grateful for their guidance, support, and belief in me.

## ABSTRACT

The deployment of multi-robot system (MRS) in real-world applications like warehouses and manufacturing plants has increased the importance of path planning algorithms for MRS. Compared to a single robot, an MRS is more effective and robust in completing tasks, even when one robot breaks down. Particle swarm optimization (PSO) outperforms conventional methods like artificial potential fields (APF), the Dijkstra algorithm, and the A\* algorithm in path planning for mobile robots. PSO focuses on finding the local and global best position of each particle through iterations, calculated based on a fitness function whereby the Euclidean distance between a particle's next waypoint and the target point is calculated. However, there is a need for optimizing smooth trajectory generation in multi-robot path planning. The application of parametric curves like the Bezier curve, Dubin's curve, and non-uniform rational B-spline (NURBS) curve is common for generating smooth trajectories. This project uses the Bezier curve equation for smooth trajectory generation as it is computationally inexpensive and easy to form desired curves. Smooth trajectories enable efficient traversal, shorter travel times, and energy conservation by limiting unnecessary movements and abrupt changes in direction. Collision avoidance is achievable through careful coordination of robot trajectories, preventing collisions and improving MRS safety. This project develops an enhanced PSO algorithm (EPSO) for smooth trajectory generation of MRS, aiming to reduce path length, execution time, and turn points, thereby increasing efficiency and conserving energy. A MPSO algorithm, without path smoothening, is used for comparison. EPSO parameters like swarm size, control points, inertia weight, and acceleration coefficients are tuned appropriately. Simulations for MPSO and EPSO are conducted five times for average results. In conclusion, EPSO outperforms MPSO in generating pathways with shorter path length, lower execution time, and fewer turn points, making it an effective solution for optimizing multi-robot path planning.

## TABLE OF CONTENTS

<b>DECLARATION</b>		<b>i</b>
<b>APPROVAL FOR SUBMISSION</b>		<b>ii</b>
<b>ACKNOWLEDGEMENTS</b>		<b>iv</b>
<b>ABSTRACT</b>		<b>v</b>
<b>TABLE OF CONTENTS</b>		<b>vi</b>
<b>LIST OF TABLES</b>		<b>viii</b>
<b>LIST OF FIGURES</b>		<b>ix</b>
<b>LIST OF SYMBOLS / ABBREVIATIONS</b>		<b>xiii</b>
<b>LIST OF APPENDICES</b>		<b>xiv</b>
 <b>CHAPTER</b>		
<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 General Introduction	1
	1.1.1 Multi-Robot Path Planning	2
	1.1.2 Multi-Robot Trajectory Smoothing	3
	1.2 Importance of the Study	5
	1.3 Problem Statement	6
	1.4 Aim and Objectives	7
	1.5 Scope and Limitation of the Study	7
<b>2</b>	<b>LITERATURE REVIEW</b>	<b>9</b>
	2.1 Introduction	9
	2.2 AI-Based Approaches	9
	2.3 Artificial Neural Network	10
	2.4 Bio-Inspired Algorithms	11
	2.4.1 Genetic Algorithm (GA)	11
	2.4.2 Particle Swarm Optimization Algorithm (PSO)	14
	2.4.3 Ant Colony Optimization Algorithm (ACO)	20
	2.4.4 Artificial Fish Swarm Algorithm (AFSA)	22
	2.5 Path Planning via Smooth Trajectory Generation	25
	2.6 Summary	42



<b>3</b>	<b>METHODOLOGY AND WORK PLAN</b>	<b>45</b>
3.1	Introduction	45
3.2	Modified Particle Swarm Optimization (MPSO)	45
3.3	Proposed Enhanced Particle Swarm Optimization (EPSO)	48
3.3.1	Assumptions	48
3.3.2	Path Planning Scheme	48
3.3.3	Trajectory Smoothing with Bezier Curve	50
3.3.4	Flowchart of EPSO Path Planning Algorithm	55
3.4	Planning and Managing of Project Activity	56
3.4.1	Gantt Chart of FYP 1	58
3.4.2	Gantt Chart of FYP 2	59
3.4.3	Summary	60
<b>4</b>	<b>RESULTS AND DISCUSSION</b>	<b>61</b>
4.1	Introduction	61
4.2	EPSO Parameters	61
4.2.1	Swarm size	61
4.2.2	Number of Control Points of Bezier Curve	64
4.2.3	Inertia weight, Cognitive and Social Acceleration Coefficients	66
4.3	Simulation Results of MPSO and EPSO	71
4.3.1	Comparison between MPSO and EPSO in Environment 1	73
4.3.2	Comparison between MPSO and EPSO in Environment 2	85
4.3.3	Comparison between MPSO and EPSO in Environment 3	97
4.4	Summary	110
<b>5</b>	<b>CONCLUSIONS AND RECOMMENDATIONS</b>	<b>112</b>
5.1	Conclusions	112
5.2	Recommendations for Future Work	113
	<b>REFERENCES</b>	<b>114</b>
	<b>APPENDICES</b>	<b>119</b>

## LIST OF TABLES

Table 2.1 :	Path Length (m) under Different Parameters of Simulation Environments.	38
Table 2.2 :	Advantaged and Disadvantages of Different Path Planning Algorithms.	43
Table 3.1 :	Gantt Chart of FYP 1.	58
Table 3.2 :	Gantt Chart of FYP 2.	59
Table 4.1 :	Simulation Results for Different Swarm Size.	62
Table 4.2 :	Simulation Result for Different Control Points.	65
Table 4.3 :	Simulation Result for Different $c_1$ , $c_2$ , and $\omega$ Values.	70
Table 4.4 :	Parameters of EPSO and MPSO.	73
Table 4.5 :	Starting Position and Ending Position for each Robots.	73
Table 4.6 :	Results of MPSO and EPSO in Environment 1.	83
Table 4.7 :	Result of MPSO for each Simulation in Environment 1.	83
Table 4.8 :	Result of EPSO for each Simulation in Environment 1.	84
Table 4.9 :	Result of MPSO and EPSO in Environment 2.	95
Table 4.10:	Result of MPSO for each Simulation in Environment 2.	96
Table 4.11:	Result of EPSO for each Simulation in Environment 2.	96
Table 4.12:	Result of MPSO and EPSO of Environment 3.	108
Table 4.13:	Result of MPSO for each Simulation in Environment 3.	109
Table 4.14:	Result of EPSO for each Simulation in Environment 3.	109

## LIST OF FIGURES

Figure 1.1 :	Taxonomy of a MRS (Farinelli, Iocchi and Nardi, 2004).	2
Figure 1.2 :	Comparison of Straight and Smooth Path (Ravankar et al., 2018).	4
Figure 1.3 :	MRS formed by UAVs (Madridano et al., 2021a).	5
Figure 1.4 :	MRS formed by UGVs (Madridano et al., 2021a).	5
Figure 2.1 :	Classification of Path Planning Techniques.	9
Figure 2.2 :	A Simplified Neural Network Model (Yi-Wen and Wei-Yu, 2015)	11
Figure 2.3 :	The Architecture of Path Planning via GA (Shuhua, Yantao and Jinfang, 2004).	13
Figure 2.4 :	Free-Space Histogram (Bilbeisi Ghaith, Al-Madi and Awad, 2015).	16
Figure 2.5 :	Free-Space Force (Bilbeisi Ghaith, Al-Madi and Awad, 2015).	16
Figure 2.6 :	Robot Swarm Navigation (Paez et al., 2021).	19
Figure 2.7 :	PRM Construction (Mbemba, Chen and Shu, 2022).	20
Figure 2.8 :	The Step Size Range of 16-Direction 24-Neighbourhood (Li, Du and Jia, 2022).	24
Figure 2.9 :	The Step Size Range of 4-Direction 4-Neighbourhood (Li, Du and Jia, 2022).	25
Figure 2.10:	Bezier Curve (Ravankar et al., 2018).	26
Figure 2.11:	Path Smoothing using Dubin's Curve (Ravankar et al., 2018).	27
Figure 2.12:	Pathway Generated by Traditional Path Planning Method (Jianwei et al., 2020).	30
Figure 2.13:	Pathway Generated by Bezier Curve Smoothing (BCA) Algorithm (Jianwei et al., 2020).	30
Figure 2.14:	Bezier Transition Curves (Xu, Song and Cao, 2021).	34

Figure 2.15:	Smooth Path Produced by PCBC + PSO-ADV (Xu, Song and Cao, 2021).	36
Figure 2.16:	Smooth Path Produced by SBC + MDPSO (Xu, Song and Cao, 2021).	36
Figure 2.17:	Path Smoothed by Bezier Curve (Mbemba, Chen and Shu, 2022).	38
Figure 2.18:	The Bezier Curve Model (Li, Du and Jia, 2022).	40
Figure 3.2 :	Kinematics of a TMR (Yang, Lee and Ryuh, 2013).	51
Figure 3.3 :	Bezier Curve-based Path Planning (Yang, Lee and Ryuh, 2013).	52
Figure 3.4 :	Flowchart of EPSO Path Planning Algorithm.	55
Figure 4.1 :	Chart of Result for Different Swarm Sizes.	63
Figure 4.2 :	Line Graph of Result of Different Number of Control Points.	66
Figure 4.3 :	Path Generated from high $c_1$ , low $c_2$ , and high $\omega$ .	68
Figure 4.4 :	Path Generated from high $c_1$ , low $c_2$ , and low $\omega$ .	68
Figure 4.5 :	Line Graph of Results for Different $c_1$ , $c_2$ , and $\omega$ Values.	70
Figure 4.6 :	Environment 1.	74
Figure 4.7 :	Pathway Generated for Robot 1 of MPSO.	75
Figure 4.8 :	Pathway Generated for Robot 2 of MPSO.	75
Figure 4.9 :	Pathway Generated for Robot 3 of MPSO.	76
Figure 4.10:	Simulation of Robots Travelling According to MPSO Pathway.	77
Figure 4.11:	Plotting of MPSO Waypoints of all Robots.	77
Figure 4.12:	Graph of Global Best Fitness Vs. Iterations of MPSO Algorithm.	78
Figure 4.13:	Pathways Generated for Robot 1 of EPSO.	79
Figure 4.14:	Pathways Generated for Robot 2 of EPSO.	79
Figure 4.15:	Pathways Generated for Robot 3 of EPSO.	80

Figure 4.16:	Simulation of Robots Travelling According to EPSO Pathway.	81
Figure 4.17:	Plotting of EPSO Waypoints of all Robots.	81
Figure 4.18:	Graph of Global Best Fitness Vs. Iterations of EPSO Algorithm.	82
Figure 4.19:	Error Bar of EPSO Path Length for Robot 1, 2, and 3.	84
Figure 4.20:	Error Bar of EPSO Execution Time for Robot 1, 2, and 3.	85
Figure 4.21:	Environment 2.	86
Figure 4.22:	Pathway Generated for Robot 1 of MPSO.	87
Figure 4.23:	Pathway Generated for Robot 2 of MPSO.	87
Figure 4.24:	Pathway Generated for Robot 3 of MPSO.	88
Figure 4.25:	Simulation of Robots Travelling According to MPSO Pathway.	89
Figure 4.26:	Plotting of MPSO Waypoints of all Robots.	89
Figure 4.27:	Graph of Global Best Fitness Vs. Iterations of MPSO Algorithm.	90
Figure 4.28:	Pathways Generated for Robot 1 of EPSO.	91
Figure 4.29:	Pathways Generated for Robot 2 of EPSO.	91
Figure 4.30:	Pathways Generated for Robot 3 of EPSO.	92
Figure 4.31:	Simulation of Robots Travelling According to EPSO Pathway.	93
Figure 4.32:	Plotting of EPSO Waypoints of all Robots.	93
Figure 4.33:	Graph of Global Best Fitness Vs. Iterations of EPSO Algorithm.	94
Figure 4.34:	Error Bar of EPSO Path Length for Robot 1, 2, and 3.	97
Figure 4.35:	Error Bar of EPSO Execution Time for Robot 1, 2, and 3.	97
Figure 4.36:	Environment 3.	98
Figure 4.37:	Pathway Generated for Robot 1 of MPSO.	99

Figure 4.38:	Pathway Generated for Robot 2 of MPSO.	100
Figure 4.39:	Pathway Generated for Robot 3 of MPSO.	100
Figure 4.40:	Simulation of Robots Travelling According to MPSO Pathway.	101
Figure 4.41:	Plotting of MPSO Waypoints of all Robots.	102
Figure 4.42:	Graph of Global Best Fitness Vs. Iterations of MPSO Algorithm.	103
Figure 4.43:	Pathways Generated for Robot 1 of EPSO.	104
Figure 4.44:	Pathways Generated for Robot 2 of EPSO.	104
Figure 4.45:	Pathways Generated for Robot 3 of EPSO.	105
Figure 4.46:	Simulation of Robots Travelling According to EPSO Pathway.	106
Figure 4.47:	Plotting of EPSO Waypoints of all Robots.	106
Figure 4.48:	Graph of Global Best Fitness Vs. Iterations of EPSO Algorithm.	107
Figure 4.49:	Error Bar of EPSO Path Length for Robot 1, 2, and 3.	110
Figure 4.50:	Error Bar of EPSO Execution Time for Robot 1, 2, and 3.	110

## LIST OF SYMBOLS / ABBREVIATIONS

MRS	Multi-robot system
UAV	Unmanned aerial vehicle
AGV	Automated Guided Vehicle
UGV	Unmanned Ground Vehicle
AUV	Autonomous Underwater Vehicle
APF	Artificial potential field
EPSO	Enhanced particle swarm optimisation
MILP	Mixed integer linear program method
OC	optimal control method
MIQP	Mixed integer quadratic program
ANN	Artificial neural network
MPCNN	Modified pulse-coupled neural network
AFSA	Artificial fish swarm algorithm
GA	Genetic algorithm
ACO	Ant colony optimisation algorithm
TC	Time complexity
BCA	Bezier curve smoothing algorithm
AG	Agoraphilic algorithm
FSH	Free-space histogram
FSF	Free-space force
D2PSO	Dynamic distributed particle swarm optimization
D-PSO	Distributed particle swarm optimization
PSO-ADV	Particle swarm optimization with adaptive delayed velocity
PCBC	Parametric cubic bezier curve
SBC	Square bezier curve
PRM	Probabilistic roadmap
$G^i$	Geometric continuity
$C^i$	Parametric continuity
NURBS	Non-uniform rational B-spline curve
MPSO	Modified particle swarm optimisation algorithm
TMR	Two-wheeled mobile robot

## LIST OF APPENDICES

Appendix A : Matlab Coding	119
----------------------------	-----



## CHAPTER 1

### INTRODUCTION

#### 1.1 General Introduction

Researchers have been motivated by the concept of creating robot groups that can work together on certain tasks since the late 1980s. Scientists have studied how groups of distinct organisms may work together to pursue shared goals by referring to natural occurrences like swarms of bees, ant colonies, schools of fish, and even human groupings. This knowledge has been used to a number of practical fields, such as item transportation, foraging, exploration, search and rescue operations, and surveillance in the real world (N. Darmanin and K. Bugeja, 2017).

Coordination and communication are the core components of a multi-robot system (MRS), as shown in Figure 1.1. The cooperation level is located on the top level of the hierarchy. In this level, robots work together within a cooperative system to complete global tasks. MRS may be considered as a group of cooperative robots. The knowledge level is the second level of the hierarchy. Robots that are "Aware" have some knowledge of their groupmates, but "Unaware" robots do not possess any knowledge of other robots in the system. A MRS can nevertheless be aware even when there is no direct inter-robot communication. The coordination level corresponds to the third level. Depending on whether it relies on a coordination protocol, coordination might be rated as strong or weak. In simple terms, a coordination protocol is a collection of rules that specify how robots should interact with one another in their environment. The organization level, which explores the MRS's decision-making architecture, is the subject of the fourth level. A centralized system is a system whereby the team's decision-making process is coordinated and directed by the leader, and other team members follow instructions. A distributed system, on the other hand, consists of autonomous agents that act independently of one another. There is no leader position in the system. Robot cooperation typically involves the use of a communication system that allows for the exchange of messages which can be classified as direct and indirect

communication. Direct communication calls for the use of specialized onboard hardware, which is more expensive and less dependable, to develop robot coordination. Conversely, stigmergy is used in indirect communication. Stigmergic communication creates a decentralized, concurrent framework for group communication that is readily accessible to each agent. The requirement for agent synchronization is eliminated by this architecture (Farinelli, Iocchi and Nardi, 2004).

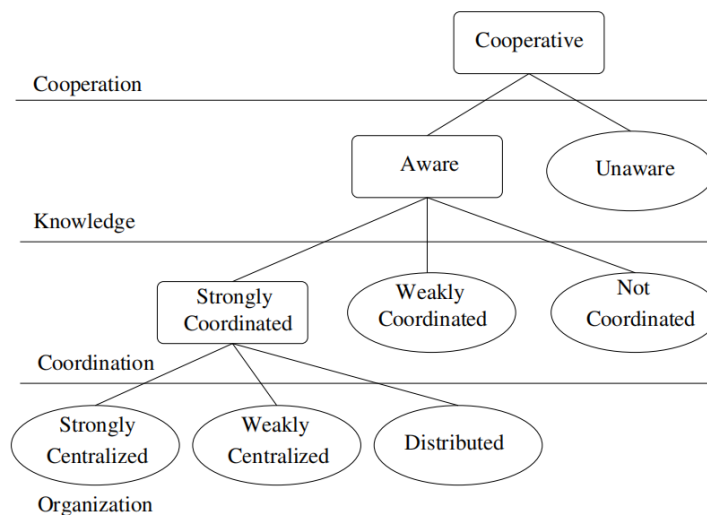


Figure 1.1: Taxonomy of a MRS (Farinelli, Iocchi and Nardi, 2004).

### 1.1.1 Multi-Robot Path Planning

Path planning is a crucial step in any multi-robot system (MRS). The goal of path planning is to compute a series of obstacle-free configurations that begin at a specified beginning point and conclude at the desired location. While smoothness is not always prioritised, finding a raw solution is the main goal, which then has to be optimised using cost function to get the optimal outcome. Smooth trajectory generation is the next step in optimum path planning. This establishes not just the paths that robots take but also considers parameters like speed, acceleration, and kinodynamic constraints as the robots travel the paths (Madridano et al., 2021a).

There are several ways to measure a path planning algorithm's performance. Path length, computational speed, path smoothness, energy efficiency, and safety are a few of the widely acknowledged measures. The

optimal path planning algorithm should provide a high-quality path with the shortest length and better smoothness while consuming the least amount of energy and taking the shortest amount of time to execute. But concurrently improving these indicators is a difficult challenge. Iterative processes are a common method used by path planning algorithms, and as the number of iterations increases, shorter and smoother pathways are produced. The map's resolution is another aspect that affects the path planning algorithm's performance. The use of higher map resolutions might result in greater path quality but at the same time increases the processing time (Abujabal et al., 2023a). Feasibility and optimality are two distinct approaches used to assess path planning results. Feasibility places less emphasis on efficiency and more on finding a safe path for robots. Optimality concentrates on developing optimal and efficient paths to reach the goal (Madridano et al., 2021a).

In a two-dimensional or three-dimensional workspace, the robots and obstacles are presented. To estimate their current position, robots need to be able to locate themselves in the workspace. Simultaneous Localization and Mapping (SLAM) is in charge of creating a workspace of the environment while also determining the location of the robot inside it. The robot will navigate to the given target point using a variety of path planning algorithms, beginning from the current robot position (Ravankar et al., 2018). Then, the configuration space will store all all possible configurations. The obstacle space corresponds to configurations located within obstacle areas. The obstacle-free space, on the other hand, is the complement of the obstacle space and includes configurations located within free spaces where the robot may move through them safely. The desired configuration is described by the goal space, which is a subset of the obstacle-free space (Abujabal et al., 2023a).

### **1.1.2 Multi-Robot Trajectory Smoothing**

Sharp turns and straight segments are common characteristics of the paths produced by various path planning algorithms. Consider a robot with its specified target position, as shown in Figure 1.2, the green path serves as an example of a path with straight segments. However, since the robot is unable to execute abrupt direction change instantaneously, such a path is

inappropriate for its mobility. Executing quick turns may even be difficult in specific circumstances, depending on the robot's kinematics (Ravankar et al., 2018).

In contrast, the red path in Figure 1.2, is ideal for efficient robot navigation. This kind of path prevents abrupt and quick shifts in direction, allowing the robot to move forward without stopping. Smooth trajectories must meet a set of requirements that include tangential continuity, curvature continuity, safety, compliance to robot motion models, and taking robot kinematics into account. Safety is ensured by making sure that the path produced does not intercept with obstacles. The motion model of a robot describes how its position and orientation change over time as it moves through the environment, allowing one to predict the robot's future positions based on its current state (Ravankar et al., 2018).

The differential drive model, holonomic model, kinematic bicycle model, and dynamic model are common categories of robot motion models. To predict changes in position and orientation, the differential drive model takes into account the robot's linear and angular velocities, wheelbase, and wheel radius. In contrast, robot kinematics focus on comprehending how the robot's joints move to direct its end effector. For example, turning radius differs for differential drive robots, Ackermann steering robots, and omni-directional robots. This turning radius controls how sharply a robot can turn or change its direction, which affects the trajectory that the robot can travel. A robot that can maneuver through narrow spaces and make accurate turns is one that has a smaller turning radius (Ravankar et al., 2018).

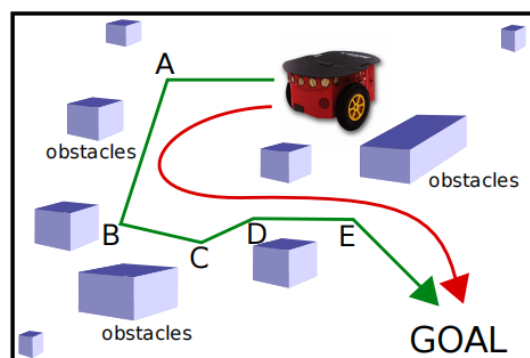


Figure 1.2: Comparison of Straight and Smooth Path (Ravankar et al., 2018).

## 1.2 Importance of the Study

Applications for robots have been widely used in a variety of fields, including industry, agriculture, surveillance, search and rescue work, environmental monitoring, and traffic management. An AI system that combines microelectronics, communication, computer science, and optics is referred to as a robot. Unmanned aerial vehicles (UAVs) for airspace as shown in Figure 1.3 (Madridano et al., 2021a), Automated Guided Vehicles (AGVs) for manufacturing plants, Unmanned Ground Vehicles (UGVs) for ground mission as shown by Figure 1.4 (Madridano et al., 2021a), and Autonomous Underwater Vehicles (AUVs) for underwater exploration are a few examples of mobile robots that have been deployed as a result of advancements in robotics technology.



Figure 1.3: MRS formed by UAVs (Madridano et al., 2021a).



Figure 1.4: MRS formed by UGVs (Madridano et al., 2021a).

It is apparent that using multi-robot system (MRS) is more cost-effective than building a single, costly robot that is capable of performing every task. These systems are frequently decentralized, distributed, and intrinsically redundant, which promotes system reliability and robustness and provides fault tolerance (Gautam and Mohan, 2012). Robots have the ability to cover large regions while working simultaneously. Benefits including robustness, flexibility, scalability, and spatial dispersion are demonstrated by the system's capacity to continue operating even if one robot breaks

down. Each robot in a multi-robot system has unique coordinates and autonomous behaviour, enabling it to imitate the cooperative behaviour observed in real-world situations (Lin et al., 2022).

Particle swarm optimisation and genetic algorithm outperform more conventional methods like artificial potential fields (APF), graph-based searches (Dijkstra, A\*, and D\*), and sampling-based strategies in path planning to create a smooth trajectory for mobile robots inside a workspace. The main objective of modern metaheuristic techniques is to avoid being captured in local minimum traps (Dian et al., 2022). But there has been a noticeable shift towards the increasing importance of optimizing path planning in multi-robot system through smooth trajectory generation. The application of suitable parametric curve to determine the trajectory is a common component of optimization techniques for generating smooth trajectories. Due to its deep influence on the overall performance, efficiency, safety, and scalability of such systems, researching the area of smooth trajectory generation within multi-robot system is of utmost importance. The use of smooth trajectories enables robots to traverse with increased efficiency, resulting in shorter travel times and the conservation of energy resources by limiting unnecessary movements, sudden changes in direction, and halts. Furthermore, collision avoidance is achievable as path planning via smooth trajectory generation is used to carefully coordinate the trajectories of various robots, preventing collisions and thus improving the safety of a multi-robot system. Moreover, effective resource utilization results from the implementation of smooth trajectory path planning in a multi-robot system. This is because smooth trajectory reduces unnecessary movements and idle times for robots, optimizing the use of resources, including battery life and computational capacity. As a result, operational durations are increased, and system performance is improved.

### **1.3 Problem Statement**

The development of Unmanned Ground and Aerial Vehicles (UGVs and UAVs) has led to a significant increase in the deployment of multi-robot system (MRS) in real-world applications in recent years. This development has made it possible to employ many robots simultaneously and autonomously,

increasing their usefulness in crucial missions by providing better efficiency, persistence, and reliability. Path planning is a crucial component of efficient coordination and navigation in MRS. Creating pathways that allow robots to independently proceed from their starting point to their destination without requiring human intervention is known as path planning. It is essential for them to find a feasible and optimal pathway to reach their goal while ensuring the smoothness of path generated (Rao and Sodhi, 2022).

Hence, other features like path length, travel time and energy expenditure can be optimised for MRS deployment in the real world. There is many path planning algorithms that may be used to find the optimal route for a single robot that travels with the minimum amount of distance or time in polynomial time. However, when multiple robots are engaged, the computational complexity of finding optimal paths has been proven to be (non-deterministic polynomial-time) NP-complete which indicates that there is no known efficient method, that runs in polynomial time, for finding optimal pathways with smooth trajectories for multi-robot path planning in all situations (Rao and Sodhi, 2022).

#### **1.4 Aim and Objectives**

This project's goal is to provide an enhanced particle swarm optimisation (EPSO) method as a possible solution for multi-robot path planning via smooth trajectory generation. Reduction of path length, execution time, and number of turn points of robots will be focused in this project. The objectives of this project are shown as below:

- 1) To review on existing multi-robot path planning algorithms.
- 2) To develop an enhanced PSO (EPSO) algorithm for smooth trajectory generation of multi-robot system.
- 3) To evaluate performance of the enhanced PSO (EPSO) algorithm through simulation.

#### **1.5 Scope and Limitation of the Study**

Multi-robot path planning via smooth trajectory generation has several constraints and challenges. When more robots are implemented in the

environment, the computational time for robot pathways will increase. As a result, there can not be too many robots, and a 2D environment is used to reduce computational complexity. Additionally, each robot will have a different turning radius and kinematics constraints if it is of a different model (non-homogeneous), which will affect the degree to which it can turn and rotate. Therefore, it could be challenging to guarantee that non-homogeneous robots can successfully follow the smooth trajectories created by the same path planning algorithm. To simplify the case, homogeneous differential-drive two-wheeled robots are used in this project. Furthermore, the combination of static and dynamic obstacles in the workspace will increase the path planning complexity. In this project, a multi-robot system (MRS) will function in a static environment with just static obstacles to make the situation simpler.



## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Introduction

In this chapter, different path planning algorithms for multi-robot system (MRS) are classified, discussed, and compared in details. The current path planning techniques for MRS, as shown in Figure 2.1, can be divided into classical approaches, mathematical model based approaches, and AI-based approaches. The classical approaches can be further divided into decomposition graph-based methods, sampling based methods and artificial potential field (APF) method. Next, the mathematical model based methods can also be separated into mixed integer linear program (MILP) method, optimal control (OC) method, and mixed integer quadratic program (MIQP). Bioinspired algorithms and artificial neural network (ANN) are classified under the AI-based methods (Madridano et al., 2021b).

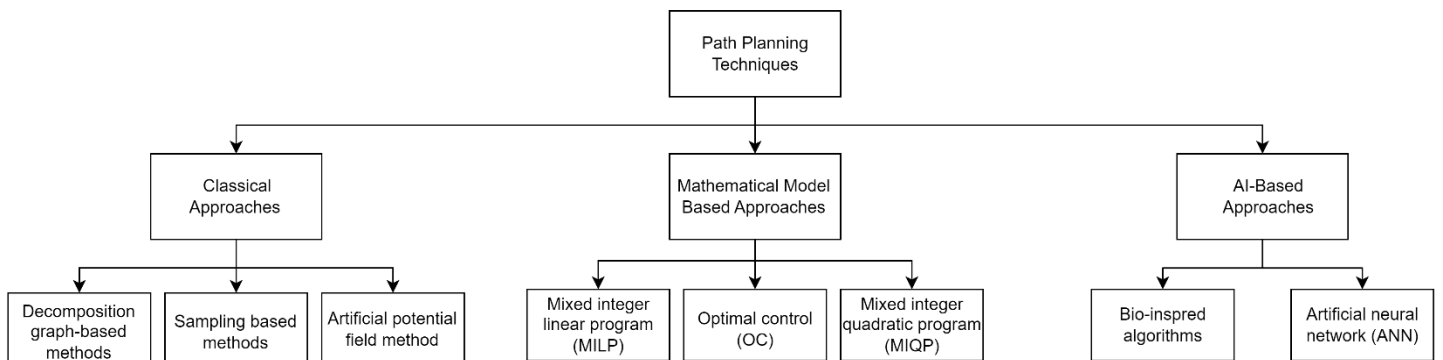


Figure 2.1: Classification of Path Planning Techniques.

#### 2.2 AI-Based Approaches

This section discusses AI-based techniques, such as artificial neural network (ANN) and the bio-inspired algorithms. Robotic planning of paths makes notable use of AI to make it easier for robots to independently navigate through diverse surroundings to perform duties (Rao et al., 2017). Robots frequently use methods like machine learning and computer vision to build on their path planning skills while gaining knowledge from what they have

previously learned. Furthermore, computer vision algorithms are applied to provide robots insight into their surroundings, enabling real-time and dynamic obstacle identification and avoidance (Abujabal et al., 2023b).

### 2.3 Artificial Neural Network

ANN serves as a functional framework made up of a large number of linked nodes, where each node corresponds to an activation function that delivers a particular output function. The artificial neural network's memory is denoted by the connections between nodes, which indicate the weighted values employed to the signals traversing through them (Tang and Ma, 2021).

A modified pulse-coupled neural network (MPCNN) algorithm is introduced to generate pathways for multiple robots in a warehouse or hypermarket. This modified algorithm will receive inputs, such as the origin, destination, static obstacles, which is transformed into a grid-based map by a map building algorithm. Then, the optimal path with the lowest required travelling time can be obtained through the modified MPCNN algorithm, which allows for the rapid map data processing. Meanwhile, dynamic obstacles can be avoided by installing infrared sensors on the robots. When there are dynamic obstacles such as human or other robots, the emitted infrared ray will get reflected back from the obstacles, thus the robot is able to determine the distance of obstacles through frequency changes in infrared waves. The robot will stop its motion to avoid collision with dynamic obstacles. Figure 2.2 showcases a simplified neural network model of the modified MPCNN algorithm. Point  $i$  represents the destination point, and  $Y_i$  stands for the status of point  $i$ . When  $Y_i$  reach an output of 1, the processing of point  $i$  is completed. If it is equal to zero, the processing of point  $i$  is awaited to be processed. The total distance of the pathway is  $d_{all,i}$ .  $R_i$  is the neighbouring points of point  $i$ . As seen in Figure 2.2, the modified MPCNN receives inputs such as the beginning point, obstacles, adjacent points, and destination point (Yi-Wen and Wei-Yu, 2015).

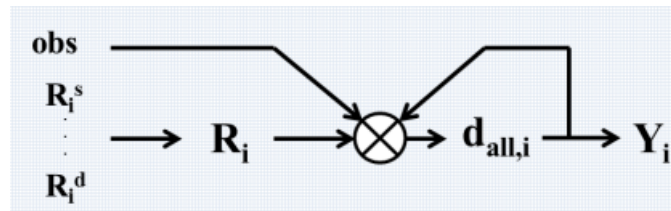


Figure 2.2: A Simplified Neural Network Model (Yi-Wen and Wei-Yu, 2015)

## 2.4 Bio-Inspired Algorithms

With the goal to build optimal pathways, bio-inspired algorithms try to mimic the behaviour and activities of biological creatures. These techniques have a number of important qualities, including parallel structures, flexibility, and quasi deterministic. These characteristics enable these algorithms to offer ideal solutions for path-finding issues without necessitating a thorough comprehension of the mission's surroundings. As a result, they are successful at solving multi-objective tasks (Guzmán and Peña, 2013).

Bio-inspired algorithms have an established procedure. In the beginning, potential solutions are chosen at random to create the first generation. The next phase is taking into account elements including the robot's capabilities, the goal, and any existing constraints in the environment or system as a whole. Then, a selection of the first generation's parents are picked to produce a new generation. The last phase involves repeating a mutation and crossover procedure until the desired outcome is achieved. Most outstanding individuals are ultimately translated and used as nodes to create the ideal pathway. The Particle Swarm Optimisation (PSO), Artificial Fish Swarm Algorithm (AFSA), Genetic Algorithm (GA), and Ant Colony Optimisation Algorithm (ACO) are a few examples of the methods that fall within the category of bio-inspired algorithms. The GA was the first bio-inspired algorithm that came to light, and as time went on, more approaches that drew influence from other natural processes also emerged (Madridano et al., 2021b).

### 2.4.1 Genetic Algorithm (GA)

The genetic algorithm (GA) utilises the based-knowledge genetic operators in robots path planning. Each robot independently plans its route using GA from

its initial location to the goal position which is also known as the decoupled planning technique. During this phase of individual planning, the existence of other robots is ignored. The resultant pathways are then mixed, and any possible collisions are dealt with by employing a reactive strategy. It is essential to recognise that algorithms using this technique are frequently insufficient, which means that even if a solution exists, they cannot guarantee to discover it. Also, GA has high computational complexity due to its genetic operators so the execution time of GA is often longer. However, the algorithm's complexity will not be affected when more robots are engaged. GA is also able to handle large and complicated maps (Shuhua, Yantao and Jinfang, 2004).

Figure 2.3 depicts the architecture of path planning using a genetic algorithm (GA). In the beginning, path planners—one for each goal—use GA to create paths for every robot to undertake in order to reach that goal and then transmit that information to every robot. There are four phases in this procedure. First, by including an active constraint, an initial population of chromosomes representing pathways is generated. The two dimension coordinates  $(i,j)$  are used to represent each chromosome. Second, two individuals are chosen using a roulette wheel selection process with an elitist model. Thirdly, two new individuals are formed using a crossover operator, making sure the crossover location is chosen outside of any connected part of a path. Finally, the two new individuals go through mutations respectively. To guarantee that the developed pathways remain linked based on the chromosomes' two-dimensional coordinates  $(i,j)$ , multi-point crossover and variable crossover rates are applied. Elitist selection is then used on the newly created population. The better individuals can be ensured to advance to the next generation by replacing the weaker individuals of the current generation with the superior individuals of the parent generation. Then, depending on its given goal and the field of vision, each local navigator, allocated to each robot, chooses a steering direction. By doing this, the robot is protected from running into any obstacles in the environment (Shuhua, Yantao and Jinfang, 2004).

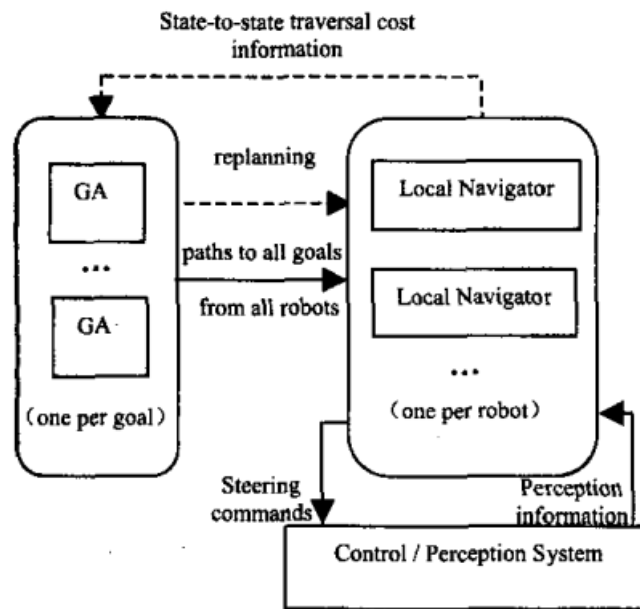


Figure 2.3 : The Architecture of Path Planning via GA (Shuhua, Yantao and Jinfang, 2004).

Genetic algorithm-based robot path planning (GA-RPP) technique often encounters the problem of premature convergence, where a solution pathway may become unfeasible upon termination, as an impact of its stochastic characteristic. It is essential to meticulously plan the evaluation and natural selection steps in order to ensure the population converges well to become a feasible and optimal solution pathway. A quick GA-RPP approach can be used to address this issue without increasing time complexity (TC) of GA. Since the fast GA-RPP is both straightforward and effective, it may be used right away in fields where commercial robots are now used, such as lawn trimming robots, medical robots, and service robots for life-saving rescue. Initialization, reproduction, proposed evaluation, and proposed natural selection are the stages included in the proposed method's workflow. An initial population  $P(t)$  is created during initialization by random in order to represent a route as a list of nodes. For instance, a path,  $Z = \{z_1, \dots, z_n\}$ . By applying crossover and mutation, reproduction can produce an offspring set  $N(t)$ . Then, using two evaluation functions,  $F(Z)$  as stated in Equation 2.1 and  $D(Z)$  as shown in Equation 2.2, respectively, the quantity of possible subpaths in  $P(t)$  and the length of each path can be evaluated. The last stage of natural

selection is the formation of  $P(t+1)$ , where pathways with lower  $F(Z)$  and  $D(Z)$  values are preferred over others. The fast GA-RPP method can increase the success rate to find a feasible path and shorten execution time without a raise in time complexity (TC) as compared to traditional GA (Lee, Kang and Kim, 2013).

$$F(Z) = \frac{\alpha(e_i)}{|Z|-1} \quad (2.1)$$

where

$F(Z)$  = infeasibility evaluation function

$\alpha(e_i)$  = number of subpaths

$Z$  = available paths

$$D(Z) = \sum_{e_i \in Z} euc(e_i) \quad (2.2)$$

where

$D(Z)$  = distance evaluation function

$euc(e_i)$  = Euclidean distance of the subpaths

#### 2.4.2 Particle Swarm Optimization Algorithm (PSO)

The path planning of multi-robot systems was addressed in 2015 by the PSO-AG algorithm, which combines the benefits of the agoraphilic (AG) algorithm with particle swarm optimisation (PSO), with PSO serves as the robot's path planner, and AG serves as its regulator, guiding the robot down the path while making sure it avoids obstacles. The evolutionary computation technique known as particle swarm optimisation (PSO) was motivated by the cooperative behaviour of natural swarms, such as bird flocks, which work together to obtain a global optimum or goal location. The PSO method begins with a start phase that randomly disperses particles throughout the search area, giving each one an initial velocity that is within the permissible range. Fitness assessments are carried out to identify the swarm's leader. The process then enters an iterative phase, updating particle velocities, positions, and fitness values at each step till the predetermined number of iterations is attained. The updated

velocity of particles is shown in Equation 2.3. Each particle modifies its position based on Equation 2.4 and reevaluates fitness based on the objective function as shown in Equation 2.5 in response to velocity changes. The new position is then evaluated to see if it can become its personal best position. Each particle uses its own best position to move towards the target while searching for a path that ultimately leads to the global best position (Bilbeisi Ghaith, Al-Madi and Awad, 2015).

$$v_i = wv_i + c_1r_1(p_i - x_i) + c_2r_2(p_g - x_i) \quad (2.3)$$

where

- $v_i$  = updated velocity of the robot
- $wv_i$  = inertia weight set to the velocity of the particle before update
- $c_1$  = parameter for setting weight of own best position
- $p_i$  = the particle's own best fitness
- $x_i$  = current position of the robot
- $c_2$  = parameter for setting weight of global best position
- $p_g$  = the particle's global best fitness

$$x_i = x_i + v_i \quad (2.4)$$

where

- $x_i$  = current position of the robot
- $v_i$  = current velocity of the robot

$$D = \sqrt{(x_i - x_t)^2 + (y_i - y_t)^2} \quad (2.5)$$

where

- D = the Euclidian distance from the target
- $(x_i, y_i)$  = coordinates of the robot
- $(x_t, y_t)$  = coordinates of the target location

AG first guides the robot to the new location by detecting surroundings within its sensor coverage. If the path is clear, the robot may go right away in the direction of the new point. However, AG employs a method to get to the new point when the robot is blocked by an obstacle. As seen in Figure 2.4, the Free-Space Histogram (FSH) is created by measuring the distances between the robot and the obstacles in its immediate environment. This map shows the distance profile of the robot's surroundings. To drive the robot, initial forces are calculated for every component in the FSH, and the magnitude of free-space forces (FSF) is directly correlated with the square of the distance measured as shown in Figure 2.5 (Bilbeisi Ghaith, Al-Madi and Awad, 2015).

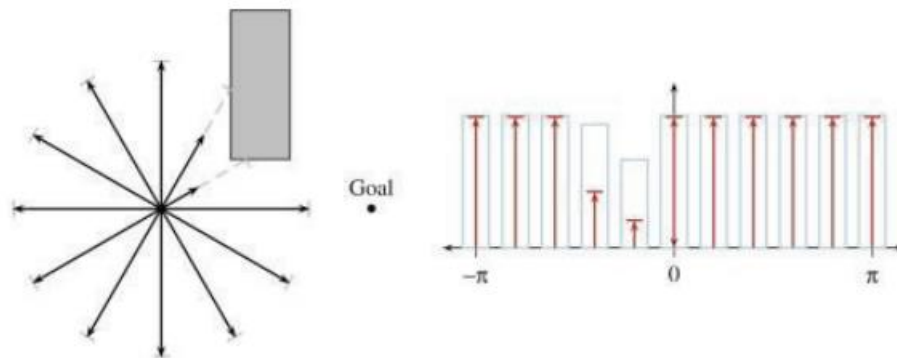


Figure 2.4: Free-Space Histogram (Bilbeisi Ghaith, Al-Madi and Awad, 2015).

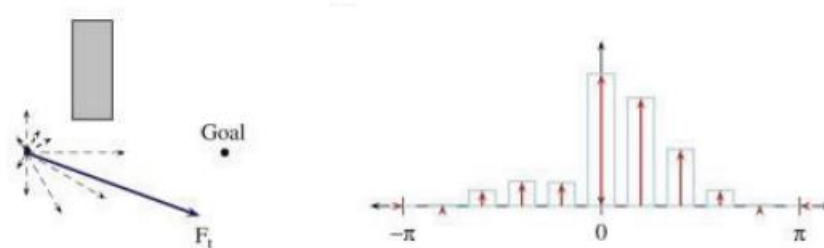


Figure 2.5: Free-Space Force (Bilbeisi Ghaith, Al-Madi and Awad, 2015).

The ideal route for a distributed multi-robot system is determined using another innovative approach called Dynamic Distributed Particle Swarm Optimisation (D2PSO). Every robot in this system fulfils the duties of a mobile, independent agent. D2PSO strives to preserve swift convergence capabilities while addressing the challenges associated with local optima and stagnation that bother the conventional PSO algorithms. Meanwhile, it also



maintains and applies the fundamental principles of PSO. It also shows scalability, being able to control multiple robots without increasing their need for communication. To achieve these enhancements, D2PSO adds two additional parameters to the PSO which are the Local Optima Detector for personal best and the Local Optima Detector for global best. The number of iterations whereby there has been no improvement for the particles' personal bests and the global best is tracked by these parameters. When particles fail to improve on their personal best, they lose their potential to find the overall optimum solution. To combat this, D2PSO steers particles in the direction of potentially unknown areas, varying the search area and giving them an external force to increase their ability to search. Likewise, if the global best does not evolve after a specific number of repetitions, it could indicate that the system is stuck in a local optimum that is misdirecting other particles. In order to alleviate such negative effects, D2PSO solves this problem by using external force to relocate the trapped particle away from the position of the local optima (IEEE Robotics and Automation Society and Institute of Electrical and Electronics Engineers, n.d.).

Next, a distributed particle swarm optimization (D-PSO) is proposed in 2021. Around the central robot point, each member of the swarm is originally distributed at arbitrary in compliance with the Gaussian distribution function  $N(\mu, \sigma)$ . Here, the robot's localization point is represented by  $\mu$ , and its size is shown by  $\sigma$ . In order to create attractive forces towards unknown areas and nearby victims that have been detected by robots, artificial potential functions are utilised. Contrary to conventional PSO theory, a novel strategy is used, involving the introduction of a repulsion vector between particles in the same swarm as they get closer to a minimum distance as shown in Equation 2.6. This repulsion vector prohibits particles from converging at the same position. The addition of a repulsion vector between different swarms as shown in Equation 2.7 also makes a substantial contribution. A minimal distance between robots is maintained by the action of this repulsion vector between a swarm's centre point and particles from other swarms. Without this method, different swarms would converge towards the same place, increasing

the probability of robot collisions. The D-PSO equation is shown in Equation 2.8 (Paez et al., 2021).

$$Rp_n = \sum_{i=0}^k \|G_p(p_i - p_n)\| \quad (2.6)$$

where

$Rp_n$  = repulsion between particles in the same swarm  
 $\|G_p(p_i - p_n)\|$  = artificial repulsion force as a function of distance existed between the current particle and the rest in the same swarm

$$Rs_n = \sum_{i=0}^j \|G_s(gM_i - p_n)\| \quad (2.7)$$

where

$Rs_n$  = inter-swarms repulsion  
 $\|G_s(gM_i - p_n)\|$  = artificial repulsion function which depends on the distance

$$v_{n+1} = wv_n + c_1r_1\|pB_n - p_n\| + c_2r_2\|gB_n - p_n\| + c_3r_3Rp_n + c_4r_4Rs_n \quad (2.8)$$

where

$v_{n+1}$  = velocity of each particle in the next iteration  
 $w$  = inertia constant computer for the velocity of the particle  
 $v_n$  = velocity of each particle in the current iteration  
 $c_{1,2,3,4}$  = learning constant  
 $r_{1,2,3,4}$  = random value between 0 and 1  
 $\|pB_n - p_n\|$  = error between the historical best position of the swarm particle and its current position  
 $\|gB_n - p_n\|$  = difference between the best position among all particles swarm and the position of the current particle calculation  
 $Rp_n$  = sum of the repulsion force of the particle with respect to the particles from the same swarm

$RS_n$  = sum of repulsion to the average position of the other swarms

As indicated by Figure 2.6, 26 robots and three victims was tested in the simulation using Vrep. The robots arrived at the victims through several robot groupings, as was to be predicted. The three yellow dotted circles represent robots that have already discovered victims; these robots were kept at a safe distance from one another by the inter-swarm repulsive forces. These repelling factors caused other groups of robots to be driven to investigate and search for other possible victims once the first victim is discovered by one of the robot groups. The suggested approach successfully avoided collisions between the robots even when several approached a victim simultaneously. The three victims were therefore successfully discovered. The simulation showed that several robots kept travelling in search for other victims even after they located all three victims; these robots are depicted in figure 2.6 as circles with dotted white lines. By the end of the simulation, different groups of nine, four, and four robots had each located one of the three victims. This result demonstrates how robots possess an effective repelling mechanism that can keep them focused on finding victims while maintaining a safe distance between each other within the same swarm as well as among different swarms (Paez et al., 2021).

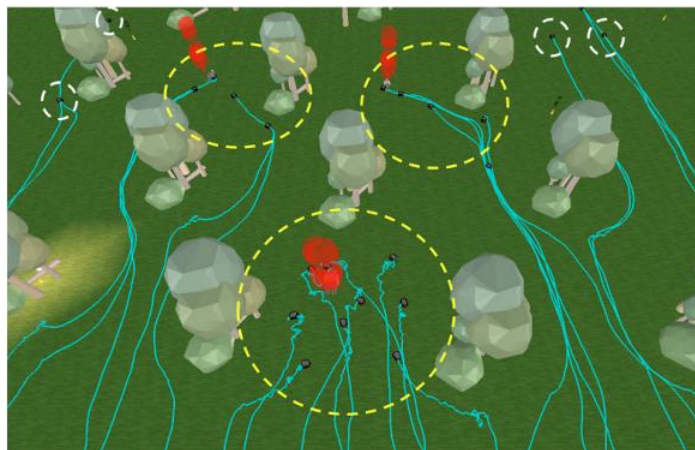


Figure 2.6: Robot Swarm Navigation (Paez et al., 2021).

### 2.4.3 Ant Colony Optimization Algorithm (ACO)

In 2022, a new path planning method that combines ant colony optimisation (ACO), and probabilistic roadmap (PRM) is presented. There are two basic steps in the process. Generating a random map using the probabilistic roadmap approach is the first stage. The path with the shortest distance between the initial point and destination is discovered in the second step using ant colony optimisation. A roadmap with edges (E) and nodes (N) represents the way a probabilistic roadmap (PRM) develops as shown in Figure 2.7. N is made up of randomly chosen nodes that are guaranteed not to run into any static obstacles in the surrounding area. These nodes are connected directly by edges (E), resulting in the roadmap  $R = (N, E)$ . Binary collision check or incremental collision check can both be used for collision checking. Edges are divided into numerous steps by incremental collision checking, which then examines for obstacles that the steps may collide onto. Contrarily, binary collision checking splits the edge into two parts and determines if the middle point is collision-free. If this is the case, the operation keeps going by splitting the edge halves until a specified number of divisions is reached. Then, it chooses some previously sampled nodes,  $q'$  to link with when adding each new node,  $q$  into N, creating new edges (E). If the gap between two nodes is wider than  $D_{max}$ , the present node,  $q$ , is unable to attach to its neighbour,  $q'$  (Mbemba, Chen and Shu, 2022).

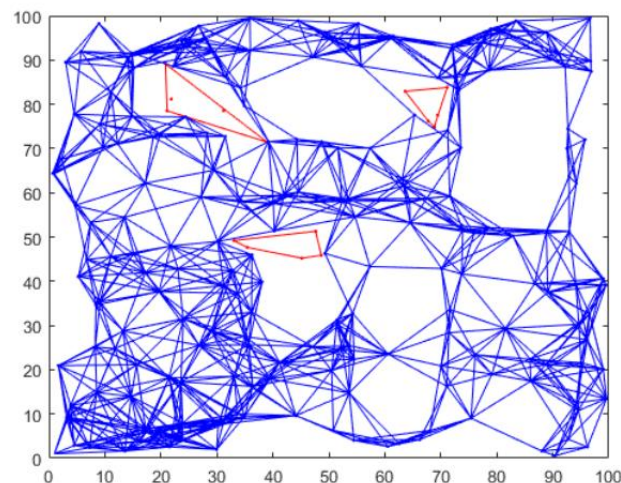


Figure 2.7: PRM Construction (Mbemba, Chen and Shu, 2022).

The ACO algorithm was created to mimic how ants search for heuristic outcomes to optimisation issues. Pheromones that are left behind by ants as they forage for food draw other ants to the food source. As a result, paths with more pheromones are regarded as more desirable since they are more likely to lead to a food source. However, the pheromones tend to fade away with time, making pathways with low pheromone levels less appealing. But the constant deposition process can outweigh evaporation. When determining its next move, each ant in the ACO algorithm takes into account the length of edges that can be reached from where it is now as well as the corresponding pheromone level. Every ant calculates a list of possible expansions to its present condition during each cycle. The prospect of an ant transferring from state  $x$  (the current node) to state  $y$  (neighbour node) is determined by the attractiveness of the move,  $\eta_{xy} = 1/d(x, y)$ , and the trail level  $\tau_{xy}$ , as stated in Equation 2.9. The selection probability only depends on the pheromone level when beta is set to zero, but when alpha is set to zero, it only depends on the heuristic distance. The trails are normally updated while all ants have finished their solution. The pheromone update for each trail is controlled by the global pheromone updating rule, denoted by Equation 2.10. Then, the level of trails changes to reflect the superiority of its movements in the solutions (Mbemba, Chen and Shu, 2022).

$$P_{xy}^k = \frac{(\tau_{xy}^k)^\alpha (\eta_{xy}^k)^\beta}{\sum_{j \in N_x^k} (\tau_{xj}^k)^\alpha (\eta_{xj}^k)^\beta} \quad (2.9)$$

where

$P_{xy}^k$  = probability of the  $k$ th ant to move from state  $x$  to state  $y$

$\tau_{xy}$  = quantity of pheromone deposited for the transition from state  $x$  to  $y$

$\alpha$  = control parameter for influence of  $\tau_{xy}$

$\eta_{xy}$  = the desirability of state transition  $x$  to  $y$

$\beta$  = control parameter for influence of  $\eta_{xy}$

$k$  = ant number

$N_x^k$  = a collection of adjacent nodes

$$\tau_{xy} \leftarrow (1 - \varphi)\tau_{xy} + \sum_{k=1}^K \Delta\tau_{xy}^k \quad (2.10)$$

where

$\tau_{xy}$  = quantity of pheromone deposited by K ants for a state transition x to y

$\varphi$  = pheromone evaporation coefficient

$$\Delta\tau_{xy}^k = \begin{cases} \frac{Q}{L_k}, & \text{If ant } k \text{ takes } x, y \text{ in its tour} \\ 0, & \text{otherwise} \end{cases}$$

#### 2.4.4 Artificial Fish Swarm Algorithm (AFSA)

Artificial Fish Swarm Algorithm (AFSA) provides a number of benefits, including reliability, enormous global search ability, wide parameter tolerance, and suited for path planning because of its insensitivity to initial values. In this study, the path planning environment is represented by a grid graph, with feasible zones represented by white squares and obstacles by black squares. Preying behaviour, swarming behaviour, following behaviour, and random behaviour are the behaviours that the AFSA simulates in order to accomplish optimization. The preying behaviour enables the artificial fish to identify a new location  $X_j$  that has the higher food concentration,  $Y_j$ . Then, random behaviour is carried out as Equation 2.11 illustrates, if no position with a greater food concentration than the present position can be identified. Being closer to the centre of the fish group and having a low degree of crowding within the fish group are both necessary conditions for swarm behaviour. Swarming behaviour is activated, if the degree of crowding is less than or equal to the crowding factor,  $\delta$ , of the fish swarm and the food concentration  $Y_c$  at the centre position is higher than the current position food concentration,  $Y_i$ , suggesting a less congested and superior centre position,  $X_c$ . If not, foraging behaviour is performed. The artificial fish can locate and explore a new spot that has a higher concentration of food than its previous location while engaging in the follow behaviour (Li, Du and Jia, 2022).

$$X_i(t + 1) = X_i(t) + step * \frac{X_j(t) - X_i(t)}{\|X_j(t) - X_i(t)\|} * rand() \quad (2.11)$$

where

$X_i(t + 1)$  = the updating function of position vector at t+1 iteration

$X_i(t)$  = the current position vector of artificial fish

The field of vision and step size range both have an effect on how quickly the AFSA algorithm converges. The fish swarm's clustering and following behaviour are highlighted by a broader field of view, which aids in identifying the global optimum. Contrarily, a narrower field of view promotes fish foraging and random activity, driving careful search that may result in local optimums. A dynamic feedback field of vision, symbolised by Equation 2.12, is developed to overcome these problems. This adaptive approach alters the field of vision based on how far away the target point is from the current location. As a result, the fish can quickly approach the target point since they initially have a wider range of view. The field of vision narrows as the algorithm advances, enabling a more thorough search to quickly arrive at the target spot. The step size range of artificial fish is also expanded to  $[1, 2\sqrt{2}]$  using the 16-Direction 24-Neighbourhood moving mode, as shown in Figure 2.8, to speed up reaching the target position, as opposed to the traditional AFSA using the 4-Direction 4-Neighbourhood moving mode, as shown in Figure 2.9, with a step size range of  $[1, \sqrt{2}]$ . The method can discover shorter pathways with fewer iterations due to the increase in the step size range, which accelerates convergence of algorithm. In order to control the step size according to the field of view, Equation 2.13, which represents an adaptable step size range, is also included. This modification eliminates step oscillation which was caused by excessive large range of vision that may otherwise slow the algorithm's rate of convergence. To put it simply, the AFSA algorithm starts by initialising the artificial fish's characteristics, for instance, field of vision, beginning point, target point, crowding factor and step size range. Information regarding obstacles is shared among the fish if the conditions of the sharing mechanism are satisfied. The algorithm then performs random

behaviour, preying behaviour, swarming behaviour, and following behaviour. The locations of the fish are regularly updated and recorded during this procedure. If the goal point is identified, the algorithm chooses the best option from all workable options, and the loop is closed. (Li, Du and Jia, 2022)

$$\begin{cases} visual = visual\ max * e^{-\lambda * \frac{t}{T}} \\ \lambda = m * \left(1 - \frac{dist}{DistMin}\right) \end{cases} \quad (2.12)$$

where

*visual* = dynamic feedback field of vision

*visual max* = maximum field of vision

*m* = adjustable parameter

*t* = current iteration number

*T* = maximum iteration number

*dist* = Euclidean distance from the target point in the fish swarm

*DistMin* = Euclidean distance between the starting point and the endpoint

$\lambda$  = feedback factor

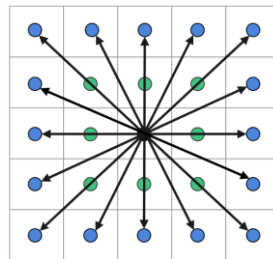


Figure 2.8: The Step Size Range of 16-Direction 24-Neighbourhood (Li, Du and Jia, 2022).



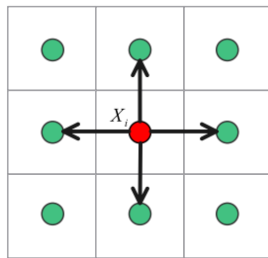


Figure 2.9: The Step Size Range of 4-Direction 4-Neighbourhood (Li, Du and Jia, 2022).

$$step = \alpha * step_{min} \quad (2.13)$$

where

$step$  = adaptable step size range

$step_{min}$  = minimum step size range

$$\alpha = \begin{cases} 2 & visual \geq step \ max \\ 1 & visual < step \ max \end{cases}$$

## 2.5 Path Planning via Smooth Trajectory Generation

In order to evaluate the concept of path smoothness, continuity is often utilised. Geometric ( $G^i$ ) and parametric ( $C^i$ ) continuity are the two main categories of continuity. Geometric continuity ensures that the ends of path segments will come together and have the same tangent vector directions. Contrarily, parametric continuity guarantees the convergence of endpoints as well as identical tangent vector magnitudes and directions. In essence, geometric continuity provides the smoothness of the robot's path navigation whereas parametric continuity indicates the smoothness of the curve itself and its parameterization. For example,  $C^1$  and  $C^2$  continuity denotes the tangent vector and acceleration vector continuity respectively.  $G^1$  continuity denotes the slope's continuity while  $G^2$  continuity denotes continuity of curvature (Ravankar et al., 2018).

Figure 2.10 shows a Bezier curve, an example of a parametric curve that uses control points to specify its shape. Equation 2.14 illustrates the use of Bernstein polynomial functions in these curves. Equation 2.15 may be used to represent the corresponding Bezier curve (Ravankar et al., 2018).

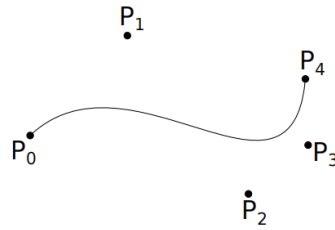


Figure 2.10: Bezier Curve (Ravankar et al., 2018).

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, 1, \dots, n, \quad (2.14)$$

where

$B_i^n(t)$  = Bernstein polynomial, whereby  $t$  is the positional parameter

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

$$C(t) = \sum_{i=0}^n B_i^n(t) P_i, \quad 0 \leq t \leq 1, \quad (2.15)$$

where

$C(t)$  = Path generated by Bezier curve,  $t$  is the positional parameter

$B_i^n(t)$  = Bernstein polynomial

$P_i$  = Control point of the Bezier curve

Next, Dubin's curve combines circular arcs with straight line segments to produce the shortest smoothed route between spots of a bounded curvature. The red segments in Figure 2.11 are straight components that have been blended with the green circular arcs (BC and DE), which illustrate how Dubin's curve is used to smoothen the path. (Ravankar et al., 2018)

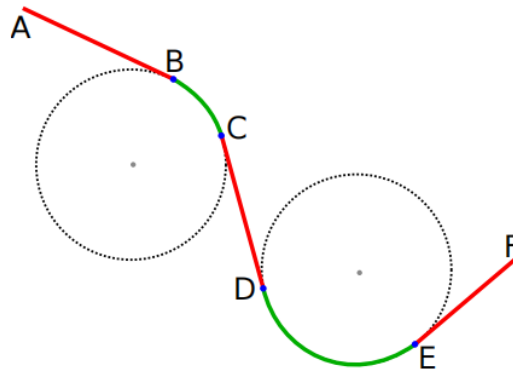


Figure 2.11: Path Smoothing using Dubin's Curve (Ravankar et al., 2018).

A non-uniform rational B-spline (NURBS) curve is defined following Equation 2.16. NURBS exhibit exceptional flexibility as a component for constructing desired trajectories by allowing the user to alter the control points and weights (Ravankar et al., 2018).

$$C(t) = \frac{\sum_{i=0}^n N_{i,p}(t)w_i P_i}{\sum_{i=0}^n N_{i,p}(t)w_i} \quad (2.16)$$

where

$C(t)$  = NURBS curve

$N_{i,p}$  = B-spline basis functions

$p$  = order

$P_i$  = control points

$w_i$  = weight of  $P_i$

The Bezier curve has the advantage of being computationally inexpensive, and by modifying the control points, curves with certain desired features can be formed. Curves with higher degrees can be managed, but as the degree of the curve rises, so does the computing complexity. As for the Dubin's curve, the shortest paths are ensured and have the fast computational speed for a given obstacles configuration. But, these curves lack of curvature continuity, and the robot encounters abrupt shifts when straight lines and circles converge. NURBS curves, on the other hand, have robust and fast computation, making them simple to use. They are also flexible in creating

the required trajectories. However, it occupies more memory storage and might experience problems if weights are initialised improperly, leading to poor parametrization. Hence, it can be seen that to smoothen robot pathways, Bezier curve is more suitable than the other two methods as it is simple and provides curvature continuity for paths generated (Ravankar et al., 2018).

In order to evaluate how the Bezier curve can be implemented to path planning algorithms to smoothen generated pathways, four research papers are studied which covers implementation of Bezier curve with GA, PSO, ACO and AFSA. Firstly, for multi-robot path planning using GA, the Bezier Curve Smoothing (BCA) algorithm has been suggested. To produce continuous smooth curves, this approach combines genetic algorithm and Bezier curves. Equations 2.17 and 2.18 build a Bezier curve from the control points along the path. The first derivative and second derivative of a Bezier curve are given by equations 2.19 and 2.20, and both may be calculated using control points to ensure a smooth transition along the path. Then, the curvature of Bezier curve on two-dimensional plane can be described using Equation 2.21 (Jianwei et al., 2020).

$$P(t) = \sum_{i=0}^m B_i^m(t)P_i, \quad 0 \leq t \leq 1, \quad (2.17)$$

where

$P(t)$  = Path generated by Bezier curve

$B_i^m(t)$  = Bernstein polynomial

$P_i$  =  $i$ th control point of the Bezier curve

$$B_i^m(t) = \binom{m}{i} t^i (1-t)^{m-i}, \quad i = 0, 1, \dots, m, \quad (2.18)$$

where

$B_i^m(t)$  = Bernstein polynomial

$$\dot{P}(t) = \frac{dP(t)}{dt} = m \sum_{i=0}^{m-1} B_i^{m-1}(t)(P_{i+1} - P_i) \quad (2.19)$$

where

$\dot{P}(t)$  = A Bezier curve's first derivative

$$\ddot{P}(t) = m(m-1) \sum_{i=0}^{m-2} B_i^{m-2}(t)(\dot{P}_{i+2} - 2\dot{P}_{i+1} + \dot{P}_i) \quad (2.20)$$

where

$\ddot{P}(t)$  = A Bezier curve's second derivative

$$k(t) = \frac{1}{R(t)} = \frac{\dot{P}_x(t)\ddot{P}_y(t) - \dot{P}_y(t)\ddot{P}_x(t)}{(\dot{P}_x^2(t) + \dot{P}_y^2(t))^{3/2}} \quad (2.21)$$

where

$k(t)$  = curvature of a Bezier curve

$R(t)$  = radius of curvature

$\dot{P}_x(t), \dot{P}_y(t)$  = X and Y coordinates components of the first derivative

$\ddot{P}_x(t), \ddot{P}_y(t)$  = X and Y coordinates components of the second derivative

Conventional path planning techniques typically lead to pathways with duplicated nodes and lots of abrupt inflection points, as seen in Figure 2.12. Here, S stands for the path's starting point, T for its ends, and P1, P2, P3, P4, and P5 for its inflection points. Using Equations 2.17 and 2.18, we can determine the equivalent Bezier curve as shown in Figure 2.13, by using P1, P2, P3, P4, and P5 as the control points. The chromosome in this algorithm indicates the Bezier curve's control point sequence. For simplicity and simple decoding, it is binary encoded before undergoing genetic manipulation (Jianwei et al., 2020).

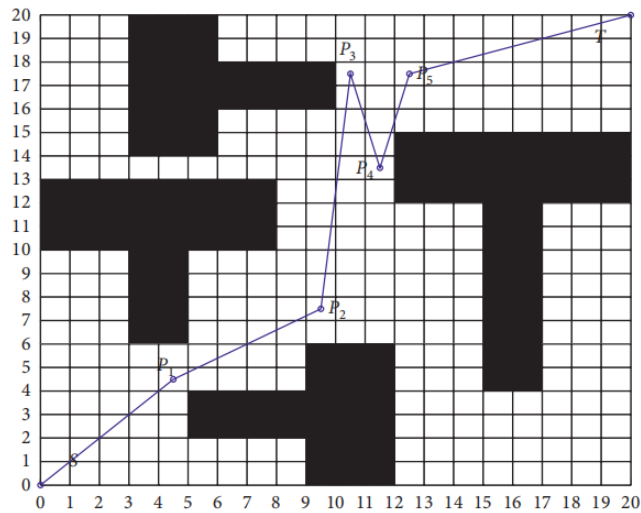


Figure 2.12: Pathway Generated by Traditional Path Planning Method  
(Jianwei et al., 2020).

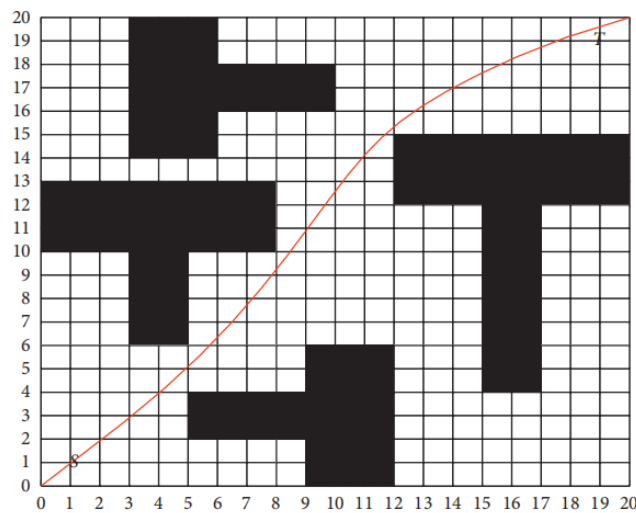


Figure 2.13: Pathway Generated by Bezier Curve Smoothing (BCA)  
Algorithm (Jianwei et al., 2020).

By using the Bezier curve, a smoother and more reliable planned path can be created. In this study, the author used the mutation, crossover, and selection operators to define the control points of the Bezier curve. As a result, energy lost during robot movement can be minimised. Roulette wheel method is applied to choose pathways. The probability of a path being selected can be illustrated by Equation 2.22. Path length will determine the fitness value. For instance, a lower path length will lead to a higher fitness value, resulting in a higher probability of the path being selected. A single-point crossover

approach was implemented to swap genes across two chromosomes in order to facilitate chromosomal crossover. Additionally, the mutation operator was given a probability of 0.1, allowing the mutation of portions of genes in the chromosome other than the start and end points. This strategy successfully inhibits premature convergence due to local optimum issue (Jianwei et al., 2020).

$$p_i = \frac{fit_i}{\sum_{i=1}^n fit_i} \quad (2.22)$$

where

$p_i$  = probability of the selected path

$fit_i$  = fitness value of the Bezier curve

$n$  = number of Bezier curves

Next, in order to create smooth pathways for mobile robots that adhere to continuous-curvature requirements, a revolutionary method was devised in 2021. The technique makes use of particle swarm optimization with adaptive delayed velocity (PSO-ADV) and a parametric cubic Bezier curve (PCBC). In conventional path smoothing, there are normally two steps in the procedure. The first step is to create a linear path using PSO while taking into constraints like path length and obstacle avoidance. A parametric curve is then used to smooth the linear path, creating discontinuous curvature at the joints to connect linear and curved segments. The study presents smooth paths made of PCBC segments to overcome this drawback by ensuring continuous curvature throughout the entire route. The improved PSO-ADV technique is also used to address problems like local trapping and premature convergence (Xu, Song and Cao, 2021).

An analytical trajectory-smoothing approach that assures continuous curvature is the parametric cubic Bezier curve (PCBC). It links straight linear segments smoothly while meeting the continuous-curvature criterion by adjusting the PCBC parameter. Equations 2.23 and 2.24 may be used to represent two cubic Bezier curves, as shown in Figure 2.14. The transition length,  $d$ , is an important factor in PCBC since it influences the overall quality

of Bezier transition curves and is directly related to the smooth trajectory generation criterion. The transition length,  $d$  is commonly established as  $d_1 = \|P_2 - B_{10}\|$  and  $d_2 = \|P_2 - B_{20}\|$  (as illustrated in Figure 2.14). Then, two constraints,  $d = d_1 = d_2$  and  $\beta = \theta/2$  are taken into account when generating the Bezier transition curves by determining their control points using Equations 2.25, 2.26, 2.27, and 2.28. The PCBC method has successfully produced continuous curvature at the intersection of  $B_1(u)$  and  $B_2(u)$ , at point  $B_{i3}$ . Equation 2.29 may be used to calculate the maximum curvature,  $k_{max}$  at point  $B_{i3}$ , which is inversely proportional to  $d$ . When a certain maximum curvature is supplied, it is crucial to ensure that the control parameter,  $d$ , satisfies the criteria stated in Equation 2.30 (Xu, Song and Cao, 2021).

$$B_1(u) = \sum_{i=0}^3 B_{1i} \binom{3}{i} u^i (1-u)^{3-i}, \quad 0 \leq u \leq 1, \quad (2.23)$$

where

$B_1(u)$  = the cubic Bezier curve

$B_{1i}$  = the  $i$ th control point

$$B_2(u) = \sum_{i=0}^3 B_{2(3-i)} \binom{3}{i} u^i (1-u)^{3-i}, \quad 0 \leq u \leq 1, \quad (2.24)$$

where

$B_2(u)$  = the cubic Bezier curve

$B_{2(3-i)}$  = the  $i$ th control point



$$B_{i0} = P_2 - T_i d, \quad i = 1 \text{ and } 2, \quad (2.25)$$

$$B_{i1} = P_2 - T_i(1 - c_1 c_3) d, \quad i = 1 \text{ and } 2, \quad (2.26)$$

$$B_{i2} = P_2 - T_i(1 - c_1 c_3 - c_3) d, \quad i = 1 \text{ and } 2, \quad (2.27)$$

$$B_{i3} = B_{i2} + \eta d u_d, \quad i = 1 \text{ and } 2, \quad (2.28)$$

where

$B_{i0,1,2,3}$  = the control point

$P_2$  = the straight line point

$$T_i = \frac{\overrightarrow{P_i P_{2,3}}}{\|\overrightarrow{P_i P_{2,3}}\|}$$

$d$  = the transition length

$$c_1 = 2(\sqrt{6} - 1)/5$$

$$c_2 = (c_1 + 4)(c_1 + 1)$$

$$c_3 = (c_1 + 4)/(c_2 + 6)$$

$$\eta = 6c_3 \cos \beta / (c_1 + 4)$$

$$u_d = \frac{\overrightarrow{B_{12} B_{22}}}{\|\overrightarrow{B_{12} B_{22}}\|}$$

$$k_{max} = \frac{2c_3 \sin \beta}{3d\eta^2} \quad (2.29)$$

where

$k_{max}$  = maximum curvature

$$d \geq \frac{2c_3 \sin \beta}{3k_{max}\eta^2} \quad (2.30)$$

where

$d$  = transition length

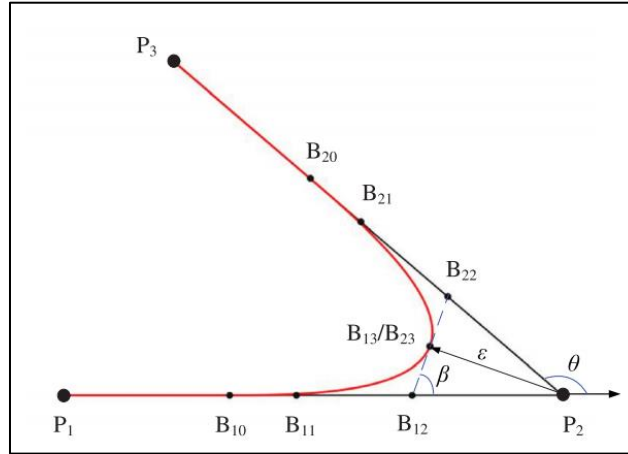


Figure 2.14: Bezier Transition Curves (Xu, Song and Cao, 2021).

The workspace is divided into  $2^n \times 2^n$  grids, each of which is given a grid number. The workspace gets increasingly detailed as the parameter  $n$  rises, albeit at the expense of greater computational expenses. The smooth trajectory generation of routes in this work focuses on identifying control points and control parameters in order to design a practical and ideal pathway. Obstacle avoidance, minimized maximum curvature, minimal path length, and continuity of curvature are the requirements for a practical and ideal smooth path. Each swarm particles in PSO-ADV represents a probable optimal solution to smooth path planning issue. Equations 2.31 and 2.32 are used to determine the position and velocity vectors of particles. Velocity is updated for the next iteration using the velocity from the previous and current iteration (Xu, Song and Cao, 2021).

$$\begin{aligned}
 V_i(k+1) &= wV_i(k) + w_1V_i(k-1) + c_1r_1(P_{ib}(k) - X_i(k)) \\
 &\quad + c_2r_2(G_b(k) - X_i(k))
 \end{aligned}
 \tag{2.31}$$

where

$V_i(k+1)$  = the updating function of velocity at  $k$ th iteration

$w$  = inertia weight of velocity

$V_i(k)$  = the current velocity of  $i$ th particle

$w_1$  = inertia weight of delayed velocity

$V_i(k-1)$  = the previous / delayed velocity of  $i$ th particle

$c_{1,2}$	= acceleration coefficients
$r_{1,2}$	= random numbers uniformly distributed on $[0, 1]$
$P_{ib}(k)$	= the particle's own best position till kth iteration
$X_i(k)$	= the current position of the ith particle
$G_b(k)$	= the global swarm's best position till kth iteration

$$X_i(k + 1) = X_i(k) + V_i(k + 1) \quad (2.32)$$

where

$X_i(k + 1)$	= the updating function of position vector at kth iteration
$X_i(k)$	= the current position vector of ith particle
$V_i(k + 1)$	= the updating function of velocity at kth iteration

These illustrations show the convex hulls of the Bezier curves and their related control points as blue solid lines and blue hollow circles, respectively. The optimal smooth pathways found are shown by the red solid curves. Figure 2.16 illustrates the path produced using the square Bezier curve (SBC) and MDPSO method, whereas Figure 2.15 shows the path produced using the parametric cubic Bezier curve (PCBC) and PSO-ADV algorithm. The smooth path in Figure 2.15 is made up of several cubic Bezier transition curve segments which is able to ensure the continuity of curvatures along the path. The smooth path in Figure 2.16, on the other hand, is made up of square Bezier curve segments that fail to keep their joints' curvature continuity. As a result, the mobile robot's acceleration and velocity are disrupted, necessitating frequent changes in motion modes. Additionally, this may result in over-actuation and slippage problems, especially during rapid motions (Xu, Song and Cao, 2021).

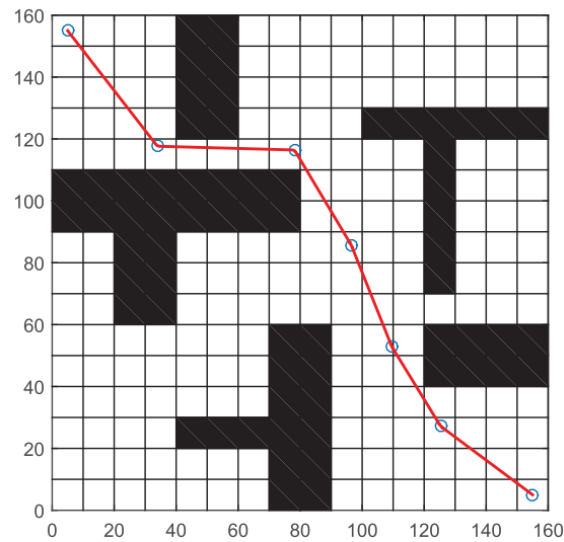


Figure 2.15: Smooth Path Produced by PCBC + PSO-ADV (Xu, Song and Cao, 2021).

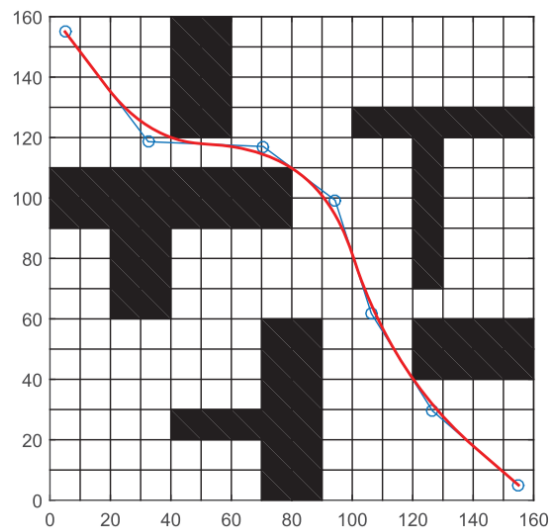


Figure 2.16: Smooth Path Produced by SBC + MDPSO (Xu, Song and Cao, 2021).

Thirdly, the integration of ACO with Bezier curve is discussed. The technique of path planning that is frequently in use produces pathways made up of segments of straight lines. These pathways are not ideal for many applications due to a number of concerns, including slippage, localization errors, mechanical wear, and discontinuity problems. The requirement that robots come to a complete stop at each turning edge to change their direction, and thereafter restart forward motion adds additional challenges. Prior

attempts integrate lines and circles together using techniques like Dubin, Reed, and Shepp to fulfill the demand for smoother trajectories with continuous curvature. These techniques generated smooth paths but also showed discontinuities. To create continuous curvature paths, it is necessary to use at least a third-order or cubic Bezier curve with a total of four control points, as shown in Equation 2.33. A starting point, an ending point, and two control points that determine the curve's shape make up the Bezier curve's four control points (Sitong and Tianyi, 2022). The path that the ACO algorithm found in Figure 2.17 is shown as a black line in simulation. The path's waypoints are shown by red dots, while the blue curve shows the Bezier curve. The advantages of using the Bezier curve are shown in Table 2.1, which shows a decrease in path length under various conditions including the quantity of obstacles, the threshold, the sample points, and the number of ants. The author emphasises that the Bezier curve outperforms the Ferguson curve when it comes to path planning concerns. This advantage results from the enhanced flexibility of the Bezier curve, which enables changes to the shape of the pathway generated by adjusting control points (Mbemba, Chen and Shu, 2022).

$$P = (1 - t)^3 P_1 + 3(1 - t)^2 t P_2 + 3(1 - t) t^2 P_3 + t^3 P_4 \quad (2.33)$$

where

$P$  = four-points Bezier curve

$P_i$  = control points that define the Bezier curve

$t$  = parameterization variable that ranges from 0 to 1 for point interpolation

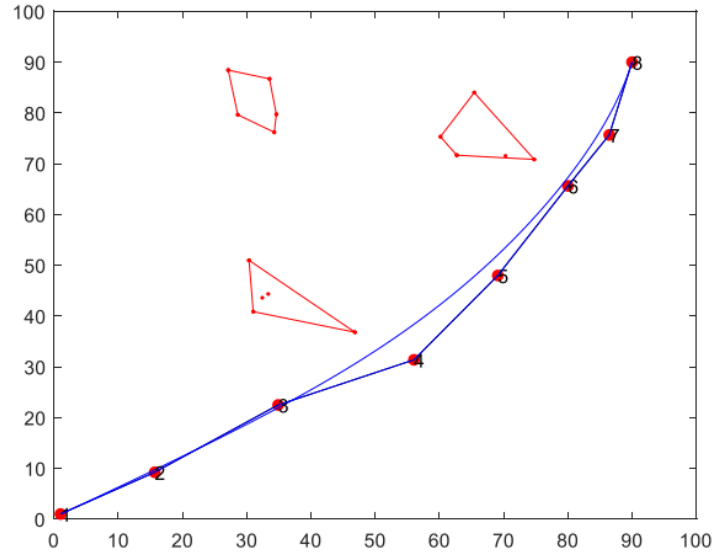


Figure 2.17: Path Smoothed by Bezier Curve (Mbemba, Chen and Shu, 2022).

Table 2.1: Path Length (m) under Different Parameters of Simulation Environments.

Environment No.	Obstacles	Threshold (m)	Sampling points	Ants number	Bezier Curve	Path length(m)
					-	129.203
1	3	25	300	1000	yes	128.974
					-	128.798
2	11	25	300	1000	yes	127.664
					-	139.151
3	11	15	20	100	yes	136.386

Lastly, the integration of AFSA with Bezier curve is discussed. Equations 2.34 and 2.35 illustrate the route smoothing procedure for the initial path generated by the AFSA method, which uses a third-order Bezier curve with four control points. A  $x$ -order Bezier curve is required to have  $x+1$  control points in order to be valid. Folded line segments with abrupt twists that make up the original path might result in curvature and path orientation discontinuities. Hence, the segments of the Bezier curve must adhere to certain requirements as shown by Equation 2.36 to guarantee continuity at

turning points. Two Bezier curves with eight control points in Figure 2.18 can be described using Equations 2.37 to 2.44. The arc created between two adjacent straight lines must not exceed the maximum rotation curvature,  $k_{max}$ , as shown by Equation 2.45, to lessen the robot's mechanical structural harm.. When transition length,  $d$  is not equal to 0 or  $\angle A_2B_2E_1, \alpha$  is not equal to  $90^\circ$ , the continuity of the third-order Bézier curve is guaranteed. The third-order Bezier curve's continuous requirement must then also be satisfied by carrying out collinear optimisation as the last step. After determining the "i" number of waypoints, the beginning point and the goal point are both remained. Then, the three points  $Bx - 1, Bx,$  and  $Bx + 1$  are tested for collinearity using the rank technique of a matrix, which is depicted by Equation 2.46. If all three of these points fall along the same straight line, it indicates that they are collinear. When the nxn matrix's determinant is non-zero, the matrix rank is "n". It denotes collinearity when  $n = 1$  or  $2$ , but a rank of  $3$  denotes non-collinearity. Given that the middle point  $Bx$  alone can define the line on which it lies,  $Bx - 1$  and  $Bx + 1$  may be safely eliminated in the event of collinearity since they become redundant. In a number of manners, the proposed AFSA approach performed better than the traditional fish swarm algorithm. It had the most effective alignment with the robot's kinematic features, had the shortest path length, and had the fewest inflection spots. Additionally, the algorithm's path-planning performance had a perfect 100% success rate (Li, Du and Jia, 2022).

$$P(s) = \sum_{i=0}^n B_i^n(s)P_i, \quad 0 \leq s \leq 1, \quad (2.34)$$

where

$P(s)$  = Path generated by Bezier curve,  $s$  is the positional parameter

$B_i^n(s)$  = Bernstein polynomial

$P_i$  = control point of the Bezier curve

$$B_i^n(s) = \binom{n}{i} s^i (1-s)^{n-i}, \quad i = 0, 1, \dots, n, \quad (2.35)$$

where

$B_i^n(s)$  = Bernstein polynomial, whereby  $s$  is the positional parameter

$n$  = order of the Bezier curve

$$F(\alpha) = c_1 d \sin \beta [\cos^2(\beta - \alpha) \sin \alpha - \cos^2 \alpha \sin(\beta - \alpha)] + 6d \cos^2(\beta - \alpha) \sin \alpha \left[ \cos^2 \alpha \sin \beta - \cos^2 \frac{\beta}{2} \sin 2\alpha \right] + 6d \cos^2 \alpha \sin(\beta - \alpha) \cos(\beta - \alpha) [\sin(\beta - \alpha) - \sin \alpha] \quad (2.36)$$

where

$F(\alpha)$  = Continuous third-order Bezier curve segments

$c_1$  = 7.2364

$d$  = transition length (distance between  $E_1B_0$  or  $E_1A_0$ )

$\beta$  =  $\angle E_0E_1E_2$

$\alpha$  =  $\angle A_2B_2E_1$

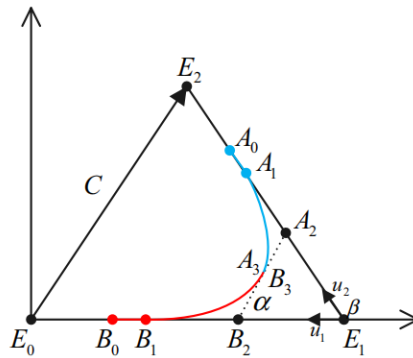


Figure 2.18: The Bezier Curve Model (Li, Du and Jia, 2022).



$$B_0 = E_1 - d_1 u_1 \quad (2.37)$$

$$B_1 = B_0 - p_b u_1 \quad (2.38)$$

$$B_2 = B_1 - q_b u_1 \quad (2.39)$$

$$B_3 = B_2 - f_b u_d \quad (2.40)$$

where

$B_{0,1,2,3}$  = the control points of first Bezier curve

$E_1$  = the inflection point of the path

$d_1$  = the transition length between  $E_1$  and  $B_0$

$u_1$  = unit vector of  $\overrightarrow{E_1 E_0}$

$p_b$  =  $c_2 c_3 d$

$c_2$  =  $\frac{2}{5}(\sqrt{6} - 1)$

$c_3$  =  $\frac{c_2 + 4}{c_1 + 6}$

$c_1$  = 7.2364

$q_b$  =  $c_3 d$

$f_b$  =  $\frac{6c_3 \cos \alpha}{c_2 + 4} d$

$u_d$  = unit vector of  $\overrightarrow{B_2 A_2}$

$$A_0 = E_1 - d_2 u_2 \quad (2.41)$$

$$A_1 = A_0 - p_a u_2 \quad (2.42)$$

$$A_2 = A_1 - q_a u_2 \quad (2.43)$$

$$A_3 = A_2 - f_a u_d \quad (2.44)$$

where

$A_{0,1,2,3}$  = the control points of the second Bezier curve

$E_1$  = the inflection point of the path

$d_2$  = the transition length between  $E_1$  and  $A_0$

$u_2$  = unit vector of  $\overrightarrow{E_1 E_2}$

$p_a$  =  $c_2 c_3 d$

$c_2$  =  $\frac{2}{5}(\sqrt{6} - 1)$

$c_3$  =  $\frac{c_2 + 4}{c_1 + 6}$

$$\begin{aligned}
c_1 &= 7.2364 \\
q_a &= c_3 d \\
f_a &= \frac{6c_3 \cos \alpha}{c_2 + 4} d \\
u_d &= \text{unit vector of } \overrightarrow{B_2 A_2}
\end{aligned}$$

$$k_{max} = \left( \frac{(c_2 + 4)^3}{54c_3} \right) \frac{\sin \alpha}{d \cos^2 \alpha} \quad (2.45)$$

where

$$\begin{aligned}
k_{max} &= \text{maximum curvature} \\
c_2 &= \frac{2}{5}(\sqrt{6} - 1) \\
c_3 &= \frac{c_2 + 4}{c_1 + 6} \\
d &= \text{transition length (distance between } E_1 B_0 \text{ or } E_1 A_0) \\
\alpha &= \angle A_2 B_2 E_1
\end{aligned}$$

$$a = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix} \quad (2.46)$$

where

$$\begin{aligned}
a &= \text{rank method of matrix} \\
x_n, y_n &= \text{coordinates of three control points, } Bx - 1, Bx, \text{ and } Bx + 1
\end{aligned}$$

## 2.6 Summary

Table 2.2 demonstrates the advantages and disadvantages of various path planning algorithms discussed in this chapter. Among these path planning algorithms, particle swarm optimization algorithm (PSO) under the bio-inspired algorithm is chosen for this project. This is because PSO can provide optimal result through tuning of PSO parameters such as inertia weight, population size, cognitive acceleration coefficient and social acceleration coefficient. The implementation and adjustment of PSO is easy and user-friendly as it does not involve complicated derivative computations. Besides, it presents faster convergence to optimal solution due to global and local

exploration by particles. Hence, PSO is the most suitable path planning algorithm for this project.

Table 2.2: Advantaged and Disadvantages of Different Path Planning Algorithms.

Artificial Neural Network (ANN)	<ul style="list-style-type: none"> <li>• Good generalization ability to adapt learnt knowledge and information to new, unknown scenarios.</li> <li>• Good capacity for learning.</li> <li>• Time consuming.</li> <li>• Convergence with the ideal answer may not be guaranteed by the learning method.</li> </ul>	
Bio-inspired Algorithm	Genetic Algorithm (GA)	<ul style="list-style-type: none"> <li>• Can be applied for large and complicated maps.</li> <li>• High parallelism for multi-robot path planning.</li> <li>• High computational complexity due to its genetic operators.</li> <li>• Premature convergence of solutions.</li> </ul>
	Particle Swarm Optimization (PSO)	<ul style="list-style-type: none"> <li>• Easy implementation because it does not apply complicated derivative computations.</li> <li>• Exploration on a global and local scale enables it to quickly identify the optimal solution.</li> <li>• Easily caught in the local optimum.</li> <li>• Convergence speed progresses slowly when there are several local optima.</li> </ul>
	Ant Colony Optimization (ACO)	<ul style="list-style-type: none"> <li>• Faster speed of convergence.</li> <li>• High variation in solution.</li> <li>• Time-consuming due to the need to tune various parameters.</li> <li>• Heavy computational cost due to</li> </ul>

	probability calculation for path selection.
Artificial Fish Swarm Algorithm (AFSA)	<ul style="list-style-type: none"><li>• Robust global searchability.</li><li>• Large parameter setting tolerance.</li><li>• Time-consuming due to the need to tune various parameters.</li><li>• High computational load for multi-robot path planning.</li></ul>

## CHAPTER 3

### METHODOLOGY AND WORK PLAN

#### 3.1 Introduction

This chapter focuses on elaborating and discussing the methodology for optimised multi-robot path planning via smooth trajectory generation. For improvement of existing modified particle swarm optimization algorithm (MPSO), an enhanced particle swarm optimization algorithm (EPSO) is implemented in this project. The optimization feature focuses on introducing trajectory smoothing by using Bezier curve for shorter path length, shorter execution time and lower number of turns in pathways generated for robots.

#### 3.2 Modified Particle Swarm Optimization (MPSO)

Kennedy and Eberhart (1995) first presented the classical PSO, an evolutionary optimization technique based on stochastic populations that draws inspiration from biological swarm intelligence. The classical PSO method treats each member of the population as a particle, and this collection of particles is known as a swarm (P.K Das et al., 2016). In MPSO, each particle's initial positions and velocities inside the search space are assigned at random after the population size is first determined. After that, each particle's fitness value is determined and contrasted with its prior surpassing record position. If the fitness value surpasses the previous one, the particle's current position will shift to become its new individual best position. The current position will be modified to take on the new global best position if it has a better fitness value than the previous global best position. The particle's new position and velocity are updated using the Equations 3.1 and 3.2 below. Once the specified number of iterations has been reached, the PSO terminates (Poy, Darmaraju and Kwan, 2023).

$$v_i = \omega v_i + c_1 r_1 (p_i - x_i) + c_2 r_2 (p_g - x_i) \quad (3.1)$$

where

- $v_i$  = current velocity of the robot
- $\omega v_i$  = inertial weight set to the velocity of the particle before update
- $c_1$  = personal acceleration coefficient
- $p_i$  = the particle's own best fitness
- $x_i$  = current position of the robot
- $c_2$  = global acceleration coefficient
- $p_g$  = the particle's global best fitness

$$x_i = x_i + v_i \quad (3.2)$$

where

- $x_i$  = current position of the robot
- $v_i$  = current velocity of the robot

Next, a method for fine-tuning the three parameters (known as  $\omega$ ,  $c_1$ , and  $c_2$ ) inside the particle's velocity equation as shown in Equation 3.1 has been developed in a study by Poy, Darmaraju and Kwan (2023). To establish an equilibrium between the particle's local exploration and global exploitation properties, the inertial weight,  $\omega$ , is crucial. It has a significant impact on the PSO algorithm's convergence. With the aid of a well-balanced local exploration ability, a robot may quickly and accurately determine the optimum path since a higher amount of  $\omega$  supports a complete and global search while a lower value  $\omega$  allows for fine and local search (Poy, Darmaraju and Kwan, 2023).

In general,  $\omega$  is set to either 1 or a number between 0 and 1. When it is set to 1, the particle's movement is fully determined by the direction of its previous motion, thus prompting it to continue moving in the same direction. In order to allow the particle to navigate through more of the search area,  $\omega$  must be set to a value between 0 and 1. This reduces the impact of prior motion. The algorithm's searching ability is improved as a result (Gadghadi,

Hachaichi and Zairi, 2022). Equation 3.3, in which a linearly decreasing  $\omega$  technique is employed, is designated as the equation for calculating the value of  $\omega$ . The value of  $\omega$  starts at 0.95, gradually reduces as more iterations are carried out, and finally ends at 0.4 for this algorithm (Poy, Darmaraju and Kwan, 2023).

$$\omega = \omega_{start} - \frac{\omega_{start} - \omega_{end}}{K} k \quad (3.3)$$

where

- $\omega_{start}$  = start value of inertial weight
- $\omega_{end}$  = end value of inertial weight
- $K$  = maximum number of iterations
- $k$  = current iteration

Then, the algorithm's performance is improved by implementing the tuning of the personal acceleration coefficient,  $c_1$  for the cognitive component and the global acceleration coefficient,  $c_2$  for the social component. The  $c_1 r_1 (p_i - x_i)$  in the Equation 3.1 stands for the cognitive component, whereas  $c_2 r_2 (p_g - x_i)$  refers to the social component. Higher  $c_1$  values draw particles towards their individual best positions, while higher  $c_2$  values drive them towards the global best positions. This is because the algorithm's ability for local exploration is determined by  $c_1$ , while its ability for global exploration is determined by  $c_2$ . The adjustment of  $c_1$  and  $c_2$  values is carried out using the corresponding Equations 3.4 and 3.5, by referring to Poy, Darmaraju and Kwan (2023). The superiority of the results that the algorithm creates gradually improves as the cognitive component gradually decreases and the social component gradually increases. The above strategy works because there is typically a desire to move towards convergence as the iterations progress. Increasing  $c_2$  can encourage convergence to optimal solution due to the ability of  $c_2$  value to promote the drive to global best position. However, in contrast to the Enhanced Particle Swarm Optimisation method (EPSO) covered in Section 3.3, the MPSO paths developed do not take trajectory smoothness into consideration.

$$c_1 = c_{1i} - \left( \frac{c_{1i} - c_{1f}}{K} \right) k \quad (3.4)$$

$$c_2 = c_{2i} - \left( \frac{c_{2i} - c_{2f}}{K} \right) k \quad (3.5)$$

where

$c_{1i}$  = initial value of personal acceleration coefficient

$c_{1f}$  = final value of personal acceleration coefficient

$c_{2i}$  = initial value of global acceleration coefficient

$c_{2f}$  = final value of global acceleration coefficient

### 3.3 Proposed Enhanced Particle Swarm Optimization (EPSO)

#### 3.3.1 Assumptions

- The initial positions, current positions, and goal positions for all robots are known in the 2D simulated environment.
- Robots will be operated in a static environment whereby there will only exist static obstacles.
- The complete paths from the initial position to goal position for all robots are calculated before navigation.
- Differential drive robots are used and all robots are homogeneous.
- The Bezier curve trajectory smoothing mechanism is incorporated separately from the PSO path planning algorithm.

#### 3.3.2 Path Planning Scheme

The Enhanced Particle Swarm Optimization (EPSO) path planning strategy uses the Modified Particle Swarm Optimisation (MPSO) algorithm, which was first introduced by Poy, Darmaraju, and Kwan in 2023. This methodology focuses on identifying individual waypoints rather than creating the entire route with several waypoints in one algorithm run. As a result, the EPSO algorithm will be executed five times, each time with a few iterations to determine a single waypoint, for a robot's whole route comprising five waypoints. It uses a global path planning technique, planning all of the robots'



pathways completely before they begin travelling through a known simulation environment (Poy, Darmaraju and Kwan, 2023).

Furthermore, the swarm of particles is not initiated at the initial robot location, but within a predefined search space. To solve the problem of path planning, a fitness function has been created. This fitness function's fundamental goal is to produce the shortest possible path for all robots. The pursuit of the global optimum position becomes crucial to achieve the shortest possible path. The spot in the local search zone where the particle swarm is able to compute the shortest path to the target is known as the global optimum position. The main fitness function, designated as  $F_1$  in Equation 3.6, is used to complete this task. By computing the Euclidean distance between the robot's consecutive waypoint and the goal position, this function selects the subsequent waypoint that is closest to the robots' goal position. The weight factor,  $\lambda_1$  of the fitness function is incorporated into the overall objective function, as seen in Equation 3.7. Due to the need to minimise the Euclidean distance between the robot's present position and the goal position, decreasing the fitness value in this scenario can produce an optimal path (Poy, Darmaraju and Kwan, 2023).

$$F_1 = \sum_{j=1}^{nr} \sqrt{(x_j^{next} - x_j^{goal})^2 + (y_j^{next} - y_j^{goal})^2} \quad (3.6)$$

where

$nr$  = number of robots

$x_j^{next}$  = x-coordinate of next waypoint for  $j^{th}$  robot  
(with minimum Euclidean distance)

$y_j^{next}$  = y-coordinate of next waypoint for  $j^{th}$  robot  
(with minimum Euclidean distance)

$x_j^{goal}$  = x-coordinate of target position for  $j^{th}$  robot

$y_j^{goal}$  = y-coordinate of target position for  $j^{th}$  robot

$$F = \lambda_1 F_1 \quad (3.7)$$

where

$\lambda_1$  = weight factor of  $F_1$

Moreover, the obstacle avoidance algorithm is implemented in the EPSO by referencing the MPSO algorithm, which was presented by Poy, Darmaraju, and Kwan in 2023. Three robots will move from their starting positions to their designated target positions during the simulation, and it is possible for their paths to cross. Hence, each robot is equipped with LIDAR sensors, which are used by the obstacle avoidance algorithm to identify obstructions that are within sensor range. Each robot has three sensors connected to it, each with a 60-degree field of view. This allows the robot to identify static and dynamic impediments for a total of 180 degrees in the direction it is heading. This allows the robot to avoid any potential obstructions in its path. When the sensor detects an obstruction, it sends a logic value of 1 to the robot control system, causing the robot to rotate its heading in another direction in search of a free area to resume its navigation (Poy, Darmaraju and Kwan, 2023).

### 3.3.3 Trajectory Smoothing with Bezier Curve

The EPSO algorithm includes smooth trajectory generation after the path planning is completed. The method integrates the use of differential drive two-wheeled mobile robots (TMRs) for smooth trajectory generation by referencing the work of Yang, Lee and Ryuh from 2013. The differential drive, which involves the use of two drive wheels positioned along a common axis, is the driving mechanism used by these TMRs. Motion can be made forward or backward by separately operating each wheel. The TMR is illustrated in Figure 3.2 within a coordinate system that makes use of the robot's central position and directional angle. This positioning takes into account both the robot frame and the world frame coordinate systems. The position of the TMR, abbreviated as  $P_c$ , is described mathematically by Equation 3.8. Equation 3.9 can be used to illustrate the TMR's kinematic model (Yang, Lee and Ryuh, 2013).

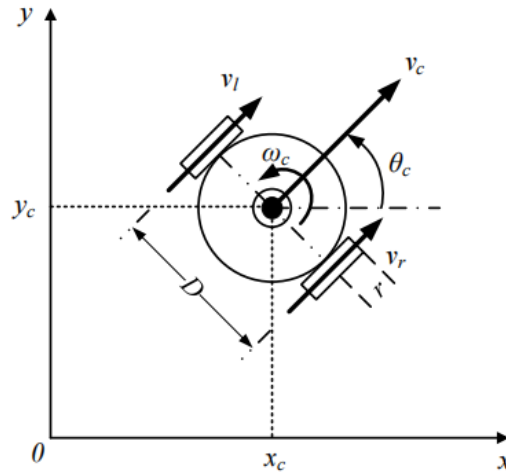


Figure 3.1: Kinematics of a TMR (Yang, Lee and Ryuh, 2013).

$$P_c = [x_c, y_c, \theta_c]^T \quad (3.8)$$

where

$x_c$  = x-coordinate of robot's position

$y_c$  = y-coordinate of robot's position

$\theta_c$  = direction angle of robot

$$P_c = \begin{bmatrix} \frac{r}{2} \cos \theta_c & \frac{r}{2} \cos \theta_c \\ \frac{r}{2} \sin \theta_c & \frac{r}{2} \sin \theta_c \\ \frac{r}{D} & -\frac{r}{D} \end{bmatrix} \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix} \quad (3.9)$$

where

$r$  = radius of a robot's wheels

$D$  = distance between its two wheels

$\omega_r$  = right wheel's angular velocity

$\omega_l$  = left wheel's angular velocity

Robots have been designed with trajectories that have the characteristic of changing direction after stopping which causes discontinuity of movement. This can cause slippage or veering off the desired path of robots. In this paper, the technique integrates PSO path planning algorithm and Bezier

curve trajectory smoothing algorithm to steer the robot in order to eliminate this concern. Bezier curves are frequently used to create curved trajectories. Figure 3.3 shows how a cubic Bezier curve is used to create a trajectory using a start point  $P_i (A_0, B_0)$ , an end point  $P_f (A_3, B_3)$ , and control points  $C_1 (A_1, B_1)$  and  $C_2 (A_2, B_2)$  (Yang, Lee and Ryuh, 2013).

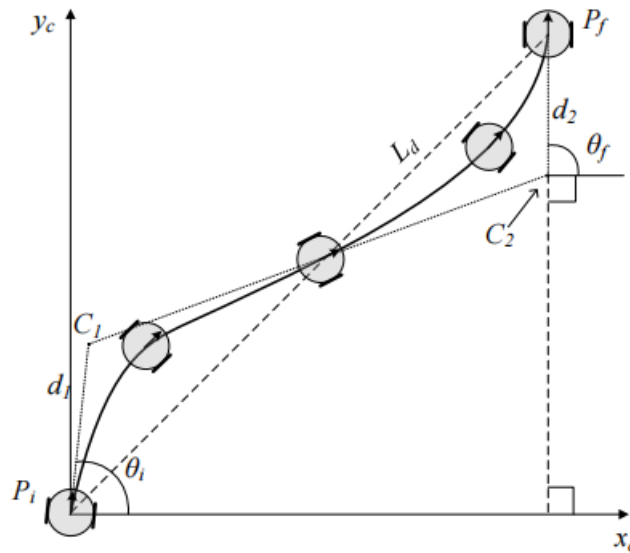


Figure 3.2: Bezier Curve-based Path Planning (Yang, Lee and Ryuh, 2013).

A cubic Bezier curve has a polynomial of third degree with four control points ( $P_i, C_1, C_2, P_f$ ), as shown in Figure 3.3. The path is divided into three unique segments that connect these control points along this curve, which has two directional transitions. In simple terms,  $C_1$  and  $C_2$ 's duties include exerting gravitational pull on the pathway, enabling the construction of a smooth S-shaped curve, as illustrated in Figure 3.3. The smooth Bezier curve generated by taking into account the control points and Bernstein polynomials is represented by Equation 3.10. A more accurate Bezier curve can be created by decreasing each successive increment of  $u$  from 0 to 1. For instance,  $u$  can be set to have a step of 0.001 in between 0 to 1 then the number of control points of Bezier curve will be 100. The Bernstein polynomial is computed using Equation 3.11, which is particularly designed for the cubic Bezier curve. This cubic Bezier curve is then broken down into its x-component and y-component, which are described in Equations 3.12 and 3.13 respectively (Yang, Lee and Ryuh, 2013).

$$B(u) = \sum_{i=0}^3 B_i \binom{3}{i} u^i (1-u)^{3-i}, \quad 0 \leq u \leq 1, \quad (3.10)$$

where

- $B(u)$  = the cubic Bezier curve  
 $B_i$  = the  $i$ th control point present on the curve  
 $\binom{3}{i} u^i (1-u)^{3-i}$  = Bernstein polynomial

$$B_{i=0}^3(u) = \binom{3}{i} u^i (1-u)^{3-i}, \quad i = 0, 1, 2, 3 \quad (3.11)$$

where

- $B_{i=0}^3(u)$  = Bernstein polynomial for cubic Bezier curve  
 $\binom{3}{i}$  =  $\frac{3!}{i!(3-i)!}$ , the binomial coefficient value

$$\begin{aligned} x(u) &= \sum_{i=0}^3 A_i B_{i=0}^3(u) \\ &= A_0(1-u)^3 + 3A_1u(1-u)^2 + 3A_2u^2(1-u) + A_3u^3 \end{aligned} \quad (3.12)$$

where

- $A_i$  = x-coordinates of point  $P_i$ ,  $C_1$ ,  $C_2$ , and  $P_f$   
 $B_{i=0}^3(u)$  = Bernstein polynomial for cubic Bezier curve

$$\begin{aligned} y(u) &= \sum_{i=0}^3 B_i B_{i=0}^3(u) \\ &= B_0(1-u)^3 + 3B_1u(1-u)^2 + 3B_2u^2(1-u) + B_3u^3 \end{aligned} \quad (3.13)$$

where

- $B_i$  = y-coordinates of point  $P_i$ ,  $C_1$ ,  $C_2$ , and  $P_f$   
 $B_{i=0}^3(u)$  = Bernstein polynomial for cubic Bezier curve

In this paper, the waypoints of the robot pathway established by the PSO algorithm are extracted to calculate the coordinates of the Bezier curve smooth trajectory. Before that, the Bernstein basis function is calculated for each robot. Then, the coordinates of Bezier curve control points for smooth trajectory generation are calculated by summing the element-wise product of the robot's waypoints and Bernstein Basis function. Finally, the Bezier curve points are plotted on the figure for each robot, and in the simulation of the robot pathway, the robot will travel from the start position to the target position by taking a smooth trajectory generated by the Bezier curve algorithm.

### 3.3.4 Flowchart of EPSO Path Planning Algorithm

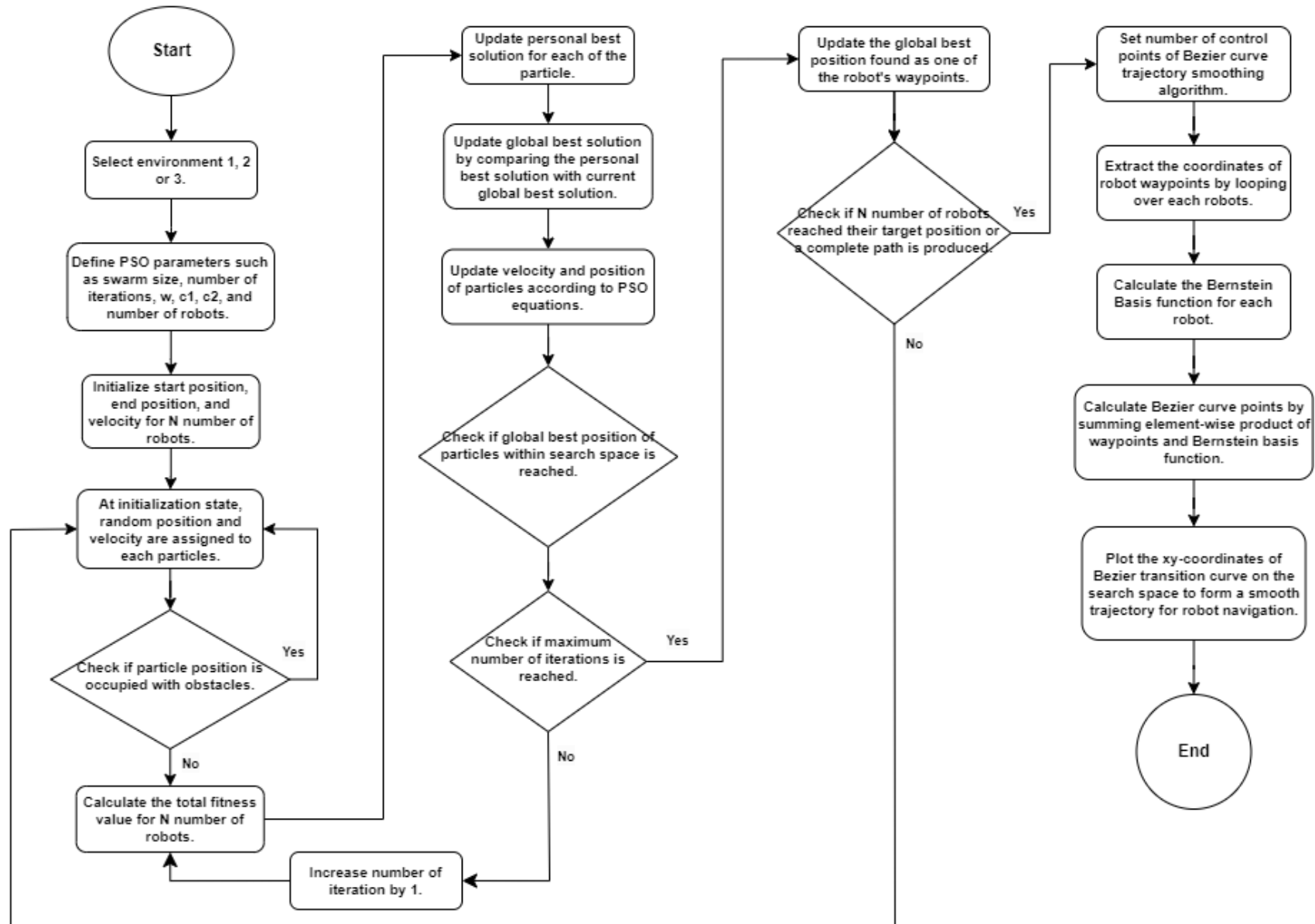


Figure 3.3: Flowchart of EPSO Path Planning Algorithm.

### 3.4 Planning and Managing of Project Activity

This section explains the detailed scheduling process that was used to finish this one-year project. It is represented by two Gantt charts, which are illustrated in Tables 3.1 and 3.2. To maximise the likelihood of a successful completion, the project was divided into two distinct sections, each lasting one semester. In a total of 28 weeks, Parts I and II of the project were finished from beginning to end. This project had no costs because it solely used software resources like Matlab.

The Gantt chart for FYP I is shown in Table 3.1. It lasts for a total of 14 weeks. Problem-formulating and project planning are done from week 1 to week 2. The broad outcomes that should be fulfilled at the end of the project are defined. The particular procedures and outputs needed to attain the outcomes are then specified. For the purpose of achieving the objectives, a 14-week work schedule outlining the tasks to be completed is established. Relevant research articles and publications have begun to be collected and read. From week 2 to week 8, literature review writing for different path planning algorithms for multi-robot systems is carried out. There are a total of 5 path planning algorithms written for this chapter, which include Artificial Neural Network (ANN), Particle Swarm Optimization (PSO) Algorithm, Genetic Algorithm (GA), Artificial Fish Swarm Algorithm (AFSA), and Ant Colony Optimization (ACO) Algorithm,. The different trajectory smoothing techniques are also explored and evaluated their advantages and disadvantages, which include non-uniform rational B-spline (NURBS) curve, Bezier curve, and Dubin's curve. Then, from week 9 to week 13, methodology research writing is carried out. It is based on the Enhanced Particle Swarm Optimization (EPSO) algorithm, which combines Bezier curve trajectory smoothing with the Particle Swarm Optimization method. Furthermore, from week 11 to week 14, report writing and presentation is carried out. The progress report writing and formatting is performed. The presentation slides are prepared for a face-to-face presentation with the FYP moderator and supervisor.

Table 3.2 shows the Gantt chart for FYP 2. From week 1 to week 6, the EPSO algorithm will be developed and simulated in Matlab. From week 5 to week 10, result and discussion writing of FYP report will be carried out to



evaluate whether the objectives set during FYP 1 are completed. From week 9 to week 10, poster for the final year project will be prepared. Moreover, from week 9 to week 14, final report for final year project will be prepared and finalised for submission.

3.4.1 Gantt Chart of FYP 1

Table 3.1: Gantt Chart of FYP 1.

No.	Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Problem formulation and project planning	█	█												
2	Literature review		█	█	█	█	█	█	█						
3	Methodology Research									█	█	█	█	█	
4	Report writing and presentation											█	█	█	█



### 3.4.3 Summary

The proposed Enhanced Particle Swarm Optimisation (EPSO) method for optimised multi-robot path planning via smooth trajectory generation is clearly explained in this chapter. To increase effectiveness, shorten execution times, and encourage the conservation of energy resources, this novel method integrates the Modified Particle Swarm Optimisation (MPSO) algorithm with Bezier curve trajectory smoothing technique.

Unlike the traditional PSO method, the EPSO algorithm starts by developing equations to compute global acceleration coefficient, inertia weight, and personal acceleration coefficient. The modifications are intended to gradually improve the quality of the algorithm's solutions over its iterations. The path planning technique adopted here prioritizes the identification of individual waypoints, with each waypoint decided by a single run of the PSO algorithm. It uses a global path planning approach to carefully plan each robot's route before it sets off on its journey within a known simulated environment.

A fitness function is also developed to produce the shortest path for each robot. To reach an ideal outcome, the fitness value is decreased just as the iterations increases. After all robot pathways have been established, Bezier curve trajectory smoothing algorithm is carried out. The waypoints the PSO algorithm selects for the robot pathway in this case define the outline of the Bezier curve. Once convergence has been reached or stopping criteria have been met, the PSO algorithm determines the waypoints from the global best particle. Then, the coordinates of Bezier curve control points for smooth trajectory generation are calculated by summing the element-wise product of the robot's waypoints and the Bernstein Basis function. Finally, the Bezier curve points are plotted on figure for each robot and in simulation of robot pathway, the robot will travel from start position to target position by taking smooth trajectory generated by Bezier curve algorithm.

## CHAPTER 4

### RESULTS AND DISCUSSION

#### 4.1 Introduction

The focus of this chapter is to present the result of an Enhanced Particle Swarm Optimization algorithm (EPSO) in three different warehouse environments. Section 4.2 presents the tuning of three EPSO parameters including swarm size, number of control points of the Bezier curve, and inertia weight, cognitive, and social acceleration coefficients. Then, section 4.3 presents the result of EPSO and Modified Particle Swarm Optimization algorithm (MPSO) based on path length, execution time, and number of turn points of three robots. Results of EPSO and MPSO are compared and evaluated through simulation.

#### 4.2 EPSO Parameters

##### 4.2.1 Swarm size

Swarm size, which is also called population size or number of particles, is one of the control parameters in PSO that need to be tuned by the user. A larger swarm size in EPSO multi-robot route planning suggests the algorithm will carry out a wider search. The number of times the fitness function is evaluated rises with each iteration as the population grows. To determine each particle's fitness value, the fitness function is given parameters about the particle, including its current location and its target position. Next, the robot's local best fitness will be compared with the fitness value of each particle. A particle's location will be kept as the best solution discovered until now inside its neighbourhood if its fitness value exceeds the local best fitness. Other particles will keep comparing their fitness values with the new local best fitness value, and in the meantime, the particle's fitness value will become the new local best fitness for its neighbourhood. Additionally, the local best position will be modified to become the new global best position for other particles to comply with if it is discovered that the local best fitness value exceeds the global best fitness value. As a result, anytime a better solution is discovered, the particles

can converge towards the goal location by continually updating the global best position.

As EPSO is directed by the global best solution, growing the population size could be advantageous. This is because a bigger population size suggests that more particles may be used to investigate the surroundings, thereby increasing the possibility that the PSO algorithm's performance will increase. Furthermore, a larger population will decrease the likelihood of particles becoming trapped in a local minimum as larger swarms show a greater ability to withstand noise by weakening the effects of individual particles' noise. However, because the fitness function is assessed for each particle more frequently with a larger population, the execution time will also become longer. Conversely, a smaller population number suggests that fewer particles may be employed to scout the surroundings and choose the best route. The likelihood of an early convergence to the local minimum solution will therefore be increased. However, because there will be fewer PSO equation calculations made during each iteration, the execution time will be shorter for smaller swarm sizes.

Hence, there must be careful calibration of population size to balance between the algorithm's convergence speed and optimality of solution to ensure EPSO can achieve its best performance. Table 4.1 below shows the simulation results for different swarm sizes of 20, 50, 80, 100, and 150. For each swarm size, the average path length and average execution time for three robots are evaluated respectively. The simulation is repeated 3 times to obtain the average path length and average execution time for different swarm sizes.

Table 4.1: Simulation Results for Different Swarm Size.

Swarm Size	Average Path Length (m)			Average Execution Time (s)		
	Robot 1	Robot 2	Robot 3	Robot 1	Robot 2	Robot 3
20	104.97	103.38	79.25	51.07	50.28	40.57
50	104.39	104.12	79.16	77.58	78.64	59.36
80	104.78	104.12	79.10	91.55	93.41	73.12
100	104.75	104.73	79.25	112.03	113.02	93.38
150	104.26	103.67	79.26	155.51	158.53	124.02

The result from Table 4.1 indicates that the average path length for all three robots will only have slight differences for different swarm sizes. This is due to the implementation of the Bezier curve trajectory smoothing algorithm. The original pathways of robots are now controlled and modified by the Bezier curve equation which can provide adequate trajectory smoothing effect for different swarm sizes. For average execution time, it can be observed that as swarm size increases, the average execution time for all three robots also increases. For instance, Robot 1 only requires 51.07 seconds when the swarm size is 20 but the execution time increases to 155.51 seconds when the swarm size becomes 150. Among these swarm sizes, 80 are chosen for further simulation. This is because, at a swarm size of 80, it can be observed that Robot 3 marks the shortest path length achieved, which is 79.10m. Further increase of swarm size beyond 80 will lead to an increase in path length for Robot 3. Consequently, it may be concluded that an excessively high swarm size may increase the computing resources required rather than necessarily improving the quality of the solution.

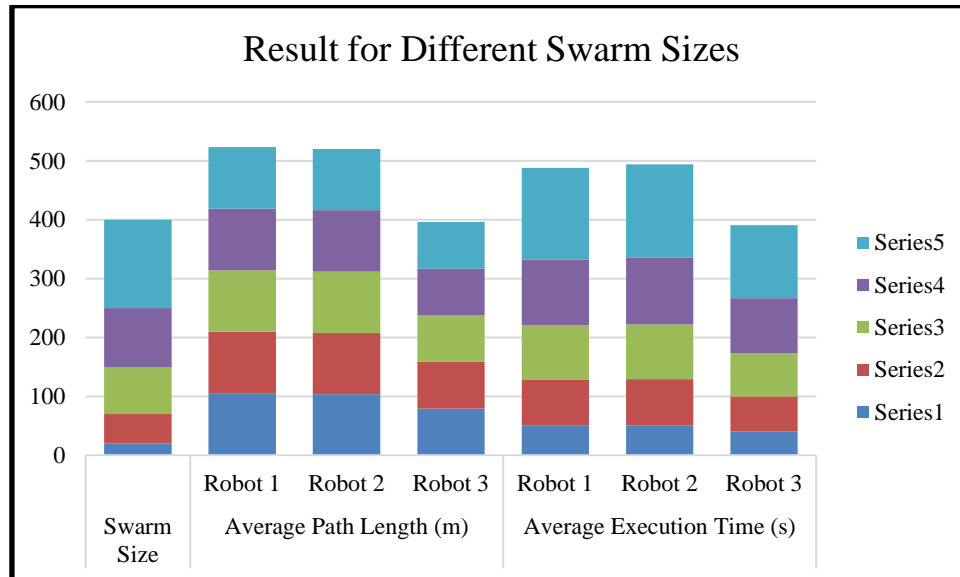


Figure 4.1: Chart of Result for Different Swarm Sizes.

From Figure 4.1, it can also be observed that the execution time of all robots for the swarm size of 150 is the highest among swarm sizes of 20, 50, 80, and 100. This illustrates that an increment in swarm size will increase the execution time greatly. For path length, the increment in swarm size will not

bring about large changes due to the implementation of the Bezier curve trajectory smoothing algorithm.

#### **4.2.2 Number of Control Points of Bezier Curve**

In EPSO multi-robot path planning, the trajectory smoothing process via the Bezier curve comes after the path planning process is completed. This is because trajectory smoothing requires the robots' waypoints generated through path planning. For each robot, the waypoints refer to the coordinates in which the robot will travel from the start position to the target position. A parametric curve defined by a collection of control points is called a Bezier curve. The waypoints that the robots created throughout the path planning process will be used to generate smooth trajectory through the Bezier curve equation. The robots' waypoints are then interpolated to provide more intermediate control points between them, which are subsequently used to build a Bezier curve path. The Bernstein basis functions are used in the Bezier curve computation process. These functions take the number of control points and waypoints as input and output the coordinates of a smooth path. It is possible to adjust the number of intermediate control points to create a pathway with fewer turn points and greater smoothness. This is because number of control points will affect the step size,  $t$  of the Bezier curve equation. A larger number of control points will produce a smaller step size. For instance, with 100 control points, the step size becomes 0.01, and with 200 control points, the step size becomes 0.005. A smaller step size produces a more precise depiction of the curve, resulting in smoother and more detailed routes.

There is a direct relationship between the number of control points of the Bezier curve and number of turn points generated. The number of turn points of robots represents the changes in the direction of the path traveled by robots when the direction change is more than 5 degrees for the robot to travel from its current waypoint to the next waypoint. Increasing the number of control points of the Bezier curve will result in the reduction of the number of turn points along the robots' pathway. This indicates an increase in trajectory smoothness as robots do not need to make abrupt direction changes to deal with each turning point and can travel according to a smooth trajectory. On the other hand, decreasing the number of control points will result in an increment



of the number of turn points along the pathway. This is because fewer control points along the path will result in an underfit to the complexity of the path and make it unable to sufficiently improve trajectory smoothness.

Hence, there must be careful tuning of the number of control points of the Bezier curve so that an optimal solution can be generated to fit the current environment. In the simulation, the threshold angle for a point to be considered as a turn point is set to be 5 degrees and the swarm size is set to be 80. The simulation is repeated 3 times to get the average number of turn points for the different number of control points.

Table 4.2: Simulation Result for Different Control Points.

No. of Control Points	Average no. of Turn Points		
	Robot 1	Robot 2	Robot 3
20	9	14	3
50	18	12	4
75	13	1	4
100	3	0	1
200	3	0	0

The result in Table 4.2 shows that there is a decrease in the average number of turn points in the pathway as the number of control points increases. For 20 control points, the percentage of turn points that occurred in 20 control points is 45%, 70%, and 15% for Robot 1, Robot 2, and Robot 3 respectively. For 50 control points, the percentage of turn points that occurred in 50 control points decreased to 36%, 24%, and 8% for Robot 1, Robot 2, and Robot 3 respectively. Then, at 100 control points, the percentage of turn points that occurred in 100 control points further decreases to 3%, 0%, and 1% for Robot 1, Robot 2, and Robot 3 respectively. If control points are increased further to 200 points, the percentage of turn points that occurred in 200 control points only shows a slight decrease to 1.5%, 0%, and 0% for Robot 1, Robot 2, and Robot 3 respectively. Hence, it has been concluded that using a control point of 100 is already enough to generate a smooth trajectory with the lowest turn points in the environment.

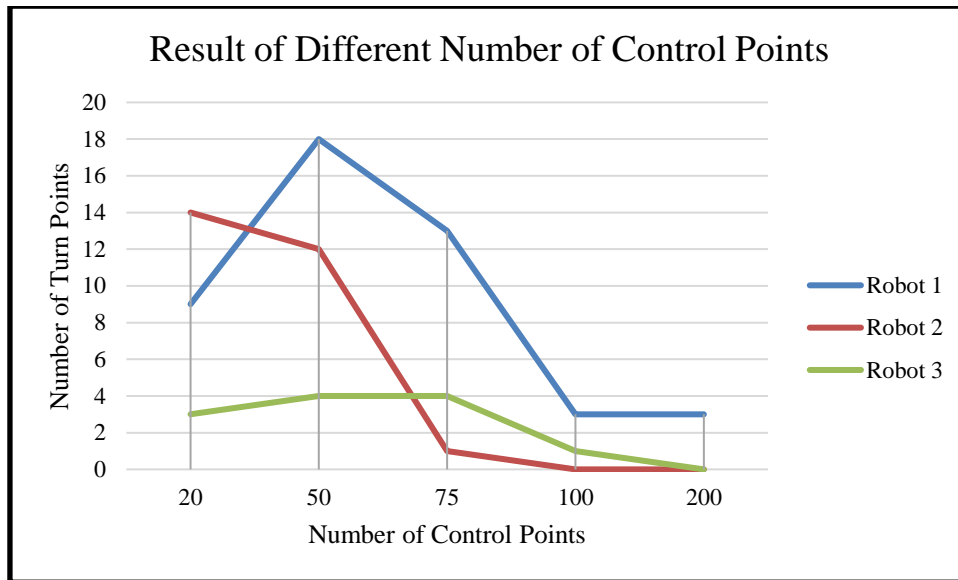


Figure 4.2: Line Graph of Result of Different Number of Control Points.

From Figure 4.2, it can be observed that for Robot 1, Robot 2, and Robot 3, as the number of control points increases, the number of turn points decreases. For instance, at 20 control points, the number of turn points for three robots is 9, 14, and 3 respectively while at 200 control points, the number of turn points decreases to 3, 0, and 0.

#### 4.2.3 Inertia weight, Cognitive and Social Acceleration Coefficients

The Particle Swarm Optimisation (PSO) algorithm uses numerous parameters to balance global exploration and local exploitation. One important parameter is the inertia weight, denoted by  $\omega$ . This weight determines how a particle's prior velocity affects its current course. A larger  $\omega$  value encourages exploitation by retaining the particle's previous velocity, allowing it to continue along its current path. While this method can speed up convergence to local optima, it also risks locking the algorithm in suboptimal regions, preventing it from finding the global optimum. In contrast, a smaller  $\omega$  value stimulates exploration by minimising the impact of a particle's previous velocity, allowing for greater directional shifts. This method allows particles to escape local optima and explore other parts of the search space. However, extensive exploration may hinder convergence, thereby extending the optimisation process. In addition to the inertia weight, the cognitive acceleration coefficient ( $c_1$ ) and the social acceleration coefficient ( $c_2$ ) are

critical components of the PSO algorithm. The cognitive coefficient determines how a particle's personal best solution affects its velocity update. A greater  $c_1$  value increases the particle's attraction to the individual optimal position, promoting exploitation in the area of that solution. In contrast, the social acceleration coefficient determines the particle's tendency to drift towards the global optimal solution discovered by the entire swarm. Increasing  $c_2$  encourages swarm convergence and exploitation in the globally optimal area, potentially speeding up the algorithm's convergence. By carefully modifying these parameters, the PSO algorithm may strike a delicate balance between exploration and exploitation, and swiftly traverse the search space while refining promising solutions for optimal path planning.

A higher  $c_1$  value, along with a lower  $c_2$  and a larger  $\omega$ , promotes strong individual exploitation while limiting exploration due to the high inertia weight. Figure 4.3 shows how this structure encourages local refinement of path lengths around the individual particles' best solutions. As a result of its large inertia weight and strong cognitive component, the algorithm achieves faster convergence and shorter execution times. Maintaining a higher  $c_1$  value while decreasing  $c_2$  and  $\omega$  can result in a shorter path length. The decreased inertia weight allows for greater exploration in the environment, which may lead to longer execution time and more directional shifts in the path, as shown in Figure 4.4. A comparison of Figures 4.3 and 4.4 demonstrates a clear contrast in generated trajectories. The high  $\omega$  value setup creates smoother pathway with good local refinement, whereas the low  $\omega$  value strategy produces trajectory with sharper twists and less refined segments, indicating that the particles are exploring more thoroughly.

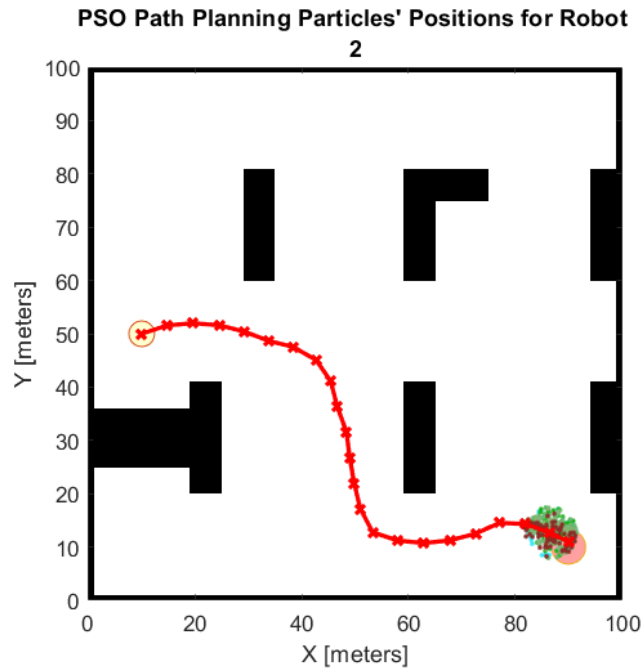


Figure 4.3: Path Generated from high  $c_1$ , low  $c_2$ , and high  $\omega$ .

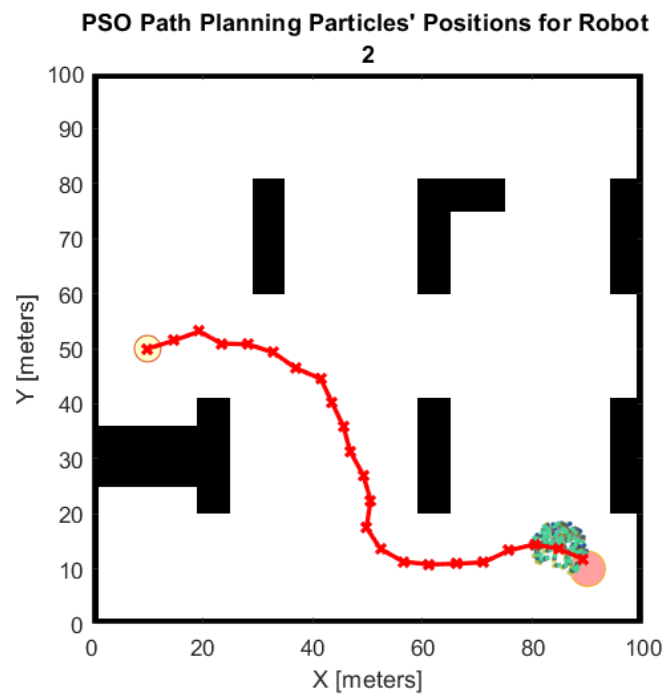


Figure 4.4: Path Generated from high  $c_1$ , low  $c_2$ , and low  $\omega$ .

A configuration with a lower  $c_1$ , a higher  $c_2$ , and a higher  $\omega$  encourages significant swarm convergence to the global optimal solution while restricting individual exploration due to the low cognitive coefficient and high inertia weight. This setting may result in reduced path lengths if the

swarm converges on an ideal solution. Furthermore, the high inertia weight and social coefficient may speed up convergence, resulting in shorter execution time. In contrast, a parameter combination with a lower  $c_1$ , higher  $c_2$ , and lower  $\omega$  also promotes swarm convergence to the global best solution. However, the low inertia weight allows for vigorous individual exploration, whereas the low cognitive coefficient prevents local path refining, resulting in longer path length. Furthermore, the low inertia weight and higher exploration can slow convergence, increasing execution time. A comparison of the data in Table 4.3 shows that a high  $c_1$  value, a low  $c_2$  value, and a low  $\omega$  value result in shorter path length than a configuration with a high  $c_1$ , a low  $c_2$ , and a high  $\omega$  value. This is due to the combined effect of the high  $c_1$  value, which allows for better exploitation of previously discovered good solutions, and the low  $\omega$  value allows for greater exploration because particles' current velocities are less bound by their previous trajectories. A high  $c_1$ , low  $c_2$ , and high  $\omega$  value method takes less time to execute. This is because the high  $\omega$  value restricts exploration, as particles' current velocities are more bound by their previous trajectories, causing them to proceed in the same direction, promoting faster convergence and lowering execution time, albeit at the expense of greater path length. When a PSO algorithm with a low  $c_1$ , a high  $c_2$ , and a high  $\omega$  value is analysed, it is found to have a shorter path length and a lower execution time than an algorithm with a low  $c_1$ , a high  $c_2$ , and a low  $\omega$  value. This phenomenon can be attributed to the combined effect of the high  $c_2$  value, which leads particles to quickly converge on the global best solution discovered by the swarm, lowering execution time and path length and the high  $\omega$  value, which restricts the particles' exploring capabilities, potentially reducing path length.

Table 4.3: Simulation Result for Different  $c_1$ ,  $c_2$ , and  $\omega$  Values.

$c_1$	$c_2$	$\omega$	Path Length (m)			Execution Time (s)		
			Robot 1	Robot 2	Robot 3	Robot 1	Robot 2	Robot 3
2	0	0.4	104.64	103.57	79.05	242.40	268.57	137.72
2	0	1.2	105.27	105.94	79.41	79.21	86.16	60.15
0	2	0.4	104.19	104.74	79.43	179.67	167.92	154.69
0	2	1.2	103.84	101.93	79.24	77.25	75.19	64.43

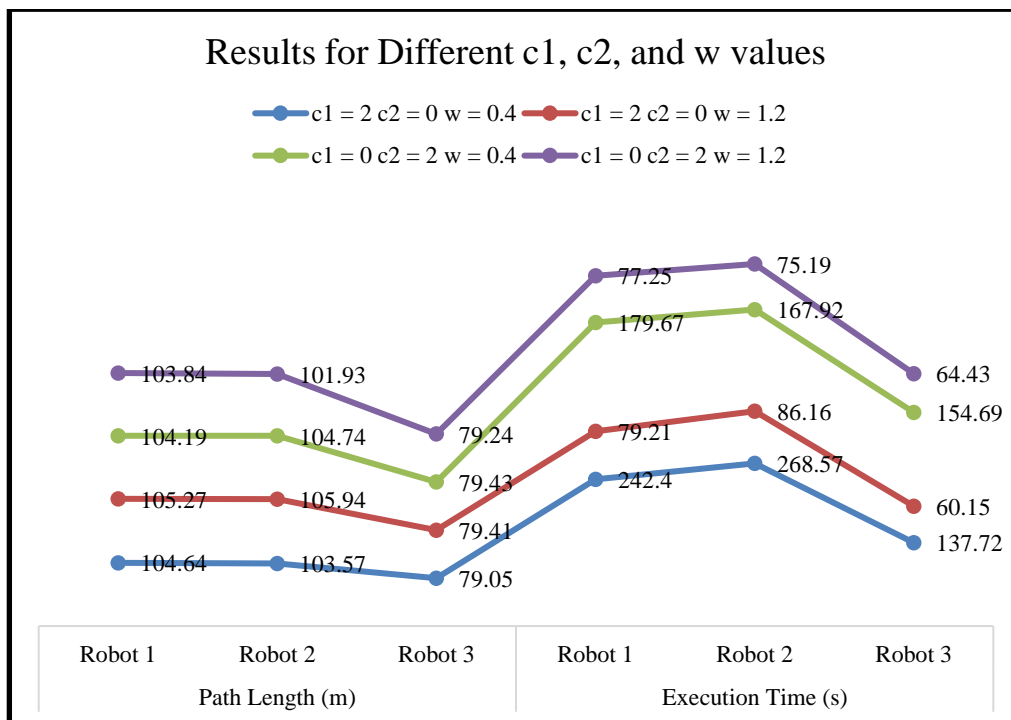
Figure 4.5: Line Graph of Results for Different  $c_1$ ,  $c_2$ , and  $\omega$  Values.

Figure 4.5 gives a clearer illustration of results for different  $c_1$ ,  $c_2$ , and  $\omega$  values. It can be observed that the path length and execution time is higher when  $c_1 = 2$ ,  $c_2 = 0$ , and  $\omega = 1.2$  than when  $c_1 = 2$ ,  $c_2 = 0$ , and  $\omega = 0.4$ . Additionally, the path length and execution time is also higher when  $c_1 = 0$ ,  $c_2 = 2$ , and  $\omega = 1.2$  than when  $c_1 = 0$ ,  $c_2 = 2$ , and  $\omega = 0.4$ . Hence, it concludes that PSO should use a linearly decreasing inertia weight approach, whereby the value of  $\omega$  begins at 0.95 and steadily decreases to 0.4 over iterations. Initially, the PSO method allows particles to explore the environment and locate the ideal solution with a higher  $\omega$  value. As iterations rise, the  $\omega$  value steadily drops, allowing particles to converge faster towards the optimum

solution with less constraint from past velocities. The  $c_1$  and  $c_2$  values are adjusted gradually, with the  $c_1$  value decreasing and the  $c_2$  value increasing as iterations progress. This is because, at the start of the PSO algorithm, particles rely mostly on their personal best solution; however, as iteration progresses, a rise in  $c_2$  value encourages convergence towards the global best position.

### 4.3 Simulation Results of MPSO and EPSO

EPSO and MPSO both apply the same path planning scheme and obstacle avoidance scheme during the optimization process to reach the target position. In the simulation, a total of three robots are utilized whereby if two robots encounter each other within their sensing range, they will avoid each other by activating an obstacle avoidance algorithm. Both robots will start rotating their heading direction to search for an unblocked direction so that they can slowly pass through the intersection point without colliding with each other. For the path planning scheme, the particles are initially placed within a fixed search region around the starting position. Within this search region, all particles start to search for the global best position. This is done by calculating the fitness value for each particle. To determine the fitness value of each particle, two factors will be affecting it. First, the Euclidean distance between particles' position and target position, and second, the distance between particles' position with surrounding obstacles. A shorter distance between the particles' position and target position as well as a longer distance between the particles' position and surrounding obstacles will generate a higher fitness value. Out of all particles' fitness values, the highest will be the local best fitness. When the local best fitness is higher than the global best fitness, the global best fitness will be replaced by the local best fitness. Then, the velocity and position of each particle are updated based on the equation  $v_i = \omega v_i + c_1 r_1 (p_i - x_i) + c_2 r_2 (p_g - x_i)$  and  $x_i = x_i + v_i$ . The global best position from this iteration of the algorithm is determined by the particles' search and is set as one of the waypoints for robots to follow through. The next iteration of the algorithm will then commence to search for the next waypoints for robots until reaching the target position.

The main difference between EPSO and MPSO is that EPSO applies the Bezier curve trajectory smoothing algorithm after the path planning scheme terminates while the MPSO does not include the trajectory smoothing scheme. Firstly, the number of control points on the Bezier curve pathway is set as 100 with a spacing,  $t$  of 0.01 from 0 to 1. The shape and smoothness of the path will be defined by the number of control points. Then, the algorithm will loop over each robot to extract each robot's generated waypoints from the path planning scheme. Bernstein basis function is calculated for each robot based on  $B_{i=0}^{no.of\ control\ points-1}(u) = \binom{3}{i}t^i(1-t)^{3-i}$ . The Bezier curve points are then calculated by summing element-wise product of waypoints and Bernstein basis function and plotted on the figure for each robot. All Bezier curve points generated will shape the smooth trajectory for each robot.

The environment in which the simulation will be conducted is in a warehouse layout. Three robots will be placed at different starting locations and travel across the warehouse environment to reach their respective destination. Due to the reason that randomness of each particle's starting location within the fixed region around the start point, it will cause changes to each particle's fitness value and hence cause slight changes in the result of the EPSO algorithm in terms of path length, execution time, and number of turn points. To tackle this issue, each environment is simulated five times to get the average result. Average path length, average execution time, and number of turn points of the three robots will be the evaluation criteria of EPSO and MPSO in three different warehouse environments. The parameters of EPSO and MPSO are fixed in all simulations conducted, as shown in Table 4.4. Each robot has a fixed starting point and ending point in all three warehouse environments as well, as shown in Table 4.5.



Table 4.4: Parameters of EPSO and MPSO.

Parameters of EPSO and MPSO	Value
Swarm size	80
Number of iterations	30
Maximum cognitive/social acceleration coefficient	2.0
Minimum cognitive/social acceleration coefficient	0.5
Maximum inertial weight	0.95
Minimum inertial weight	0.4
Number of robots	3
Environment size ( $m^2$ )	100 x 100
Number of control points	100

Table 4.5: Starting Position and Ending Position for each Robots.

	Starting Coordinate	Ending Coordinate
Robot 1	(90,50)	(10,10)
Robot 2	(10,50)	(90,10)
Robot 3	(10,90)	(90,90)

### 4.3.1 Comparison between MPSO and EPSO in Environment 1

The first environment, as shown in Figure 4.6, is designed based on a warehouse layout. This environment contains three shelves located around the center of the space, a loading station at the bottom left corner, and two unloading stations at the right side of the map, which are considered static obstacles inside the map. Among the three shelves, the first and the third shelves have rectangular shapes while the second shelf has an L shape. Robot 1 is set to move from the right side of the map towards the target point at the bottom left side of the map so that it can go towards the loading station to carry products. Robot 2 is set to move from the left side of the map towards the target point at the bottom right side of the map so that it can bring the products from the loading station to the unloading station. Then, Robot 3 will travel from the upper left side of the map towards the upper right side of the map, passing through shelf 1 and shelf 2.

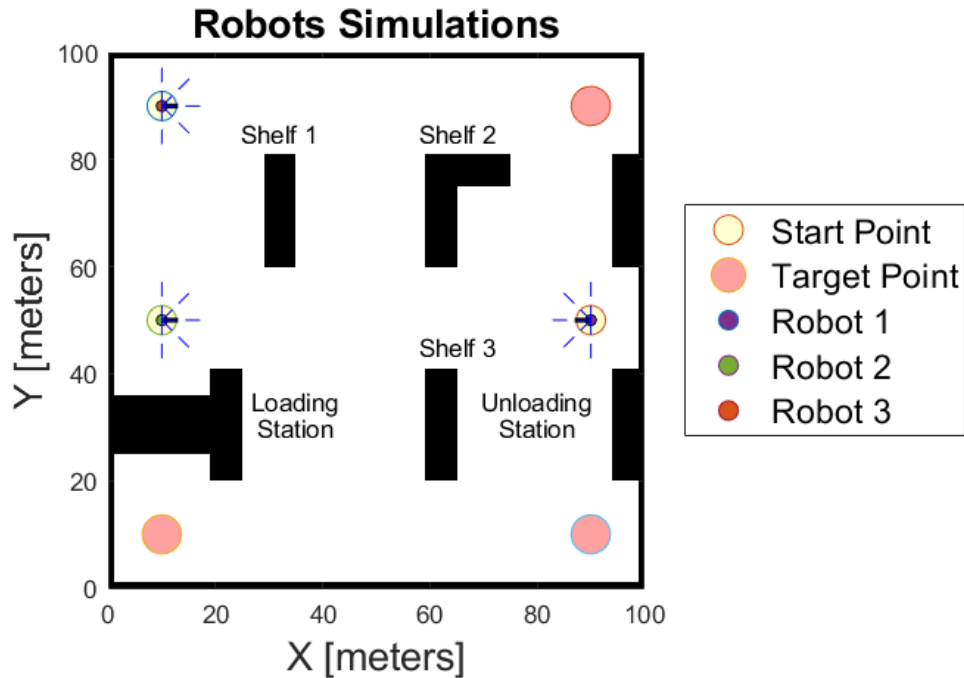


Figure 4.6: Environment 1.

#### 4.3.1.1 Result of MPSO Algorithm

The MPSO algorithm only includes the path planning scheme to generate the waypoints for a robot to travel from the start point to its target point. Figure 4.7 illustrates a total of 23 waypoints generated through the MPSO algorithm for Robot 1. Each blue circles that exist along the pathway represents a turn point in the pathway. There is a total of 15 turn points in the pathway of Robot 1. A waypoint will only be counted as a turning point if the direction exceeds a threshold angle of five degrees when proceeding to the next waypoint. The angle between consecutive line segments is counted using dot product based on  $\text{angle} = \arccos(\text{dot}(\text{vector1}, \text{vector2}) / (\text{norm}(\text{vector1}) * \text{norm}(\text{vector2})))$ . Next, Figure 4.8 shows that Robot 2 passes through a total of 23 waypoints to reach its target point. There are 18 turn points along the pathway of Robot 2. For Robot 3, Figure 4.9 indicates that it has 18 waypoints and out of the 18 waypoints, there are 12 turn points.

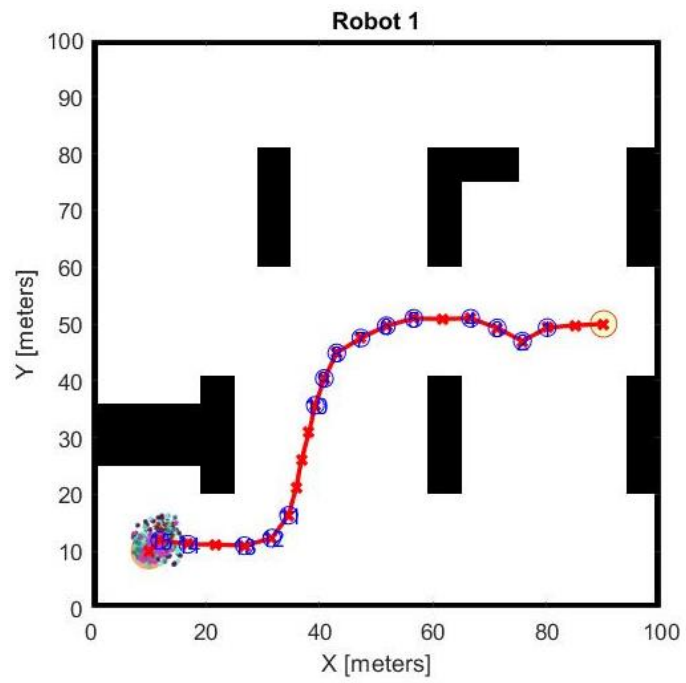


Figure 4.7: Pathway Generated for Robot 1 of MPSO.

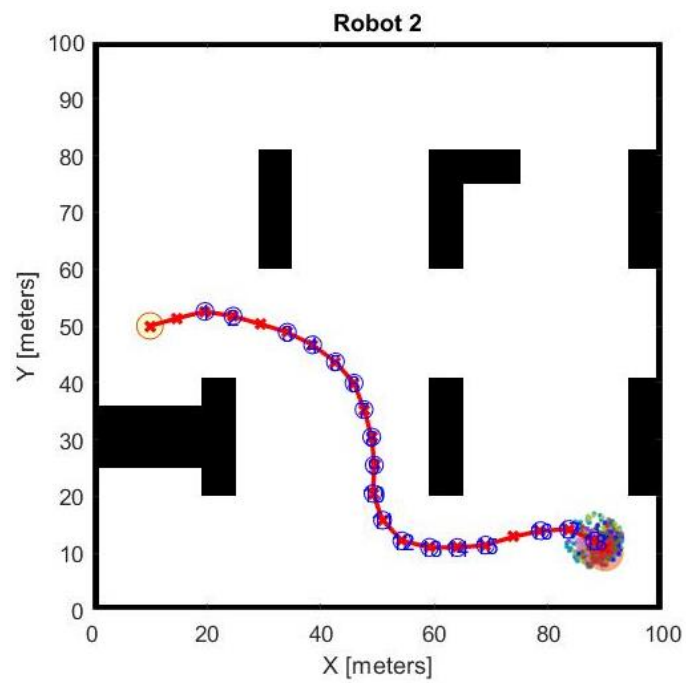


Figure 4.8: Pathway Generated for Robot 2 of MPSO.

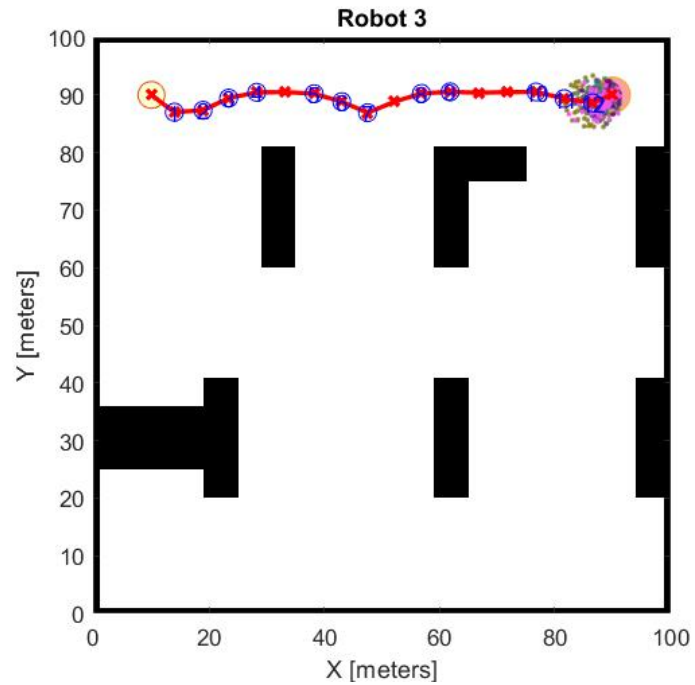


Figure 4.9: Pathway Generated for Robot 3 of MPSO.

Figure 4.10 shows the illustration of three robots traveling from their start point to their respective target points. Their pathways are all represented by dash lines in different colours. It can be observed that Robot 1 and Robot 2 intersect with each other along their pathway and both of them rotated their headings to avoid each other, which can be seen from the sudden change of direction of dashed lines for both robots. On the other hand, Figure 4.11 gives a clearer view of waypoints by plotting each waypoint of robots on the figure.

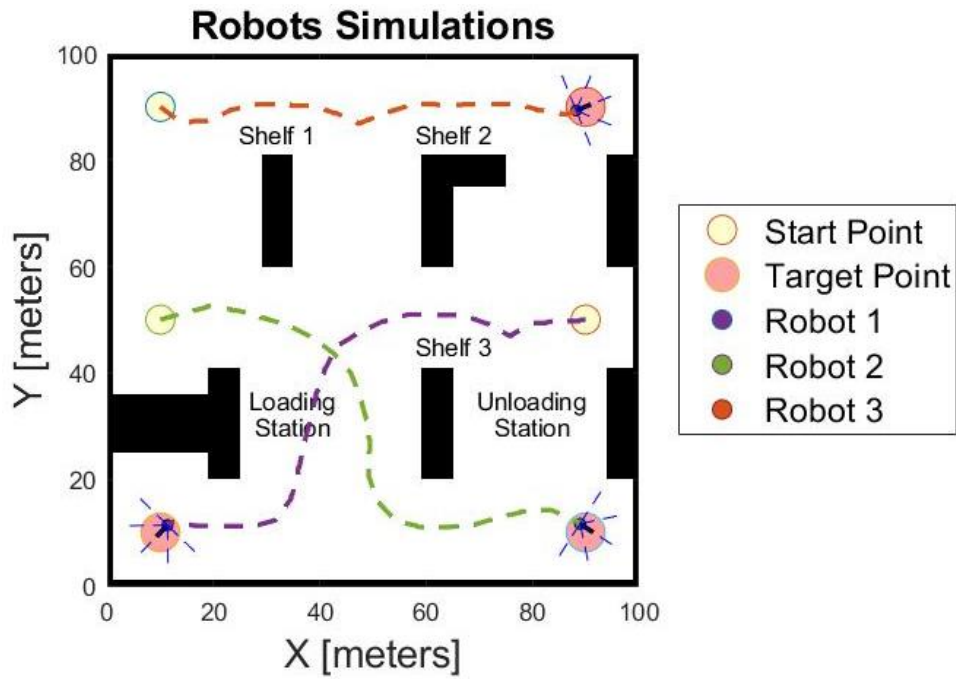


Figure 4.10: Simulation of Robots Travelling According to MPSO Pathway.

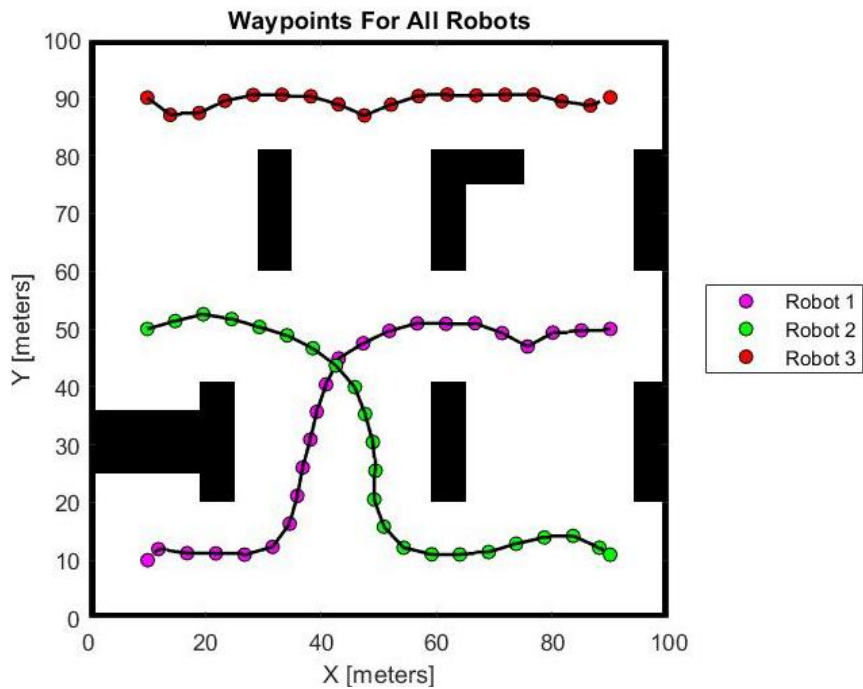


Figure 4.11: Plotting of MPSO Waypoints of all Robots.

Figure 4.12 shows the graph of global best fitness with respect to number of iterations. It can be observed that from iteration one, the total global best fitness of all robots starts from 291.11 and gradually decreases over the iterations. When iteration 24 is reached, all robots successfully reach

their respective target position, and the total global best fitness of all robots decreases to 32.48. The gradual decrease of the global best fitness value symbolizes the convergence towards optimal solutions in the search for optimal pathways.

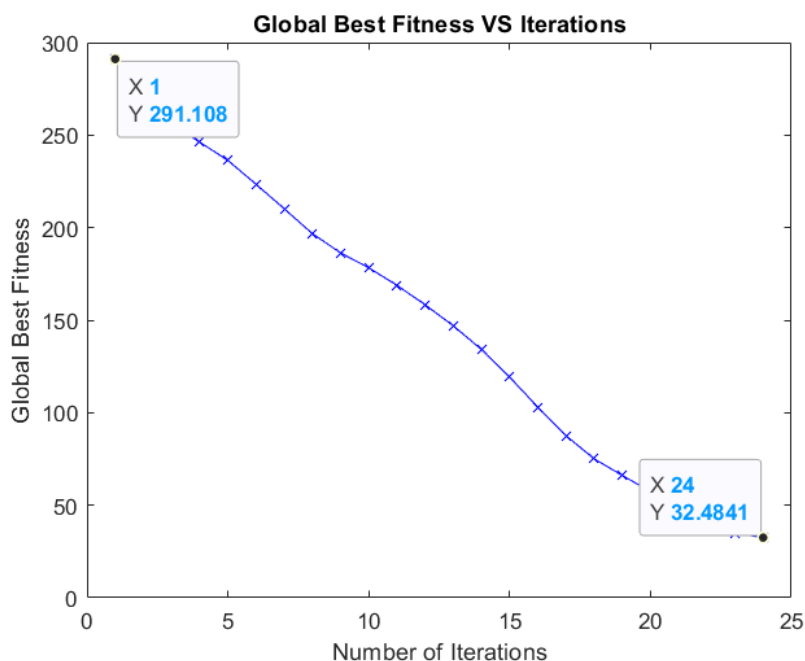


Figure 4.12: Graph of Global Best Fitness Vs. Iterations of MPSO Algorithm.

#### 4.3.1.2 Result of EPSO Algorithm

As mentioned earlier, the EPSO algorithm implements the Bezier curve trajectory smoothing algorithm after the path planning scheme is completed. From Figure 4.13, it can be observed that the blue line, which is produced from the Bezier curve trajectory smoothing algorithm, possesses greater smoothness and a lesser number of turn points than the red line, which is generated from the path planning scheme. Each red circle along the blue line represents each turn point present along the pathway. From Figure 4.14, and Figure 4.15, it can be observed that there is no turn point present along the pathway at all. This indicates that among 100 control points, there is not a single waypoint that has exceeded the threshold angle of five degrees when proceeding to the next waypoint. The angle of the line segment between each two waypoints can be calculated using  $\text{angle} = \arccos(\frac{\text{dot}(\text{vector1}, \text{vector2})}{(\text{norm}(\text{vector1}) * \text{norm}(\text{vector2}))})$ . The shelves, loading station, and unloading

station, which are treated as static obstacles, will also be avoided when updating the velocities and positions of particles.

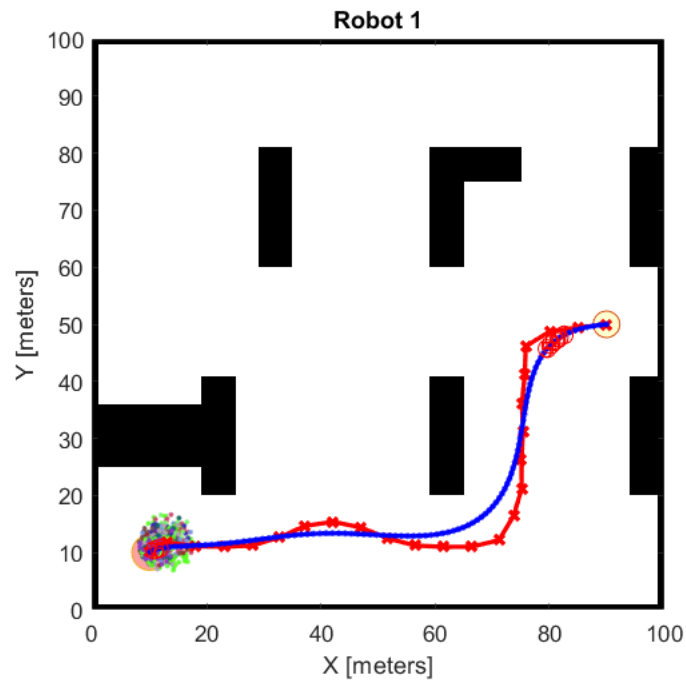


Figure 4.13: Pathways Generated for Robot 1 of EPSO.

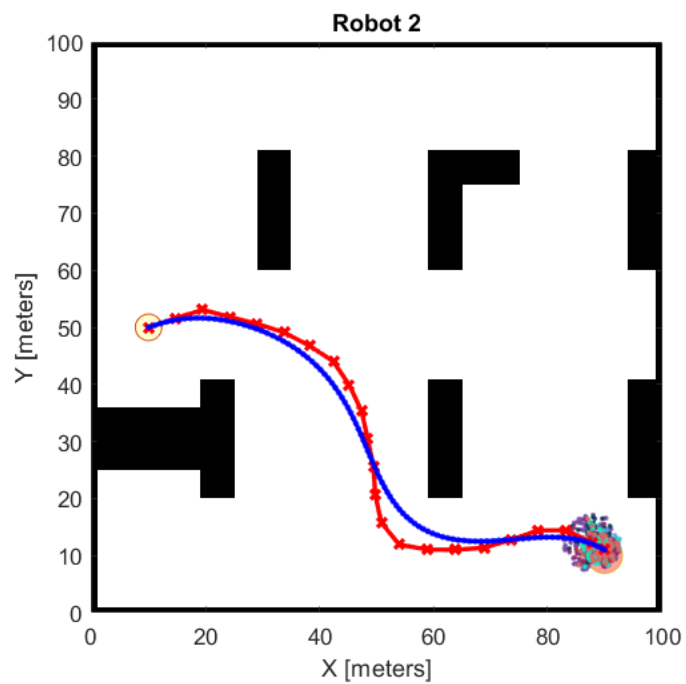


Figure 4.14: Pathways Generated for Robot 2 of EPSO.

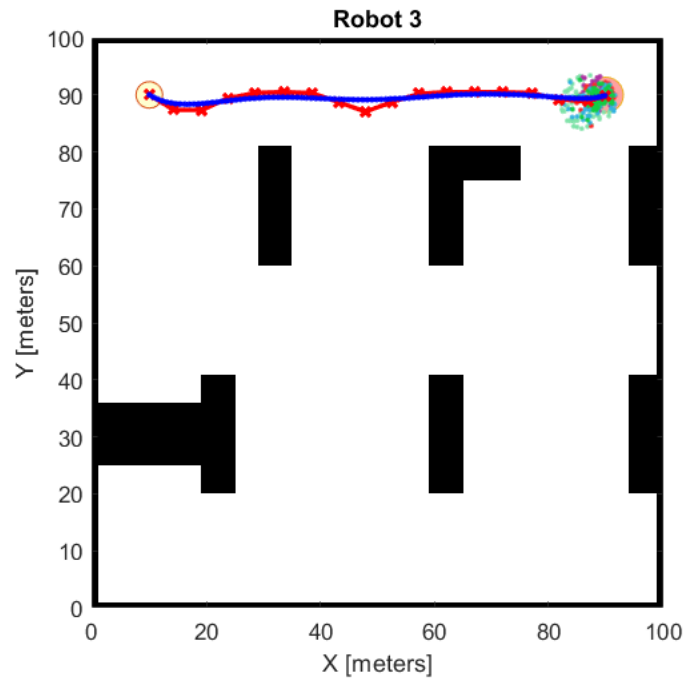


Figure 4.15: Pathways Generated for Robot 3 of EPSO.

The simulation of robots traveling according to the smooth trajectories generated from the Bezier curve trajectory smoothing algorithm can be observed in Figure 4.16. Robot 1's pathway is represented in purple colour, Robot 2's pathway has green colour and Robot 3's pathway is coloured in red. Figure 4.17 gives a clearer picture of all 100 control points of the Bezier Curve pathway, which are now treated as waypoints for each robot to follow.



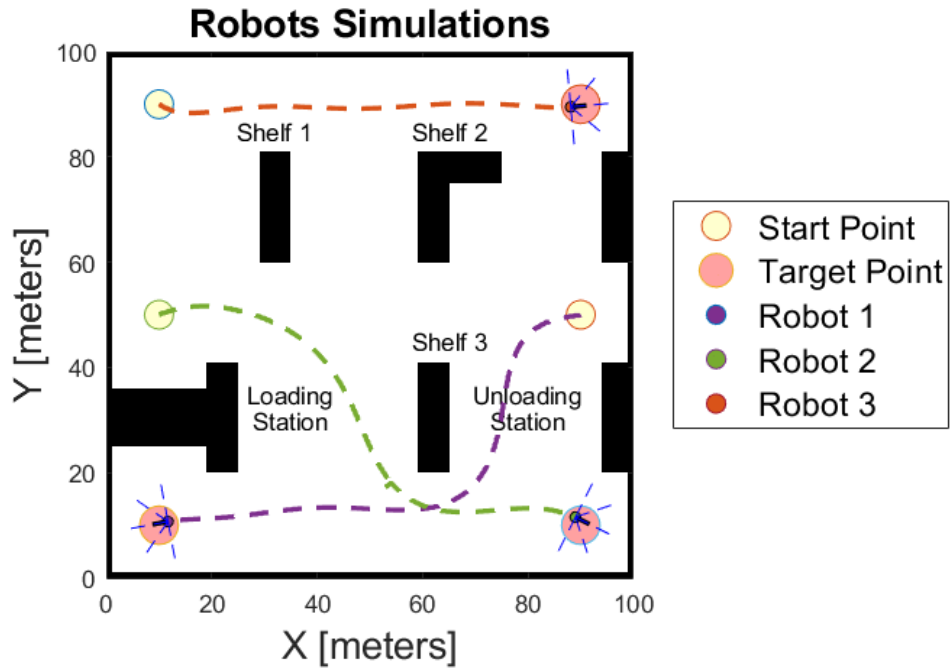


Figure 4.16: Simulation of Robots Travelling According to EPSO Pathway.

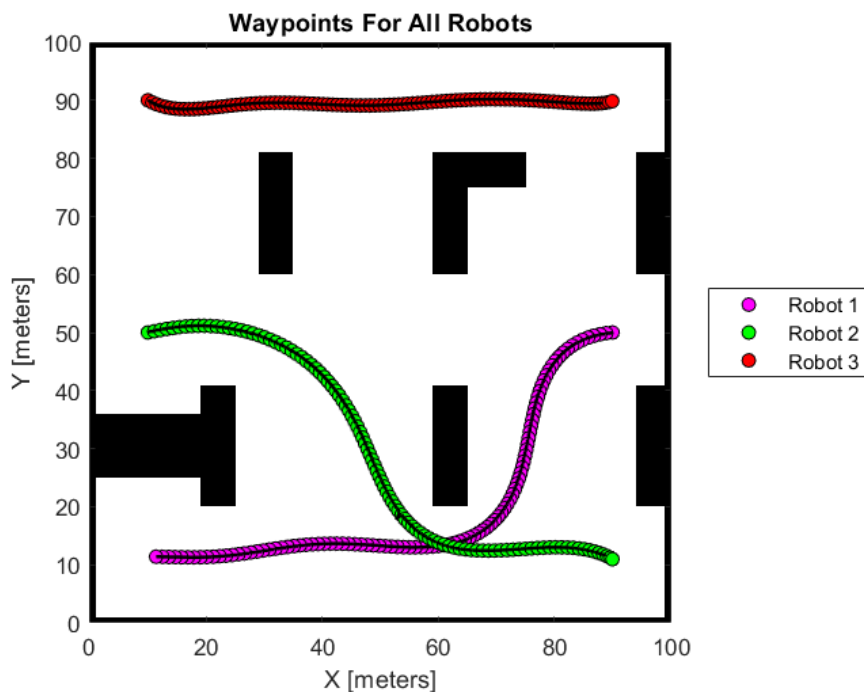


Figure 4.17: Plotting of EPSO Waypoints of all Robots.

The global best fitness graph concerning the number of iterations is displayed in Figure 4.18. It is evident that the total global best fitness of all robots begins at 291.11 in iteration one and progressively declines throughout the iterations. All robots successfully achieve their designated goal positions

by iteration 24, at which point the total global best fitness of all robots drops to 32.21. The progressive decline in the global best fitness value represents the convergence of the search for the optimum pathway toward the ideal solution.

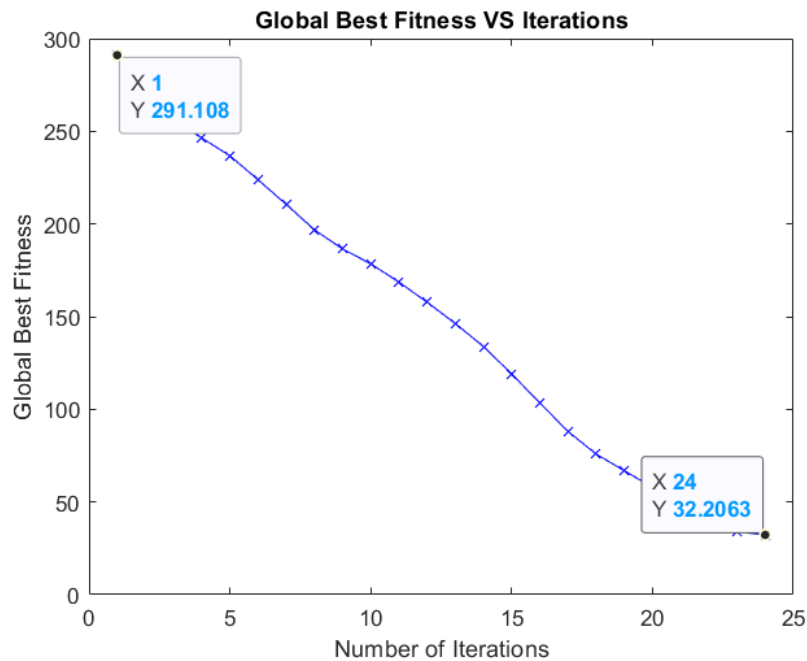


Figure 4.18: Graph of Global Best Fitness Vs. Iterations of EPSO Algorithm.

The evaluation criteria of MPSO and EPSO are based on the average path length, average execution time, and number of turn points of robots in the environment. The result can be shown in Table 4.6. From Table 4.6, it can be observed that for all three robots, the average path length, average execution time, and the number of turn points of the EPSO algorithm are lower than the MPSO algorithm. This clearly shows the positive effect of EPSO having a Bezier curve trajectory smoothing algorithm in the reduction of robots' path length, execution time, and number of turn points. Both the MPSO and EPSO algorithm is run for a total of five times in the same environment to tackle the randomness of the PSO algorithm. The result of MPSO and EPSO for each simulation in Environment 1 can be shown in Table 4.7 and Table 4.8 respectively. Figure 4.19 showcases the error bar of the path length of EPSO for all robots in Environment 1 and it can be observed that the length of the error bar is very short and nearly cannot be seen. This means that the data has a low standard deviation and is tightly clustered around the average value.

Next, Figure 4.20 displays the error bar of the execution time of EPSO for all robots in Environment 1. The standard deviation for execution time data of Robot 1, Robot 2, and Robot 3 are 3.49, 5.09, and 7 respectively. This indicates that execution time data obtained from every simulation differs slightly from the average execution time.

Table 4.6: Results of MPSO and EPSO in Environment 1.

	Average Path Length (meters)		Average Execution Time (s)		Number of Turn Points	
	MPSO	EPSO	MPSO	EPSO	MPSO	EPSO
Robot 1	117.51	104.42	112.90	95.25	15	7
Robot 2	114.92	103.51	105.55	101.31	18	0
Robot 3	81.82	79.24	77.92	77.48	12	0

Table 4.7: Result of MPSO for each Simulation in Environment 1.

MPSO	Robot 1		Robot 2		Robot 3	
	Path Length	Execution Time	Path Length	Execution Time	Path Length	Execution Time
First Simulation	114.76	105.83	110.03	98.76	81.54	74.86
Second Simulation	115.94	112.48	117.92	104.19	82.04	76.51
Third Simulation	116.54	121.05	120.35	121.14	81.99	79.67
Fourth Simulation	115.21	114.83	114.52	108.59	82.00	83.21
Fifth Simulation	125.10	110.30	111.79	95.05	81.51	75.37

Table 4.8: Result of EPSO for each Simulation in Environment 1.

EPSO	Robot 1		Robot 2		Robot 3	
	Path Length	Execution Time	Path Length	Execution Time	Path Length	Execution Time
First Simulation	104.91	93.46	105.19	96.99	79.19	69.56
Second Simulation	103.79	92.65	104.63	108.80	79.36	74.40
Third Simulation	104.50	100.38	103.23	104.39	79.24	77.09
Fourth Simulation	104.85	92.41	101.61	98.11	79.26	77.79
Fifth Simulation	104.07	97.34	102.88	98.25	79.14	88.59

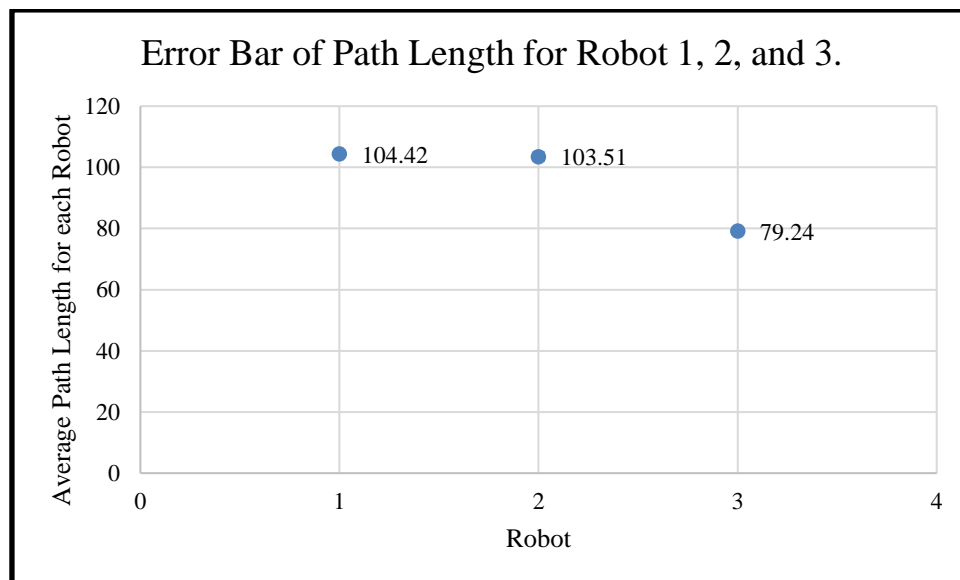


Figure 4.19: Error Bar of EPSO Path Length for Robot 1, 2, and 3.

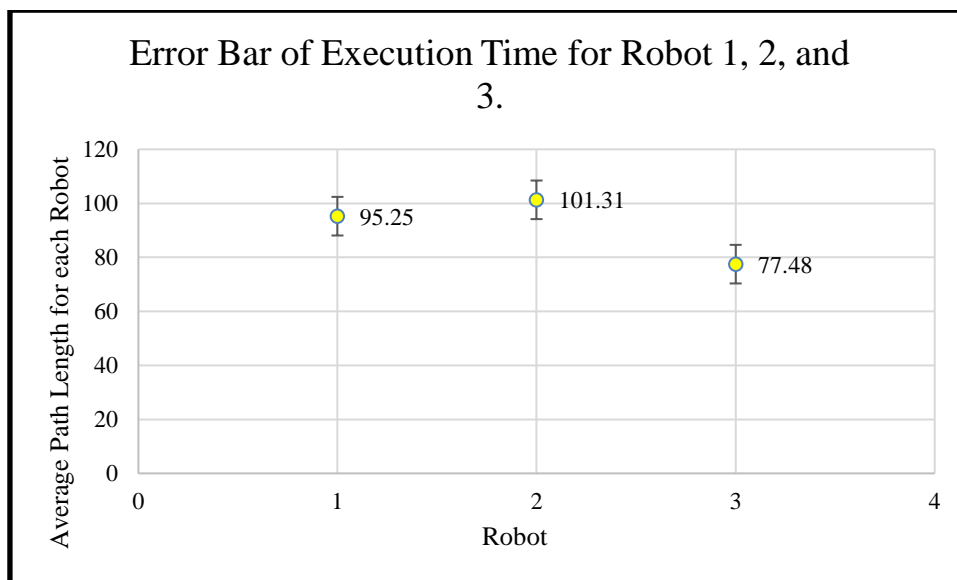


Figure 4.20: Error Bar of EPSO Execution Time for Robot 1, 2, and 3.

#### 4.3.2 Comparison between MPSO and EPSO in Environment 2

Figure 4.21 shows the warehouse layout of Environment 2. In Environment 2, there is a T-shaped loading station, a T-shaped unloading station, two L-shaped shelves, which are shelf 1 and shelf 5, a square shelf 2, a circular shelf, and shelf 4. These shelves, loading station, and unloading station are treated as static obstacles in the environment by which robots shall not collide with them. Robot 1 starts from the unloading station at the right side of the map and will travel across the map to reach its target point of the loading station at the bottom left corner of the map. Robot 2 will travel from the loading station and pass by shelf 4 to reach its target point beside shelf 5. Robot 3 starts at the upper left corner of the map and it will pass by shelf 1 and shelf 2 to reach its target point beside the unloading station. In Environment 2, the number of obstacles increases to 7 compared to Environment 1 which has 6 obstacles. Also, the obstacles' shapes become more varied compared to Environment 1 in which most of the obstacles are rectangular and T-shape.

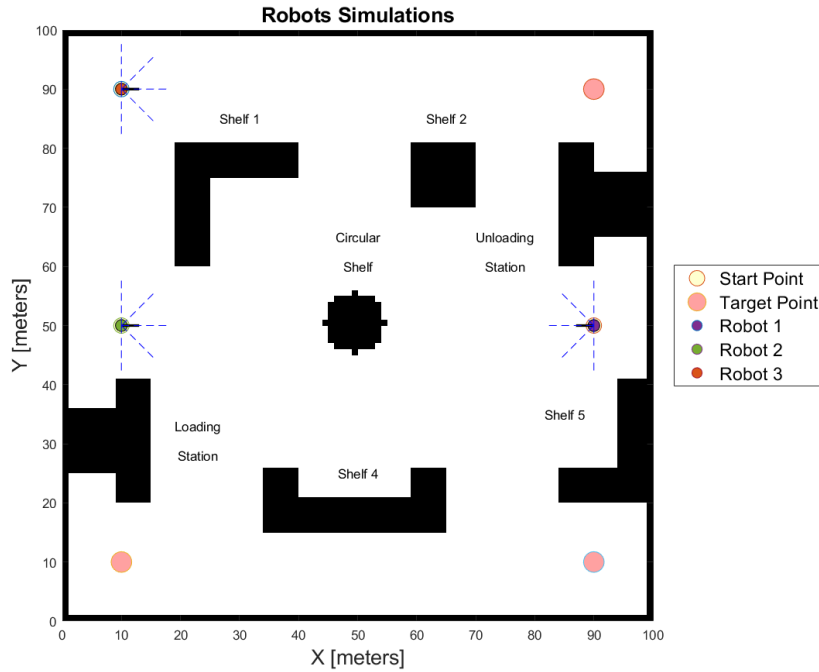


Figure 4.21: Environment 2.

#### 4.3.2.1 Result of MPSO Algorithm

MPSO algorithm searches for and plans each robot's pathway from its respective starting position to its ending position without considering the smoothness of the trajectory. Figure 4.22 showcases the pathway generated for Robot 1 to travel towards its target point. When the algorithm detects that it is blocked by shelf 4 in its path, it detours around the obstacle to avoid colliding with it. There is a total of 22 waypoints, represented by the red cross, exist in the path generated for Robot 1, and among the 22 waypoints, 14 turn points are detected. The turn points are represented by blue circles on the waypoints. For Robot 2, as shown in Figure 4.23, there are 23 waypoints generated for it to reach its target point. Among the 23 waypoints, there are 15 turn points for Robot 2. Next, there are 17 waypoints in the pathway generated and eight turn points out of all waypoints for Robot 3 in Figure 4.24.

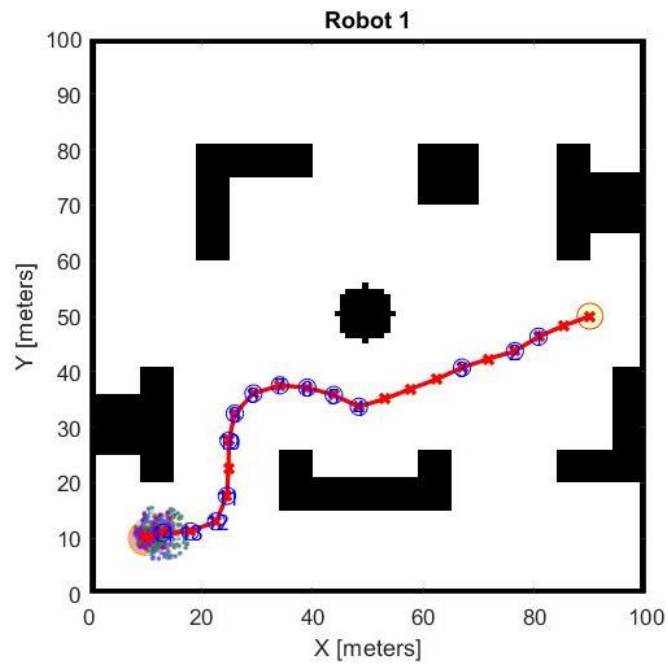


Figure 4.22: Pathway Generated for Robot 1 of MPSO.

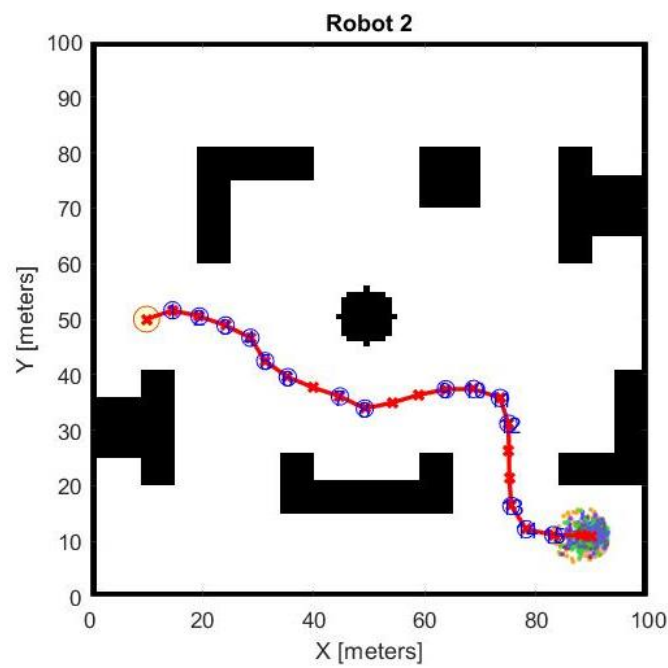


Figure 4.23: Pathway Generated for Robot 2 of MPSO.

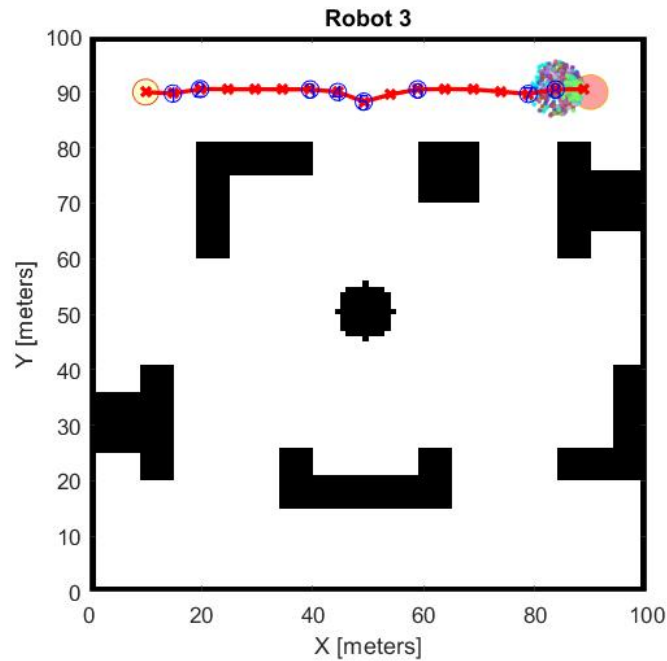


Figure 4.24: Pathway Generated for Robot 3 of MPSO.

In Figure 4.25, the pathways for each robot are represented in dotted lines with different colours. For Robot 1, its pathway is represented by purple colour. Robot 2 is represented by a green colour and red colour for Robot 3. It can be observed that there is an intersection of pathways between Robot 1 and Robot 2. However, when both robots detect each other within their sensor range, they will turn their headings to other directions to find a space to move forward so that both robots will not collide with each other. Figure 4.26 plots all the waypoints of pathways for all robots to give a clearer insight into each path.



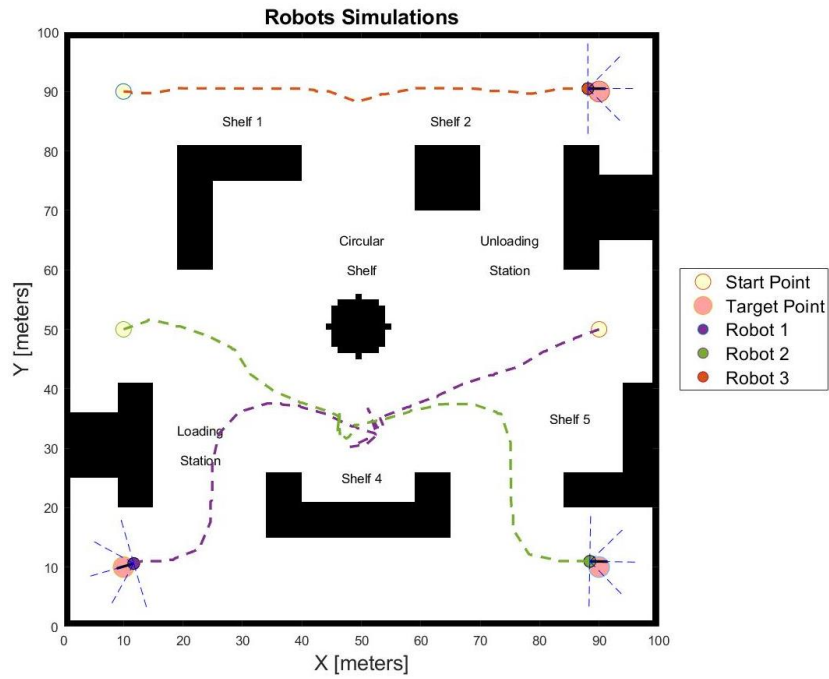


Figure 4.25: Simulation of Robots Travelling According to MPSO Pathway.

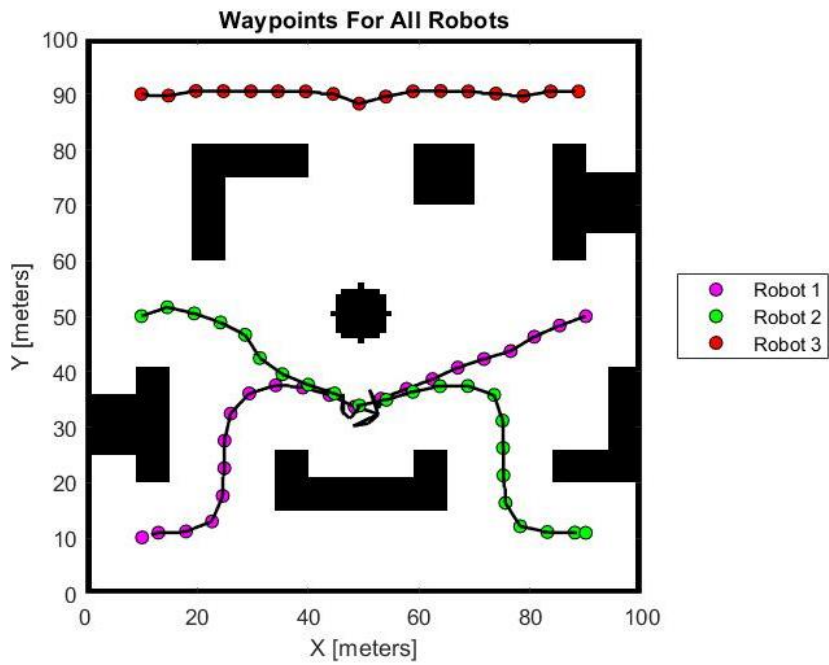


Figure 4.26: Plotting of MPSO Waypoints of all Robots.

From Figure 4.27, it can be observed that the total global best fitness for all three robots decreases as the iterations increase. At iteration one, the total global best fitness is 291.11 and the value decreases to 34.06 at iteration

23. This indicates that at iteration 23, all three robots completed their optimization process and reached their respective destinations.

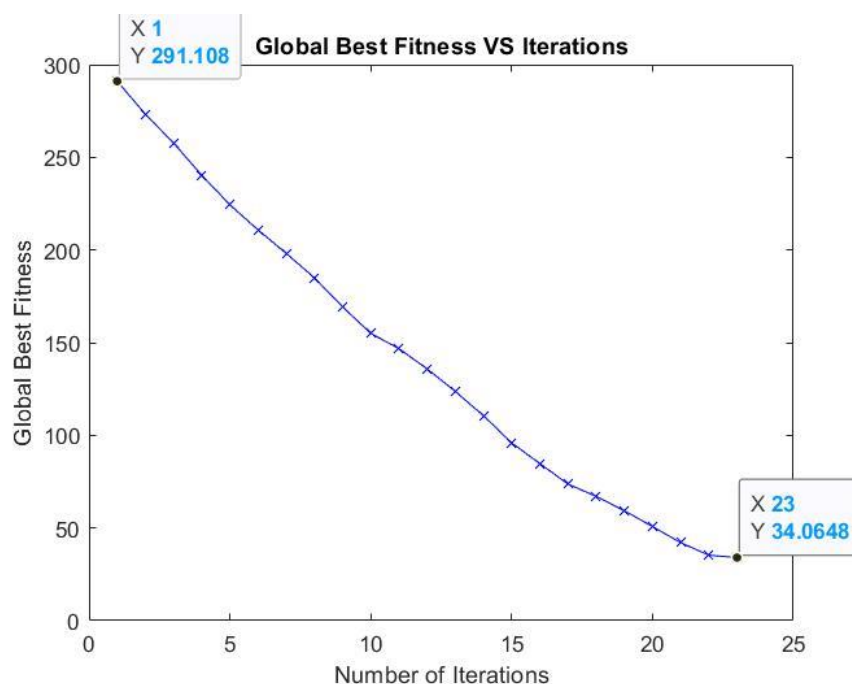


Figure 4.27: Graph of Global Best Fitness Vs. Iterations of MPSO Algorithm.

#### 4.3.2.2 Result of EPSO Algorithm

As stated earlier, the EPSO algorithm integrates the Bezier curve trajectory smoothing algorithm after the completion of path planning. The paths generated from the trajectory smoothing algorithm are represented by blue colour and it has 100 control points present in the paths. Figure 4.28 shows the two paths generated for Robot 1. The red colour path is generated by the path planning scheme and each waypoint is represented by the red cross, and the path is connected with red solid lines. Then, by importing these waypoints into the Bezier curve trajectory smoothing algorithm, a new smooth path can be generated, which is represented by blue solid lines. It can be observed that there is no turn points exist along the smooth path of Robot 1. For Robot 2, its pathway is illustrated in Figure 4.29. From Figure 4.29, it can be seen that there are four turn points, represented by red circles, exist in the smooth pathway generated. These turn points are located towards the end of the pathway when the robot is about to reach its target point. Next, from Figure 4.30, it can be seen that the smooth path generated transforms the sharp turn at

the middle of the original red pathway with a straight and smooth path for the robot to travel. There are also no turn points in Robot 3's pathway.

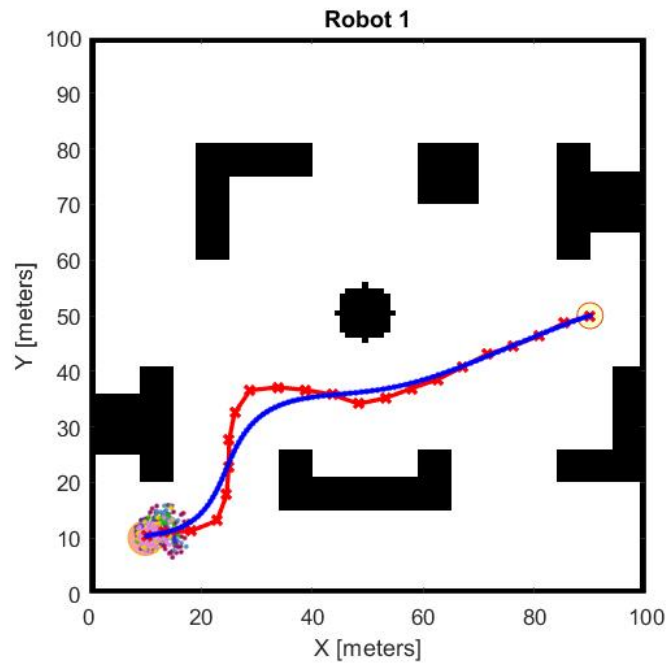


Figure 4.28: Pathways Generated for Robot 1 of EPSO.

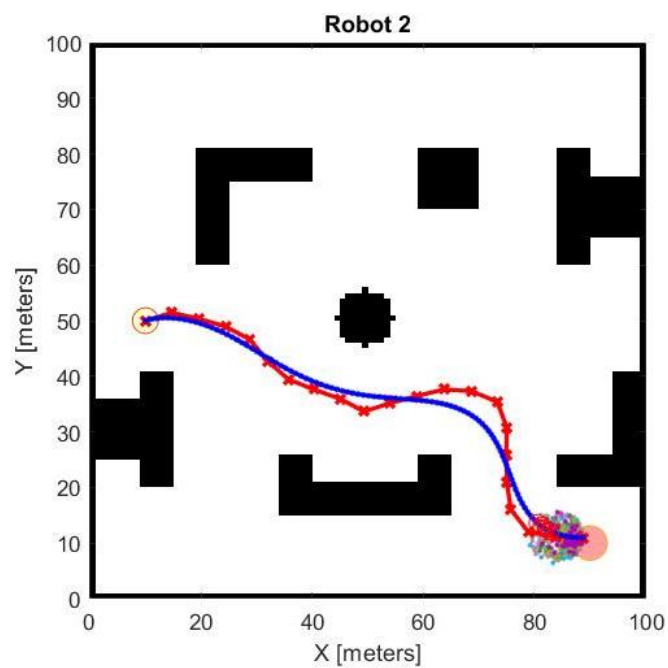


Figure 4.29: Pathways Generated for Robot 2 of EPSO.

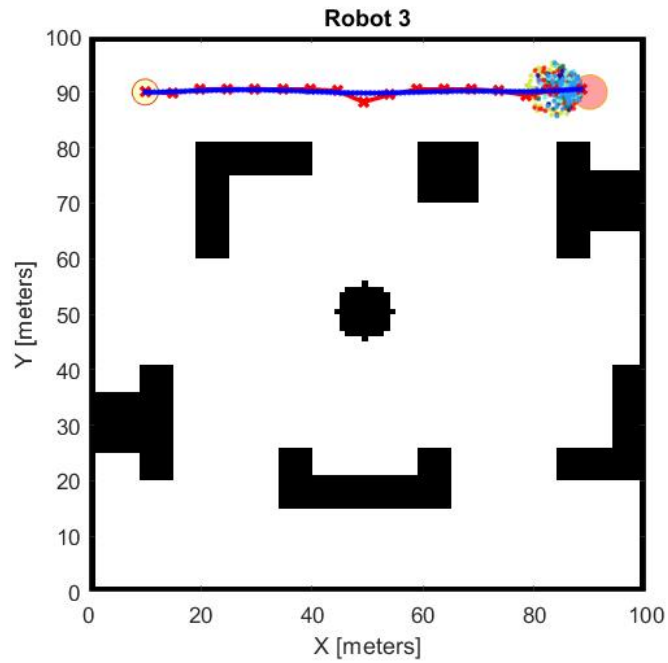


Figure 4.30: Pathways Generated for Robot 3 of EPSO.

The simulation of robots traveling according to the EPSO pathway is illustrated in Figure 4.31. The path of Robot 1 is coloured purple, the path of Robot 2 is green, and the path of Robot 3 is coloured red. The pathways followed by the robots present greater smoothness compared to MPSO. A more accurate depiction of the 100 control points of the Bezier Curve pathway is shown in Figure 4.32. These points are now regarded as path indicators that each robot must follow.

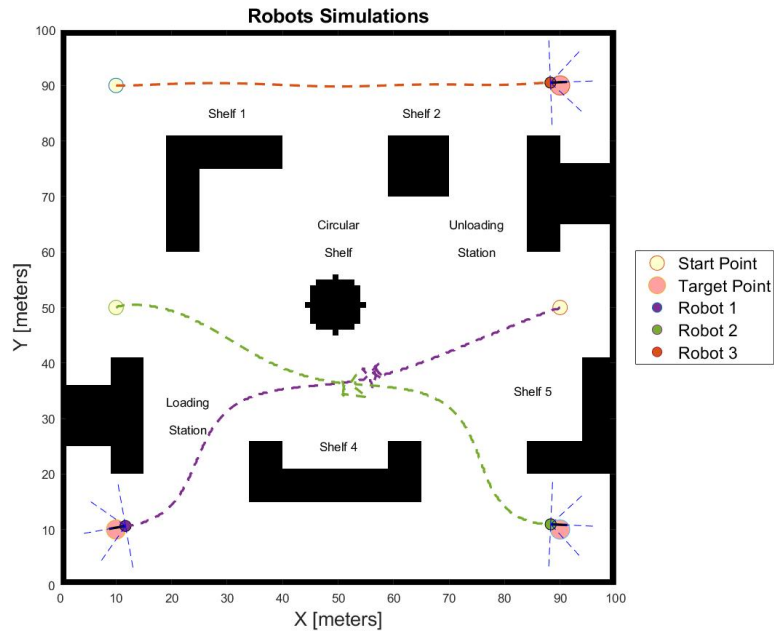


Figure 4.31: Simulation of Robots Travelling According to EPSO Pathway.

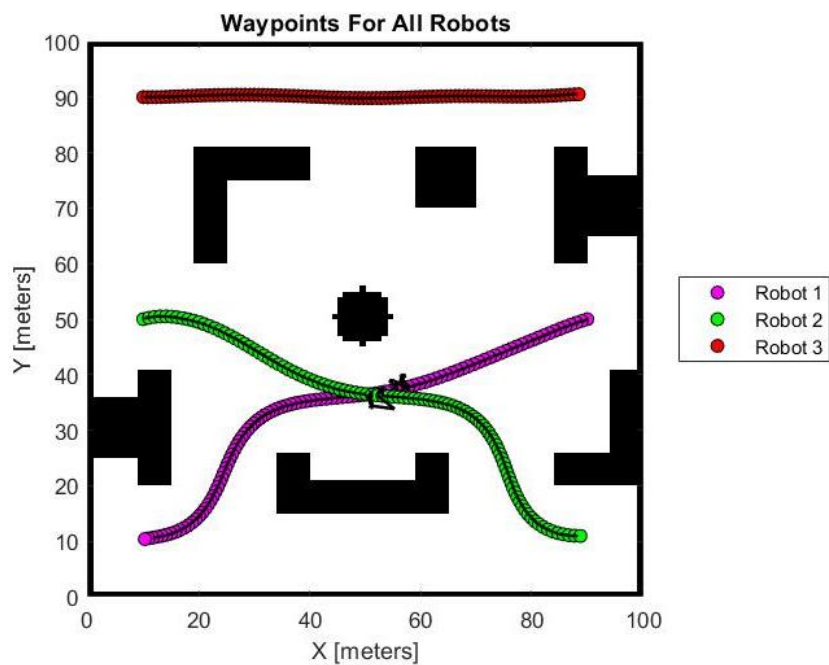


Figure 4.32: Plotting of EPSO Waypoints of all Robots.

Figure 4.33 showcases the graph of global best fitness versus iterations of the EPSO algorithm for Environment 2. At iteration one, the total global best fitness for all three robots is at a maximum value of 291.11. As the

iteration progresses, the global best fitness value decreases and finally at iteration 22, it reaches a minimum value of 35.01.

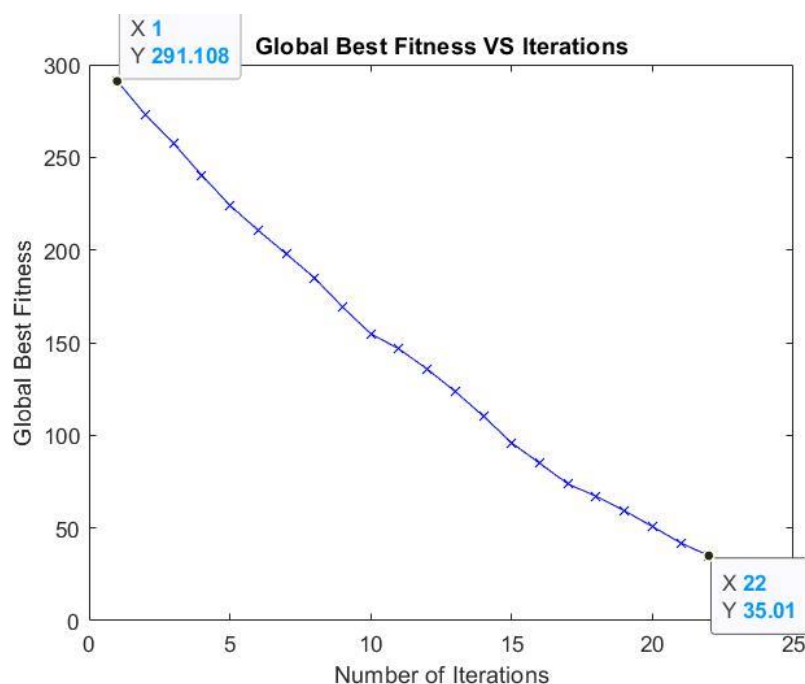


Figure 4.33: Graph of Global Best Fitness Vs. Iterations of EPSO Algorithm.

To compare the result between MPSO and EPSO in Environment 2, each algorithm is evaluated based on path length, execution time, and number of turn points of robots as shown in Table 4.9. From Table 4.9, it can be observed that the average path length of all robots for EPSO is shorter than MPSO. For the average execution time, the average execution time of Robot 1 for EPSO is longer than MPSO by 4.49 seconds. The average execution time of Robot 2 for EPSO is longer than MPSO by 6.82 seconds. This is because the pathways of Robot 1 and Robot 2 intersect with each other in the middle of the path. This intersection causes both robots to activate their obstacle avoidance algorithm. Both robots will turn their headings to other directions to try to search for a direction free of obstacles. However, the directions to which the robots will turn are determined randomly and in some cases, this will cause the robots to take more time to find a direction without obstacle. Hence, this causes the execution time of EPSO for Robot 1 and Robot 2 to exceed the execution time of MPSO. The average execution time of Robot 3 for EPSO is shorter than MPSO by 4.76 seconds because the pathway of Robot 3 is not

blocked by any obstacles. For the number of turn points, it can be observed that for all robots, the number of turn points of EPSO is lower than MPSO. Next, Table 4.10 displays the result of MPSO for each simulation of each robot in Environment 2. Table 4.11 shows the result of EPSO for each simulation of each robot in Environment 2. The simulation is repeated for five times for both EPSO and MPSO so that the average result can be obtained. Moreover, Figure 4.34 and Figure 4.35 illustrate the error bar of path length and execution time respectively for all robots of the EPSO algorithm. From Figure 4.34, it can be observed that the length of the error bar is very short, which indicates that the data is tightly clustered around the average value of path length and only has low standard deviations. Furthermore, from Figure 4.35, it can also be observed that the length of the error bar for Robot 1 is longer than for Robot 2 and Robot 3. A longer length of the error bar indicates that the execution time has a higher standard deviation of 15.12 and that the execution time data is more spread out for Robot 1.

Table 4.9: Result of MPSO and EPSO in Environment 2.

	Average Path Length (meters)		Average Execution Time (s)		Number of Turn Points	
	MPSO	EPSO	MPSO	EPSO	MPSO	EPSO
Robot 1	124.25	116.50	114.30	118.79	14	0
Robot 2	118.60	114.83	115.30	122.12	15	4
Robot 3	79.63	78.31	81.65	76.89	8	0

Table 4.10: Result of MPSO for each Simulation in Environment 2.

MPSO	Robot 1		Robot 2		Robot 3	
	Path Length	Execution Time	Path Length	Execution Time	Path Length	Execution Time
First Simulation	124.13	134.15	118.44	137.16	79.63	86.43
Second Simulation	124.13	117.81	118.44	121.68	79.63	83.18
Third Simulation	124.73	108.08	119.26	100.16	79.63	68.47
Fourth Simulation	124.13	103.46	118.44	106.73	79.63	72.32
Fifth Simulation	124.13	107.99	118.44	110.79	79.63	97.84

Table 4.11: Result of EPSO for each Simulation in Environment 2.

EPSO	Robot 1		Robot 2		Robot 3	
	Path Length	Execution Time	Path Length	Execution Time	Path Length	Execution Time
First Simulation	117.75	140.96	116.41	126.15	78.39	80.50
Second Simulation	120.36	121.94	114.62	133.46	78.30	81.22
Third Simulation	112.3	104.75	111.62	109.92	78.33	72.63
Fourth Simulation	120.36	121.85	114.62	127.83	78.30	76.20
Fifth Simulation	111.7	104.43	116.9	113.24	78.23	73.91



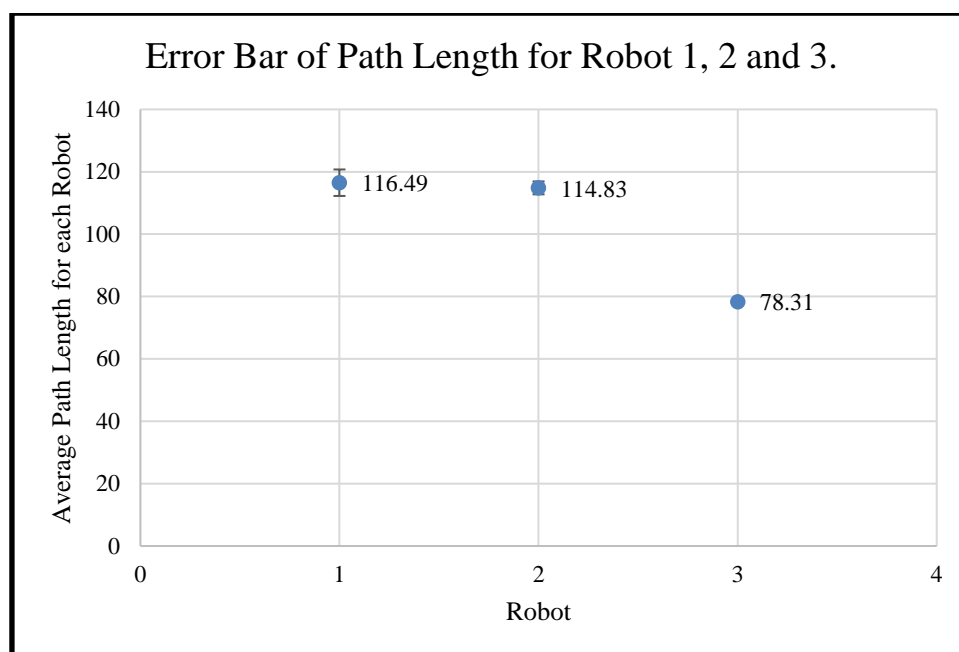


Figure 4.34: Error Bar of EPSO Path Length for Robot 1, 2, and 3.

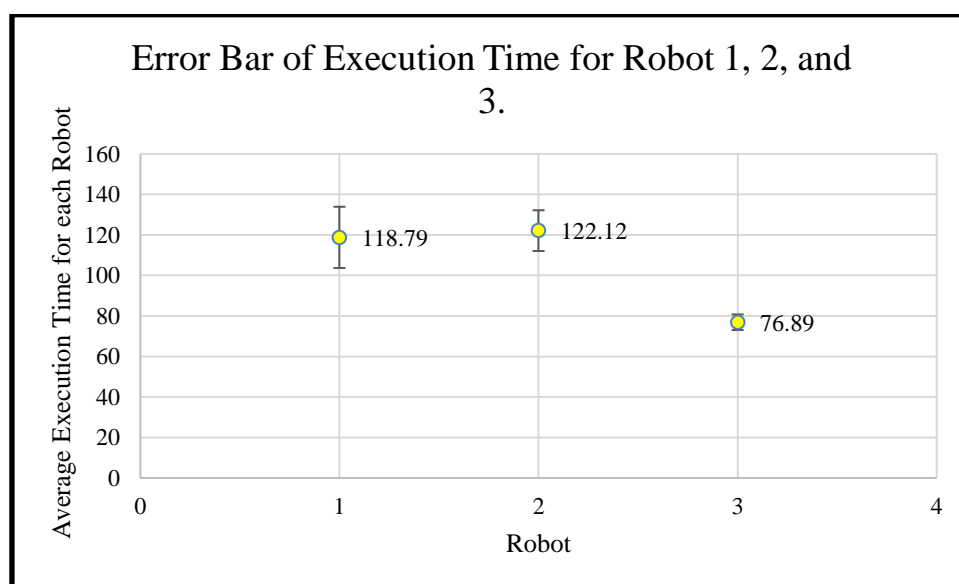


Figure 4.35: Error Bar of EPSO Execution Time for Robot 1, 2, and 3.

### 4.3.3 Comparison between MPSO and EPSO in Environment 3

Figure 4.36 displays the warehouse layout of Environment 3. In Environment 3, there is a T-shaped loading station, a rectangular unloading station, five rectangular shelves, and a circular shelf at the center. They are treated as static obstacles for all three robots and the pathways generated should not intersect the obstacles' position. Environment 3 has the most obstacles which are eight static obstacles in the environment, whereas Environment 1 has six obstacles

and Environment 2 has seven obstacles. From Environment 3, it can be observed that Robot 1 is situated at the right side of the map and it needs to cross the map to reach its target point beside the loading station at the bottom left corner. Robot 2 has to travel from the left side to the target point beside the unloading station, which is situated at the bottom right corner of the map. Robot 3 is situated at the upper left side of the map and will only need to travel straight towards its target point at the upper right corner of the map by passing through several shelves.

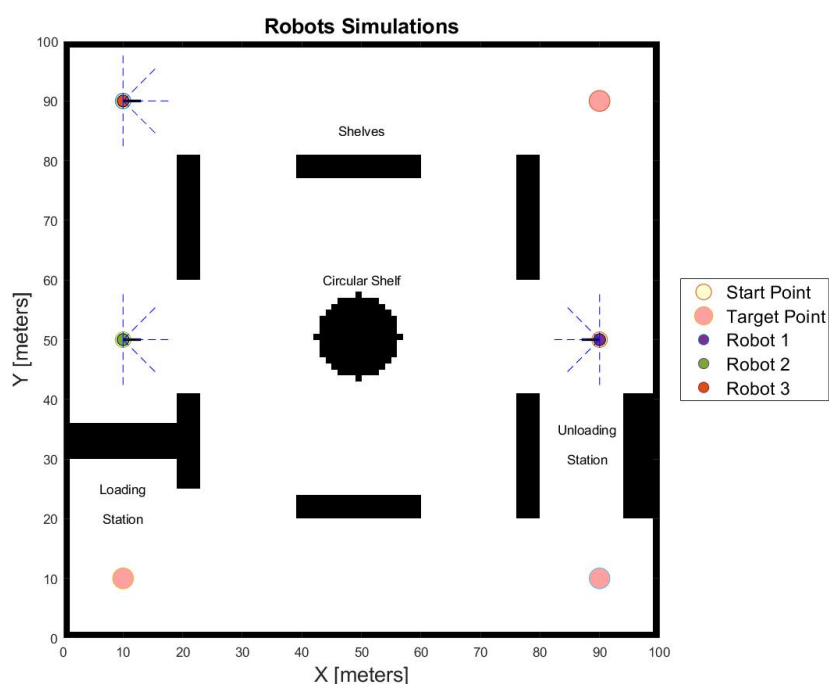


Figure 4.36: Environment 3.

#### 4.3.3.1 Result of MPSO Algorithm

The modified particle swarm optimization algorithm (MPSO) only implements path planning for all three robots in Environment 3 without considering the smoothness of pathways compared to EPSO. Figure 4.37 displays the pathway generated for Robot 1 through the MPSO algorithm. It has a total of 22 waypoints, which are represented by the red cross along the pathway. These waypoints are generated by the searches of particles of the MPSO algorithm. Out of the 22 waypoints, there are 13 turn points, which are labeled as blue circles along the path. It can be observed that for MPSO, the pathway generated has more twists and turns and is not beneficial for the robot's

movement. From Figure 4.38, Robot 2's pathway can be observed to have 23 waypoints, represented by red cross along the pathway. Out of the waypoints, there are 17 turn points for Robot 2. For Robot 3, its pathway can be seen in Figure 4.39. Pathway of Robot 3 has 17 waypoints and out of them, 10 turn points are recorded as represented by blue circles label.

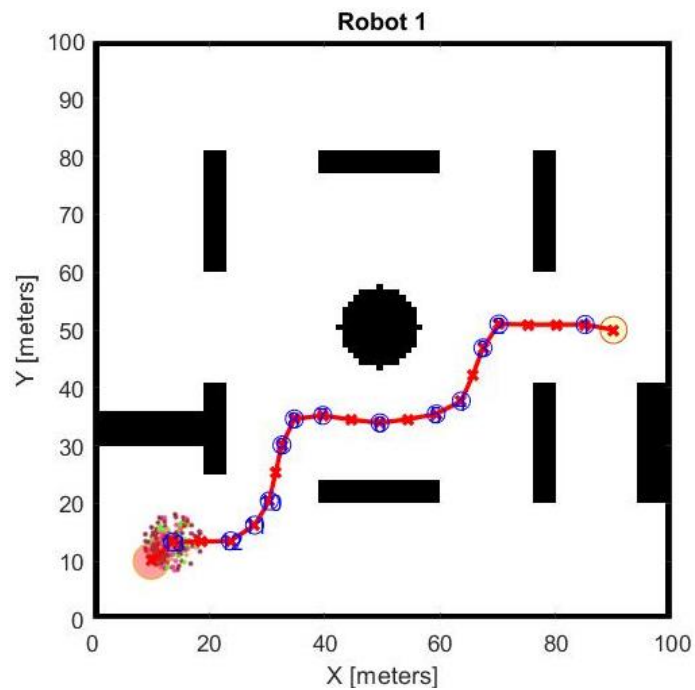


Figure 4.37: Pathway Generated for Robot 1 of MPSO.

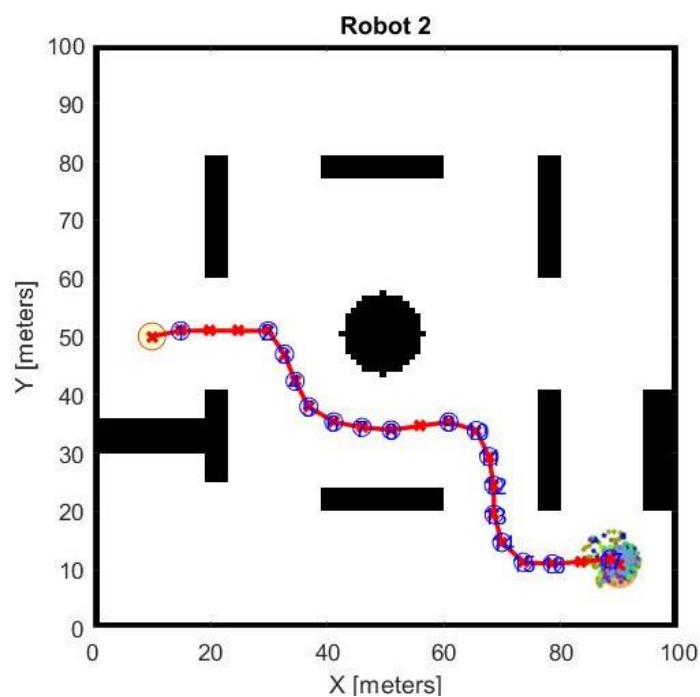


Figure 4.38: Pathway Generated for Robot 2 of MPSO.

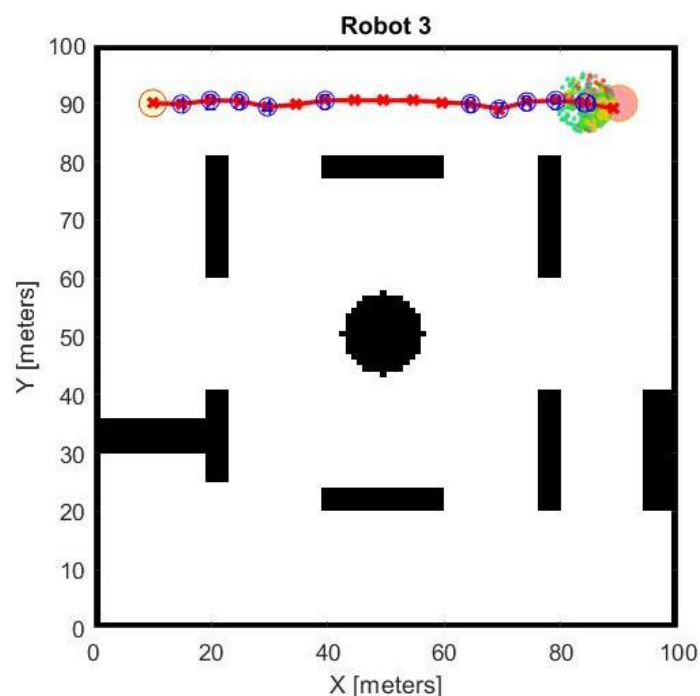


Figure 4.39: Pathway Generated for Robot 3 of MPSO.

Next, Figure 4.40 displays different pathways generated for all three robots according to MPSO in Environment 3. For Robot 1, the pathway is represented by a purple dotted line whereas Robot 2's pathway is represented by the dotted green line. It can be observed that there is an intersection of

Robot 1 and Robot 2's pathways near the circular shelf. When both robots enter each other's sensor range, the obstacle avoidance algorithm will be activated and both robots turn their headings toward the other direction in search of a direction without obstacles. Then, as Robot 1 can continue with its pathway, Robot 2 will also resume its original pathway towards their respective target points. For Robot 3, its pathway is represented by a red dotted line and it does not face any obstacles along its pathway. Then, Figure 4.41 shows the plotting of MPSO waypoints for all three robots. This gives a clearer view of each waypoint traveled by each robot.

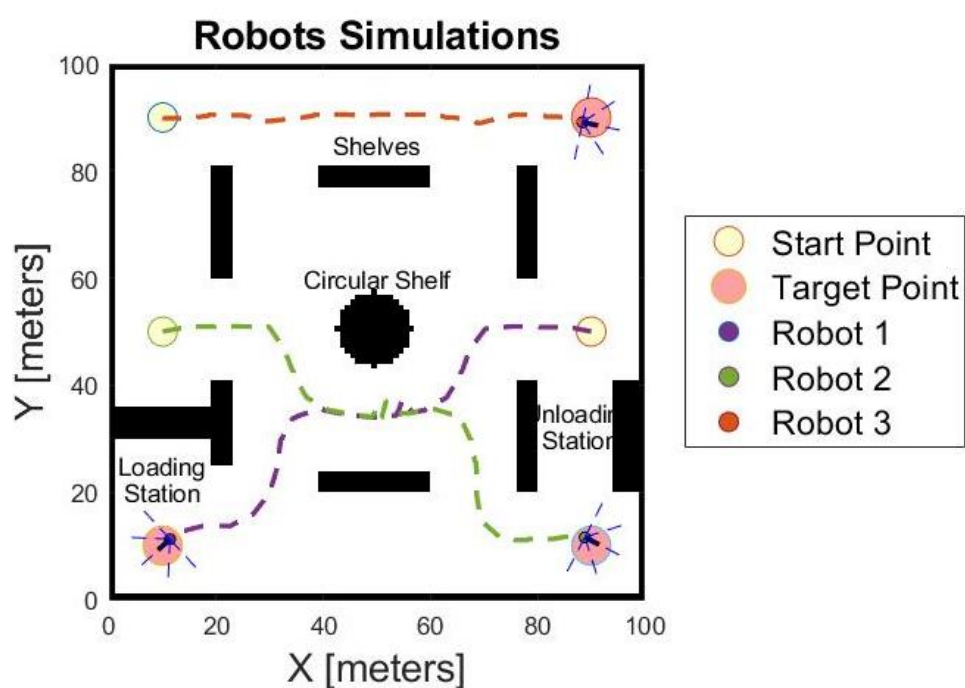


Figure 4.40: Simulation of Robots Travelling According to MPSO Pathway.

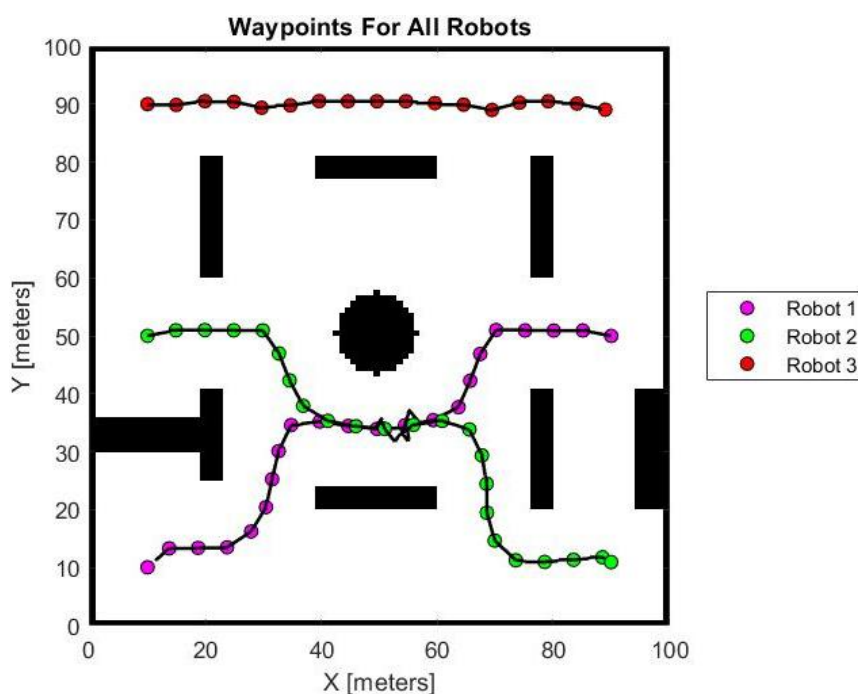


Figure 4.41: Plotting of MPSO Waypoints of all Robots.

Furthermore, Figure 4.42 showcases the graph of global best fitness concerning the iterations of the MPSO algorithm. As the number of iterations increases, the total global best fitness for all three robots decreases. At iteration one, the total global best fitness is 291.11 and the value decreases to 32.70 at iteration 23. This indicates that at iteration 23, all three robots completed their optimization process and reached their respective destinations.

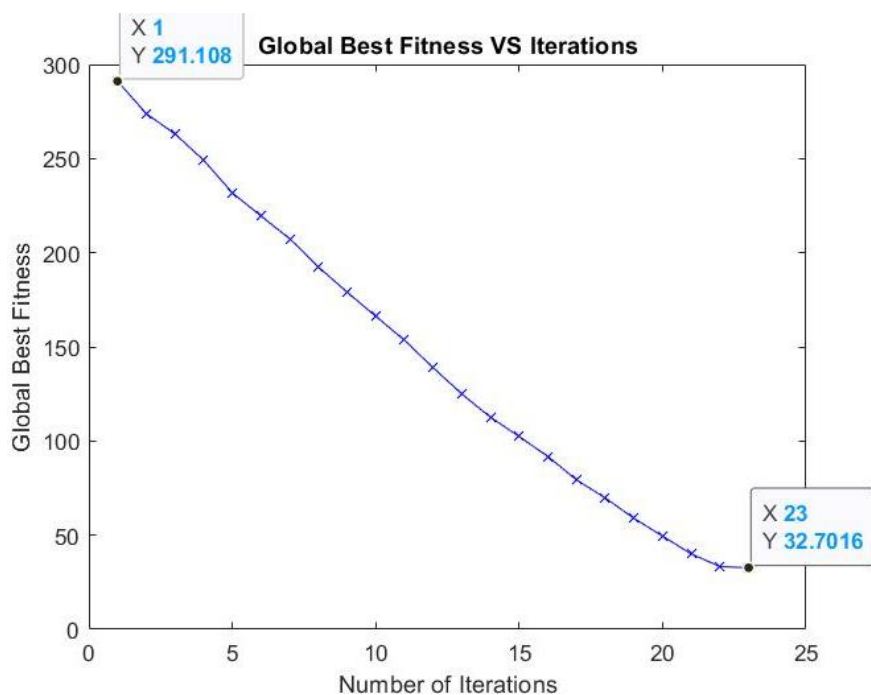


Figure 4.42: Graph of Global Best Fitness Vs. Iterations of MPSO Algorithm.

#### 4.3.3.2 Result of EPSO Algorithm

The enhanced particle swarm optimization algorithm (EPSO) implemented the Bezier curve trajectory smoothing algorithm after the path planning algorithm was completed. The waypoints generated from the path planning algorithm are input into the trajectory smoothing algorithm to generate control points that can be connected to become a smooth pathway for robots to travel. Figure 4.43 shows the smooth pathway generated, as represented by a blue solid line, for Robot 1 to travel from its start point to the target points. It can be observed that the blue pathway presents greater smoothness and fewer twists and turns as compared to the red pathway generated from the path planning algorithm. As stated earlier, the control points generated are set to be 100, and out of all control points, there are only two turn points, labeled by the red circle, present along the blue pathway of Robot 1. Next, Figure 4.44 displays the smooth pathway, as represented by a blue solid line, generated for Robot 2 from its start point to the target point. There is only one turn point present along the blue pathway of Robot 2. Moreover, Figure 4.45 showcases the pathway generated for Robot 3. As Robot 3 only needs to travel straight towards its endpoint, the twist and turn of the pathway are not obvious. However, it can be observed that the blue pathway possesses greater smoothness than the red

pathway generated from the path planning algorithm. There is no turn point present along the blue pathway generated from the Bezier curve trajectory smoothing algorithm.

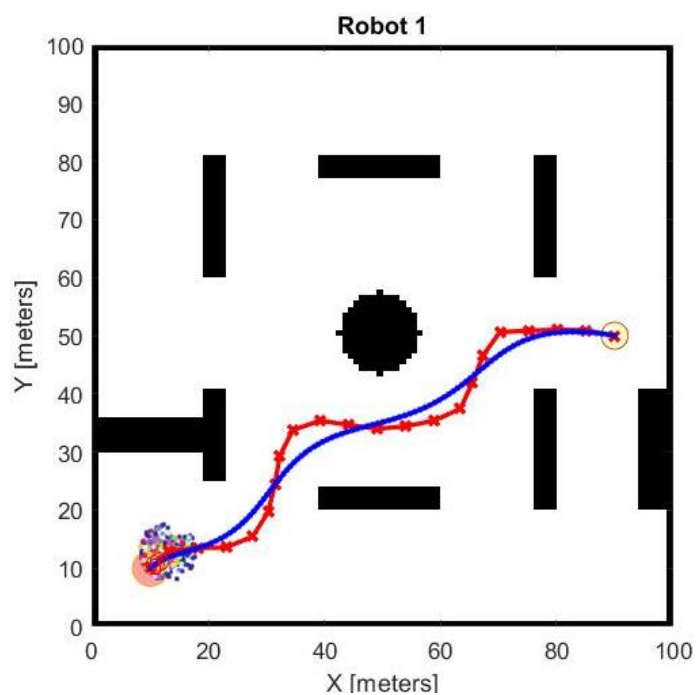


Figure 4.43: Pathways Generated for Robot 1 of EPSO.

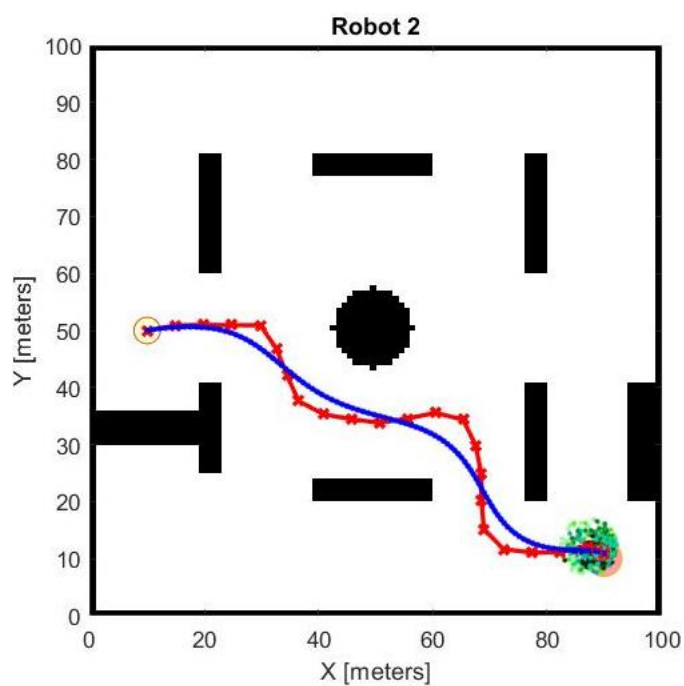


Figure 4.44: Pathways Generated for Robot 2 of EPSO.



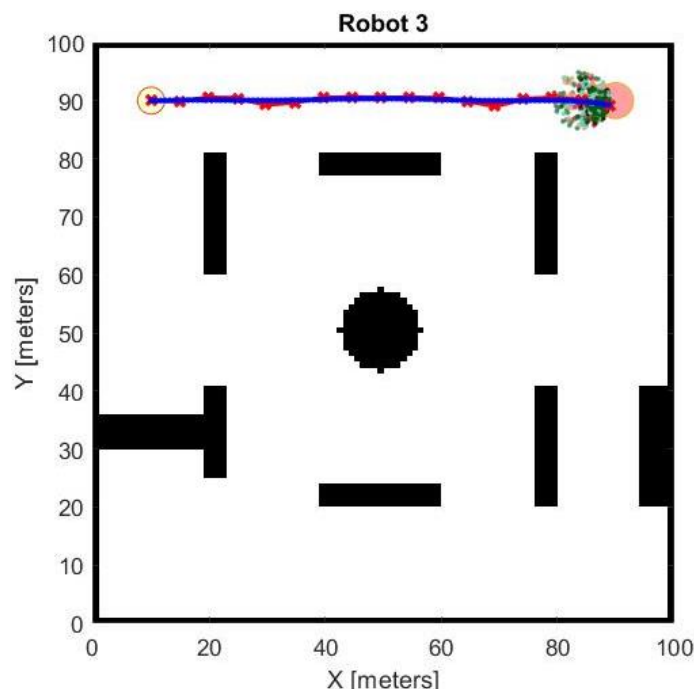


Figure 4.45: Pathways Generated for Robot 3 of EPSO.

Next, Figure 4.46 displays the simulation of robots traveling according to the EPSO pathway. Robot 1 travels from the right side of the map to the bottom left corner of the map beside the loading station. The pathway of Robot 1 is represented by a purple dotted line. For Robot 2, its pathway is indicated by a green dotted line and it travels from the left side to the bottom right corner of the map beside the unloading station. As for Robot 3, it travels straight towards its target point at the upper right corner of the map, in which its pathway is labeled as the red dotted line. It can be observed that the EPSO pathways followed by the three robots during simulation are smoother than the MPSO pathway. Then, Figure 4.47 displays the EPSO waypoints of all robots to provide a clearer insight into the 100 control points generated by the Bezier curve trajectory smoothing algorithm.

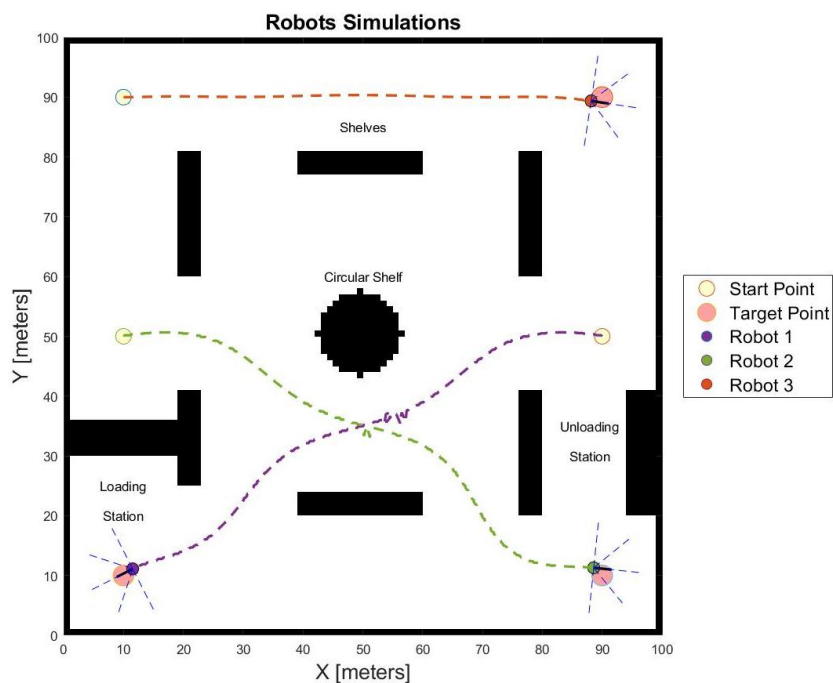


Figure 4.46: Simulation of Robots Travelling According to EPSO Pathway.

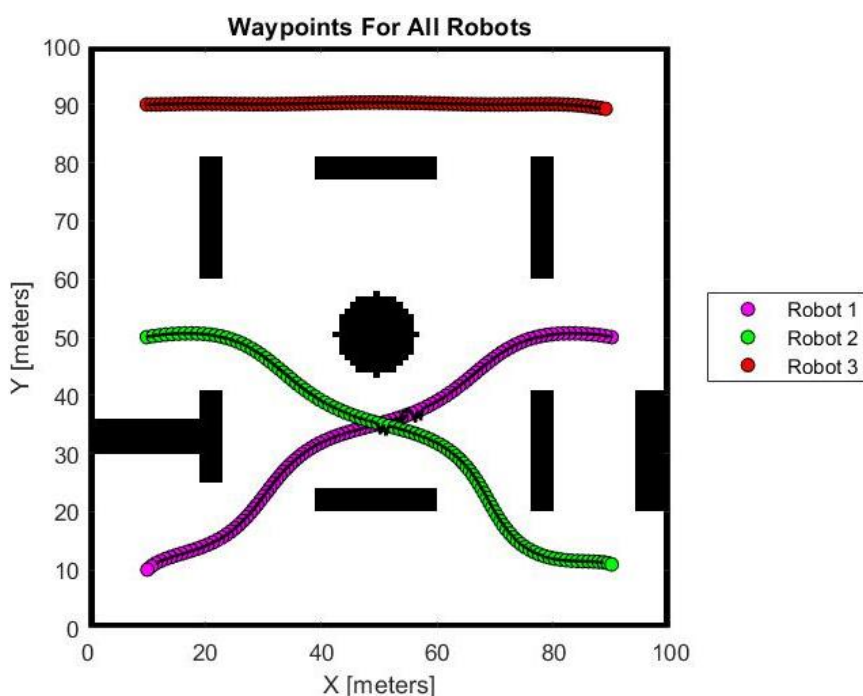


Figure 4.47: Plotting of EPSO Waypoints of all Robots.

Furthermore, Figure 4.48 showcases the relationship between global best fitness concerning the number of iterations of the EPSO algorithm. At the first iteration, all robots are still in their start point so their total global best fitness value is 291.11. As the iteration increases, the total global best fitness

of all robots decreases. At iteration 23, all three robots have successfully arrived at their respective target points and the total global best fitness value decreases to 32.74.

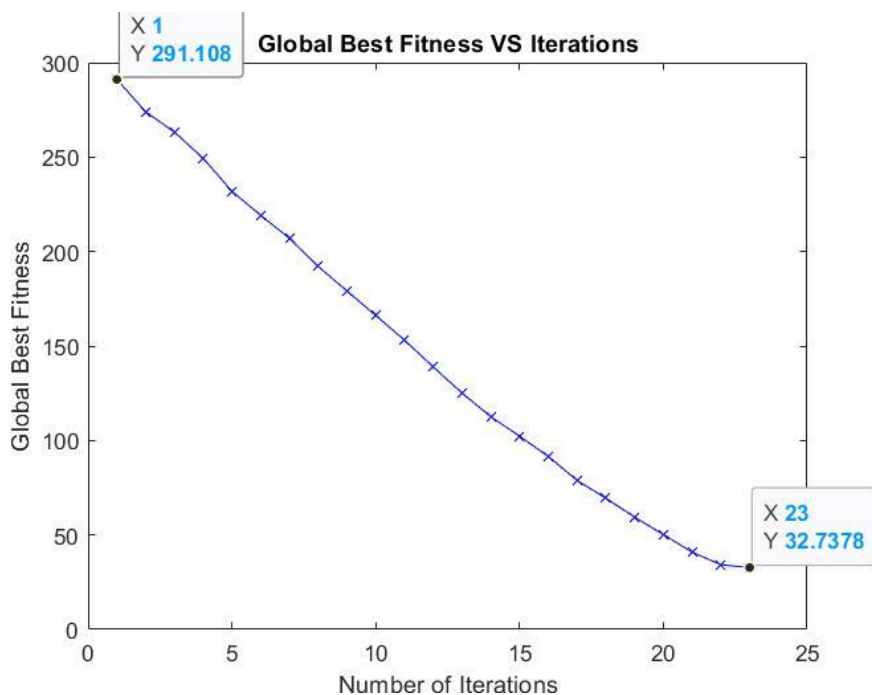


Figure 4.48: Graph of Global Best Fitness Vs. Iterations of EPSO Algorithm.

To provide a clear comparison of the result of MPSO and EPSO, Table 4.12 has tabulated the average path length, average execution time, and number of turn points of MPSO and EPSO for all three robots in Environment 3. It can be observed that the average path length of EPSO is shorter than MPSO for all three robots. Next, the average execution time of EPSO is also lower than MPSO. The number of turn points of robots that follow the EPSO pathway is also significantly reduced as compared to the number of turn points in the MPSO pathway. The simulation is repeated a total of five times to obtain the average result for path length and execution time for all three robots. Next, Table 4.13 showcases the result of MPSO for each simulation conducted in Environment 3. The longest path lengths achieved for Robot 1, Robot 2, and Robot 3 are 120.83 m, 117.72m, and 79.93m respectively. The shortest path lengths recorded for Robot 1, Robot 2, and Robot 3 are 110.73m, 111.04m, and 79.74m respectively. Then, the greatest execution time recorded is 128.64s, 140.73s, and 95.28s with the shortest execution time recorded to be 117.48s,

117.78s, and 67.81s respectively for Robot 1, Robot 2, and Robot 3. Moreover, Table 4.14 displays the result of EPSO for each simulation conducted in Environment 3. The longest path lengths recorded are 107.47m, 106.16m, and 79.14m while the shortest path lengths recorded are 100.50m, 99.51m, and 78.33m respectively for Robot 1, Robot 2, and Robot 3. As for the execution time, the highest execution time recorded is 119.23s, 111.56s, and 78.38s with the lowest execution time recorded to be 102.19s, 103.32s, and 69.05s respectively for Robot 1, Robot 2, and Robot 3. Apart from that, Figure 4.49 and Figure 4.50 showcase the error bars of path length and execution time for Robot 1, Robot 2, and Robot 3 of the EPSO algorithm based on standard deviation. From Figure 4.49, it can be observed that the error bars for all three robots only have a short length, which implies that the path lengths obtained through repeated simulations five times are all tightly clustered around the average value of path length. Hence, it has a low standard deviation. Then, from Figure 4.50, it can be observed that the error bar of execution time for Robot 1 has a longer length than for Robot 2 and Robot 3. This is because, for Robot 1, the highest execution time recorded is 119.23s while the lowest execution time recorded is 102.19. It has the highest standard deviation of 6.56 compared to 4.23 for Robot 2 and 4.09 for Robot 3. This implies that the execution time data is more spread out for Robot 1.

Table 4.12: Result of MPSO and EPSO of Environment 3.

	Average Path Length (meters)		Average Execution Time (s)		Number of Turn Points	
	MPSO	EPSO	MPSO	EPSO	MPSO	EPSO
Robot 1	115.18	103.98	122.87	107.97	13	2
Robot 2	114.06	102.32	132.88	106.50	17	1
Robot 3	79.85	78.69	80.44	73.30	10	0

Table 4.13: Result of MPSO for each Simulation in Environment 3.

MPSO	Robot 1		Robot 2		Robot 3	
	Path Length	Execution Time	Path Length	Execution Time	Path Length	Execution Time
First Simulation	112.24	128.64	111.04	140.73	79.93	79.53
Second Simulation	120.83	123.82	117.72	125.90	79.84	71.72
Third Simulation	110.73	124.41	112.32	139.46	79.90	95.28
Fourth Simulation	111.26	120.00	111.52	140.52	79.74	87.84
Fifth Simulation	120.83	117.48	117.72	117.78	79.84	67.81

Table 4.14: Result of EPSO for each Simulation in Environment 3.

EPSO	Robot 1		Robot 2		Robot 3	
	Path Length	Execution Time	Path Length	Execution Time	Path Length	Execution Time
First Simulation	106.75	119.23	102.39	111.56	78.33	78.38
Second Simulation	100.79	102.19	100.69	103.38	78.69	69.05
Third Simulation	104.38	106.32	102.84	103.32	79.14	70.97
Fourth Simulation	107.47	106.95	99.51	103.56	78.87	71.19
Fifth Simulation	100.50	105.14	106.16	110.70	78.43	76.93

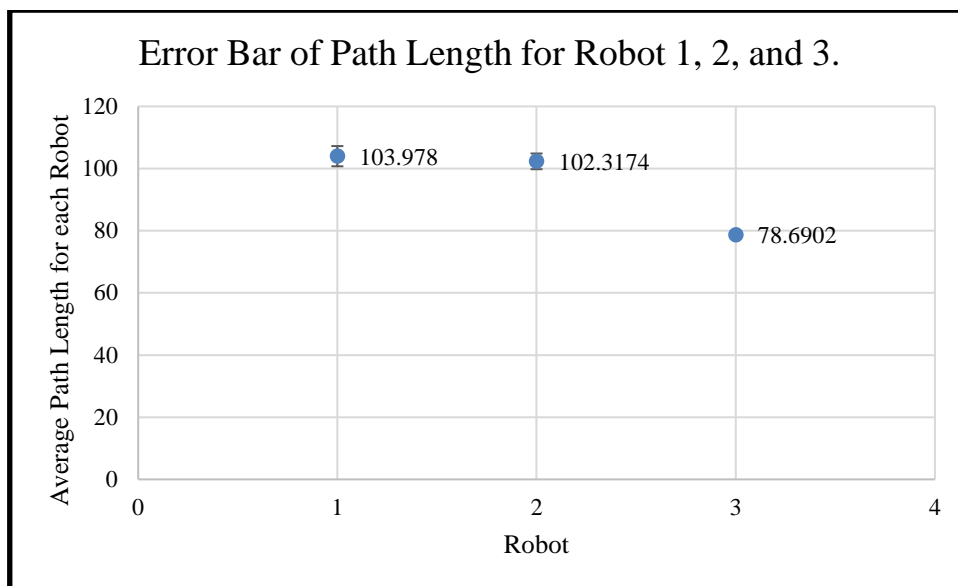


Figure 4.49: Error Bar of EPSO Path Length for Robot 1, 2, and 3.

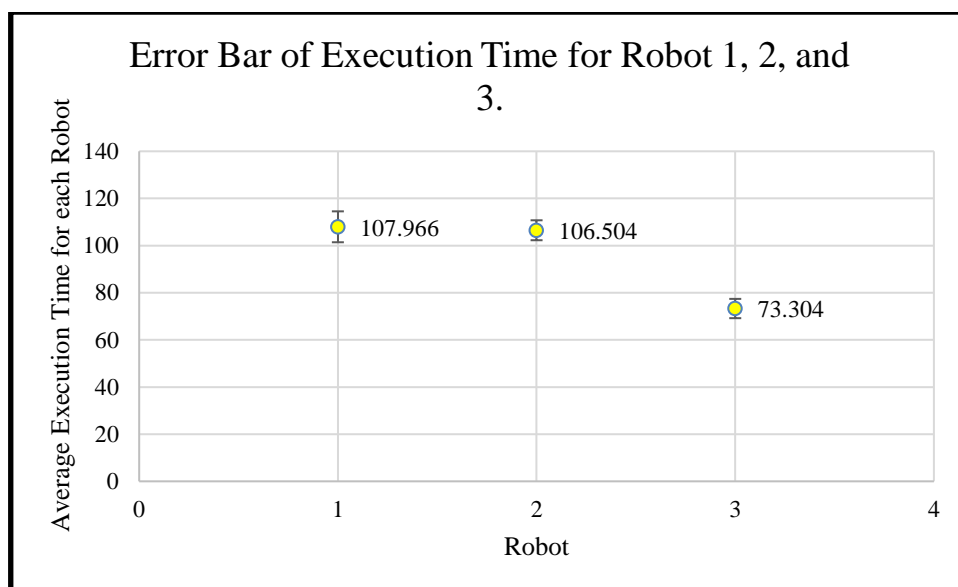


Figure 4.50: Error Bar of EPSO Execution Time for Robot 1, 2, and 3.

#### 4.4 Summary

The tuning of EPSO parameters showcases the impact of each parameter on the result of the EPSO algorithm. For the first parameter, careful calibration of swarm size is important to balance between convergence speed and optimality of solution, and hence, a swarm size of 80 is chosen for simulation. For the second parameter, the number of control points is set to 100 to generate a smooth trajectory with the lowest turn points for each robot. For the third parameter, a linearly decreasing inertia weight approach is implemented as a

higher inertia weight promotes the exploitation of particles then a lower inertia weight promotes convergence toward the optimal solution with lower constraints from previous velocities. Next, the cognitive acceleration coefficient is set to decrease and the social acceleration coefficient is set to increase as iterations increase. This is because a decrease in the cognitive acceleration coefficient will reduce the reliance of particles on their personal best solution and an increase in the social acceleration coefficient will encourage convergence towards the global best position.

Furthermore, the results of MPSO and EPSO algorithms are evaluated and compared based on path length, execution time, and number of turn points that exist along the pathway. From each aspect, it can be concluded that the EPSO algorithm outperforms the MPSO algorithm as it can generate pathways with shorter path lengths, travel using shorter execution time, and a lower number of turn points exist on the EPSO pathway for every robot in all three warehouse environments.

## CHAPTER 5

### CONCLUSIONS AND RECOMMENDATIONS

#### 5.1 Conclusions

In conclusion, this project aims to provide an enhanced particle swarm optimization (EPSO) algorithm as a possible solution for multi-robot path planning optimised through smooth trajectory generation via the Bezier curve. There are three objectives in this project. The first one is to review on existing multi-robot path planning algorithm. There are five multi-robot path planning algorithms discussed including artificial neural network (ANN), genetic algorithm (GA), particle swarm optimization (PSO), ant colony optimization (ACO), and artificial fish swarm algorithm (AFSA). PSO is chosen compared to the other four algorithms due to easy implementation of the algorithm, fast convergence speed due to exploration on both global and local scales, and the ability to generate robust results by parameter tuning.

Then, the second objective is to develop an enhanced PSO (EPSO) algorithm for smooth trajectory generation of a multi-robot system. The methodology of EPSO includes initialization, optimization process through iterations, and path generation with smooth trajectory. Moreover, the third objective is to evaluate the performance of EPSO through simulation. Before the evaluation of performance, parameter tuning is carried out to tune five parameters, including swarm size, number of control points of the Bezier curve algorithm, inertia weight, and cognitive and social acceleration coefficients. A swarm size of 80, control points of 100, and a linearly decreasing inertia weight approach from 0.95 to 0.4 as iterations increase are chosen as the required parameter values to generate optimal results. As iteration progresses, the cognitive acceleration coefficient is set to decrease and the social acceleration coefficient is set to increase.

To evaluate the performance of the EPSO algorithm, the result of the EPSO algorithm is compared with the MPSO algorithm based on path length, execution time, and number of turn points of robots. The simulations for both EPSO and MPSO are conducted repeatedly five times to obtain average results



for each of the three environments. It can be found that EPSO outperforms MPSO with a shorter path length, shorter execution time, and lower number of turn points along the pathways of all robots. Overall, this project has successfully proposed a new EPSO algorithm that integrates the Bezier curve trajectory smoothing algorithm with the PSO path planning algorithm. The EPSO algorithm implemented in multi-robot path planning can efficiently shorten path length, and execution time, and decrease the number of turn points of pathways generated. All objectives of this project have been accomplished.

## **5.2 Recommendations for Future Work**

In this project, two recommendations can be made for further enhancement of the algorithm. The first one is to achieve smooth path planning of robots' pathways as PSO iterations progress. Smooth path planning refers to the implementation of a smooth function in the PSO algorithm to set trajectory constraints based on control points. This enables particles to generate waypoints that can directly construct a smooth path as iteration terminates. Setting the trajectory constraint based on control points will limit the particles to choose waypoints that can generate smooth paths for robots.

The second improvement that can be made is to implement the EPSO algorithm in real-life environment. Currently, the EPSO algorithm is only evaluated and tested using simulation, and its effectiveness in real-life situations remains unknown. A Robotics Operating System (ROS) can be used to implement the EPSO algorithm in real-life situations to evaluate the effectiveness of the algorithm in reducing path length, execution time, and number of turn points of robots.

In a nutshell, the recommendation for future work includes adding a smooth function in the PSO algorithm to achieve smooth path planning as iterations terminate, and to implement the EPSO algorithm in real life environment using ROS.

## REFERENCES

- Abujabal, N., Fareh, R., Sinan, S., Baziyad, M. and Bettayeb, M., 2023a. A comprehensive review of the latest path planning developments for multi-robot formation systems. *Robotica*, 41(7), pp.2079–2104. <https://doi.org/10.1017/s0263574723000322>.
- Abujabal, N., Fareh, R., Sinan, S., Baziyad, M. and Bettayeb, M., 2023b. A comprehensive review of the latest path planning developments for multi-robot formation systems. *Robotica*, [online] 41(7), pp.2079–2104. <https://doi.org/10.1017/S0263574723000322>.
- Bilbeisi Ghaith, Al-Madi, N. and Awad, F., 2015. PSO-AG: A Multi-Robot Path Planning and Obstacle Avoidance Algorithm. In: *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*. [online] Amman, Jordan: IEEE. pp.1–6. <https://doi.org/https://doi.org/10.1109/AEECT.2015.7360565>.
- Dian, S., Zhong, J., Guo, B., Liu, J. and Guo, R., 2022. A smooth path planning method for mobile robot using a BES-incorporated modified QPSO algorithm. *Expert Systems with Applications*, 208, pp.1–15. <https://doi.org/10.1016/j.eswa.2022.118256>.
- Farinelli, A., Iocchi, L. and Nardi, D., 2004. Multirobot systems: A classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(5), pp.2015–2028. <https://doi.org/10.1109/TSMCB.2004.832155>.
- Gadhgadhi, A., Hachaichi, Y. and Zairi, H., 2022. Tuning PSO Parameters For the Path Planning Problem. In: *2022 IEEE Information Technologies and Smart Industrial Systems, ITSIS 2022*. Institute of Electrical and Electronics Engineers Inc. pp.1–6. <https://doi.org/10.1109/ITSIS56166.2022.10118408>.

Gautam, A. and Mohan, S., 2012. A Review of Research in Multi-Robot Systems. In: *2017 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*. IEEE. pp.1–5. <https://doi.org/10.1109/ICIInfS.2012.6304778>.

Guzmán, M.A. and Peña, C.A., 2013. *Algoritmos bioinspirados en la planeación off-line de trayectorias de robots seriales Bio-inspired algorithms in serial-robot path off-line planning*. <https://doi.org/10.14483/22484728.4390>.

Jianwei, M., Yang, L., Shaofei, Z. and Wang, L., 2020. Robot Path Planning Based on Genetic Algorithm Fused with Continuous Bezier Optimization. *Computational Intelligence and Neuroscience*, pp.1–10. <https://doi.org/10.1155/2020/9813040>.

Lee, J., Kang, B.Y. and Kim, D.W., 2013. Fast genetic algorithm for robot path planning. *Electronics Letters*, 49(23), pp.1449–1451. <https://doi.org/10.1049/el.2013.3143>.

Li, F.F., Du, Y. and Jia, K.J., 2022. Path planning and smoothing of mobile robot based on improved artificial fish swarm algorithm. *Scientific Reports*, 12(1), pp.1–16. <https://doi.org/10.1038/s41598-021-04506-y>.

Lin, S., Liu, A., Wang, J. and Kong, X., 2022. A Review of Path-Planning Approaches for Multiple Mobile Robots. *Machines*, 10(9), pp.1–27. <https://doi.org/10.3390/machines10090773>.

Madridano, Á., Al-Kaff, A., Martín, D. and de la Escalera, A., 2021a. Trajectory planning for multi-robot systems: Methods and applications. *Expert Systems with Applications*, 173, pp.1–14. <https://doi.org/10.1016/j.eswa.2021.114660>.

Madridano, Á., Al-Kaff, A., Martín, D. and de la Escalera, A., 2021b. *Trajectory planning for multi-robot systems: Methods and applications. Expert Systems with Applications*, <https://doi.org/10.1016/j.eswa.2021.114660>.

Mbemba, B., Chen, Y. and Shu, Y., 2022. Path Planning Based on Probabilistic Roadmap and Ant Colony Optimization. In: *5th International Conference on Intelligent Autonomous Systems, ICoIAS 2022*. Institute of Electrical and Electronics Engineers Inc. pp.102–108. <https://doi.org/10.1109/ICoIAS56028.2022.9931206>.

N. Darmanin, R. and K. Bugeja, M., 2017. A Review on Multi-Robot Systems Categorised by Application Domain. *2017 25th Mediterranean Conference on Control and Automation (MED)*, pp.1–6. <https://doi.org/10.1109/MED.2017.7984200>.

Paez, D., Romero, J.P., Noriega, B., Cardona, G.A. and Calderon, J.M., 2021. Distributed particle swarm optimization for multi-robot system in search and rescue operations. In: *IFAC-PapersOnLine*. Elsevier B.V. pp.1–6. <https://doi.org/10.1016/j.ifacol.2021.10.001>.

P.K Das, B.M. Sahoo, H.S. Behera and S Vashisht, 2016. An Improved Particle Swarm Optimization for Multi-Robot Path Planning. In: *2016 1st International Conference on Innovation and Challenges in Cyber Security (ICICCS 2016)*. IEEE. pp.1–10. <https://doi.org/10.1109/ICICCS.2016.7542324>.

Poy, Y.L., Darmaraju, S. and Kwan, B.H., 2023. Multi-robot Path Planning using Modified Particle Swarm Optimization. In: *2023 IEEE International Conference on Automatic Control and Intelligent Systems, I2CACIS 2023 - Proceedings*. Institute of Electrical and Electronics Engineers Inc. pp.225–230. <https://doi.org/10.1109/I2CACIS57635.2023.10193290>.

Rao, Dr.S.V.A., Kondaiah, Dr.K., Chandra, Dr.G.R. and Kumar, Dr.K.K., 2017. A Survey on Machine Learning: Concept, Algorithms and Applications. In: *International Conference on Innovative Research in Computer and Communication Engineering A Survey on Machine Learning: Concept, Algorithms and Applications*. SMEC. pp.1–9. <https://doi.org/10.15680/IJIRCCE.2017.0502001>.

Rao, P.U. and Sodhi, B., 2022. Collision-free Path Planning in Multi-vehicle Deployments - A Quantum Approach. In: *Proceedings - 2022 IEEE International Conference on Quantum Computing and Engineering, QCE 2022*. Institute of Electrical and Electronics Engineers Inc. pp.13–21. <https://doi.org/10.1109/QCE53715.2022.00019>.

Ravankar, A., Ravankar, A.A., Kobayashi, Y., Hoshino, Y. and Peng, C.C., 2018. Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges. *Sensors (Switzerland)*, 18(9), pp.1–30. <https://doi.org/10.3390/s18093170>.

Shuhua, L., Yantao, T. and Jinfang, L., 2004. Multi Mobile Robot Path Planning Based on Genetic Algorithm\*. In: *5th World Congress on intelligent Control and Automation*. HangZhou: IEEE. pp.1–4. <https://doi.org/10.1109/WCICA.2004.1342412>.

Sitong, Z. and Tianyi, Z., 2022. Mobile Robot Path Planning in 2D Space: A Survey. *Highlights in Science, Engineering and Technology AMMSAC*, 16, pp.1–11. <https://doi.org/10.54097/hset.v16i.2508>.

Tang, Z. and Ma, H., 2021. An overview of path planning algorithms. In: *IOP Conference Series: Earth and Environmental Science*. IOP Publishing Ltd. pp.1–11. <https://doi.org/10.1088/1755-1315/804/2/022024>.

Xu, L., Song, B. and Cao, M., 2021. A new approach to optimal smooth path planning of mobile robots with continuous-curvature constraint. *Systems Science and Control Engineering*, 9(1), pp.138–149. <https://doi.org/10.1080/21642583.2021.1880985>.

Yang, G.H., Lee, H. and Ryuh, Y., 2013. Development of a 3-DOF Fish Robot ICHTHUS V5'. *Studies in Computational Intelligence*, 466, pp.225–234. [https://doi.org/10.1007/978-3-642-35485-4\\_18](https://doi.org/10.1007/978-3-642-35485-4_18).

Yi-Wen, C. and Wei-Yu, C., 2015. Optimal Robot Path Planning System by Using a Neural Network-Based Approach. In: Y.-W. Chen and W.-Y. Chiu, eds. *2015 International Automatic Control Conference (CACCS)*. Yilan, Taiwan: IEEE. pp.1–6. <https://doi.org/10.1109/CACCS.2015.7378370>.

## APPENDICES

## Appendix A: Matlab Codes

```

function Robots = bezierCurve(Robots, NO_OF_ROBOTS, num_points, t_values)
    % Check if t_values are provided, otherwise generate them
    if nargin < 4
        t_values = linspace(0, 1, num_points);
    end

    % Initialize waypoints_all cell array
    waypoints_all = cell(1, NO_OF_ROBOTS);

    % Loop over robots to generate or provide matrices of waypoints
    for j = 1: NO_OF_ROBOTS
        waypoints_all{j} = [Robots(j).waypoints(:, 1), Robots(j).waypoints(:, 2)];
    end

    % Initialize the Bezier curve points for each robot
    bezier_curve_all = zeros(2, num_points, NO_OF_ROBOTS);

    % Calculate the Bernstein basis functions for each robot
    basis_functions_all = cell(1, NO_OF_ROBOTS);
    for j = 1: NO_OF_ROBOTS % Loop over robots
        num_waypoints = size(waypoints_all{j}, 1);
        basis_functions = zeros(num_waypoints, num_points);
        for i = 1:num_waypoints
            basis_functions(i, :) = nchoosek(num_waypoints-1, i-1) * t_values.^(i-1) .* (1-t_values).^(num_waypoints-i);
        end
        basis_functions_all{j} = basis_functions;
    end

    % Calculate the Bezier curve points for each robot
    smoothed_waypoints_all = cell(1, NO_OF_ROBOTS);

    for j = 1: NO_OF_ROBOTS % Loop over robots
        num_waypoints = size(waypoints_all{j}, 1);
        for i = 1:num_points
            bezier_curve_all(:, i, j) = sum(waypoints_all{j} .* basis_functions_all{j}(:, i), 1)';

            % Store Bezier curve points directly in the Robots structure as double
            Robots(j).bezier_curve = double(bezier_curve_all(:, :, j)');

        end
    end

end

end

end

```

CodeA-1: Function of Bezier Curve Trajectory Smoothing Algorithm Code.

```

% Define the number of points on the Bezier curve and parameter values
num_points = 100;
t_values = linspace(0, 1, num_points);

% Call the bezierCurve function to calculate and store Bezier curve points
Robots = bezierCurve(Robots, NO_OF_ROBOTS, num_points, t_values);

% Initialize array to store the number of turns for each robot
numTurnsPerRobot = zeros(NO_OF_ROBOTS, 1);

% Plot Bezier curves and turns
for j = 1:NO_OF_ROBOTS
    figure(j)
    hold all;

    % Plot Bezier curve points as a solid blue line
    bezier_curve = Robots(j).bezier_curve;
    x_curve = bezier_curve(:, 1); % Extract x coordinates from the first column
    y_curve = bezier_curve(:, 2); % Extract y coordinates from the second column
    plot(x_curve, y_curve, 'b-', 'LineWidth', 2);

    % Plot Bezier curve points
    for i = 1:num_points
        plot(bezier_curve(i, 1), bezier_curve(i, 2), 'b.', 'MarkerSize', 8);
    end

    % Plot turns on the Bezier curve as yellow circles and count the number of turns
    threshold_degree = 5; % Start with a small threshold_degree
    threshold = threshold_degree * pi / 180; % Convert threshold to radians
    smoothedPath = Robots(j).bezier_curve; % Get the smoothed path for the current robot
    numPoints = size(smoothedPath, 1);

    for i = 2:numPoints-1
        % Calculate vectors representing consecutive line segments
        vector1 = smoothedPath(i-1,:) - smoothedPath(i,:);
        vector2 = smoothedPath(i,:) - smoothedPath(i+1,:);

        % Calculate the angle between consecutive line segments using dot product
        angle = acos(dot(vector1, vector2) / (norm(vector1) * norm(vector2)));

        % If angle is greater than the threshold, plot the turn point as a red circle and increment the turn count
        if angle > threshold
            plot(smoothedPath(i, 1), smoothedPath(i, 2), 'ro', 'MarkerSize', 8);
            numTurnsPerRobot(j) = numTurnsPerRobot(j) + 1;

            % Add label at the turn point
            text(...
                smoothedPath(i, 1), ...
                smoothedPath(i, 2), ...
                num2str(numTurnsPerRobot(j)), ...
                'Color', 'red', ...
                'FontSize', 10, ...
                'HorizontalAlignment', 'center', ...
                'VerticalAlignment', 'bottom');
        end
    end

    % Display the number of turns for the smoothed path
    disp(['Number of turns for Robot ', num2str(j), ' (Smoothed Path): ', num2str(numTurnsPerRobot(j))]);

    title(['Robot ', num2str(j)]);
    xlabel('X [meters]');
    ylabel('Y [meters]');
    hold off;
end
end

```

CodeA-2: Bezier Curve Trajectory Smoothing Algorithm Code in Main EPSO Program.



```
function [map,obstacles] = Environment (value)

if value == 1
    map_1 = binaryOccupancyMap(100, 100, 1, "grid");
    obs = zeros(100, 100);

    % Define walls
    obs(1, :) = 1; % Top wall
    obs(end, :) = 1; % Bottom wall
    obs(:, 1) = 1; % Left wall
    obs(:, end) = 1; % Right wall

    % Place different shaped shelves
    % Shelf 1 (Rectangular)
    obs(20:40, 30:35) = 1;

    % Shelf 2 (L-shaped)
    obs(20:40, 60:65) = 1;
    obs(60:80, 60:65) = 1;
    obs(20:25, 60:75) = 1;

    % Unloading station
    obs(20:40, 95:end) = 1;
    obs(60:80, 95:end) = 1;

    % Loading station
    obs(65:75, 1:20) = 1;
    obs(60:80, 20:25) = 1;

    setOccupancy(map_1, [1 1], obs, "grid")
    map = map_1;
    occMatrix = checkOccupancy(map_1);
    occMatrix = rot90(occMatrix, 3);
    [rol, col] = find(occMatrix == 1);
    obstacles = [rol, col];
```

CodeA-3a: Environment Generation Code.

```

elseif value == 2

    map_2 = binaryOccupancyMap(100, 100, 1, "grid");

    obs = zeros(100, 100);

    % Define walls
    obs(1, :) = 1; % Top wall
    obs(end, :) = 1; % Bottom wall
    obs(:, 1) = 1; % Left wall
    obs(:, end) = 1; % Right wall

    % Loading station
    obs(60:80, 10:15) = 1;
    obs(65:75, 1:10) = 1;

    % Unloading station
    obs(20:40, 85:90) = 1;
    obs(25:35, 90:end) = 1;

    % Shelf 1 (L-shaped)
    obs(20:40, 20:25) = 1;
    obs(20:25, 25:40) = 1;

    % Shelf 2 (L-shaped)
    obs(60:80, 95:end) = 1;
    obs(75:80, 85:95) = 1;

    % Shelf 3 (Circular at the center)
    [X, Y] = meshgrid(1:100, 1:100); % Define X and Y for the circular shelf at the center
    centerX_shelf5 = 50;
    centerY_shelf5 = 50;
    radius_shelf5 = 5;
    obs((X - centerX_shelf5).^2 + (Y - centerY_shelf5).^2 <= radius_shelf5^2) = 1;

    % Shelf 4 (square)

    obs(20:30, 60:70) = 1;

    %Shelf 5 (rectangular)
    obs(80:85, 35:65) = 1;
    obs(75:80, 35:40) = 1;
    obs(75:80, 60:65) = 1;

    % Update the occupancy map
    setOccupancy(map_2, [1 1], obs, "grid")

    % Display the occupancy map
    map = map_2;
    occMatrix = checkOccupancy(map_2);
    occMatrix = rot90(occMatrix, 3);
    [rol, col] = find(occMatrix == 1);
    obstacles = [rol, col];

```

CodeA-3b: Environment Generation Code.

```

elseif value == 3

    map_3 = binaryOccupancyMap(100, 100, 1, "grid");
    obs = zeros(100, 100);

    % Define walls
    obs(1, :) = 1; % Top wall
    obs(end, :) = 1; % Bottom wall
    obs(:, 1) = 1; % Left wall
    obs(:, end) = 1; % Right wall

    %Loading station
    obs(60:75, 20:23) = 1;
    obs(65:70, 1:20) = 1;

    %Shelves
    obs(20:40, 20:23) = 1;
    obs(20:40, 77:80) = 1;
    obs(60:80, 77:80) = 1;
    obs(77:80, 40:60) = 1;
    obs(20:23, 40:60) = 1;

    %Unloading Station
    obs(60:80, 95:end) = 1;

    % Round shelf
    radius = 7;

    center_x = 50;
    center_y = 50;

    % Iterate over the map and set occupancy based on distance from center
    for i = 1:100
        for j = 1:100
            distance = sqrt((i - center_x)^2 + (j - center_y)^2);
            if distance <= radius
                obs(i, j) = 1;
            end
        end
    end

    setOccupancy(map_3, [1 1], obs, "grid")
    map = map_3;
    occMatrix = checkOccupancy(map_3);
    occMatrix = rot90(occMatrix, 3);
    [rol, col] = find(occMatrix == 1);
    obstacles = [rol, col];

end
end

```

CodeA-3c: Environment Generation Code.