Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Hantong Liu

A GQPSO Based Optimal Path Planner for Autonomous Underwater Vehicle with Waypoint Guidance System

Master's thesis in Marine Technology
Supervisor: Professor Martin Ludvigsen
June 2022

NTNU
Norwegian University of
Science and Technology

Hantong Liu

A GQPSO Based Optimal Path Planner for Autonomous Underwater Vehicle with Waypoint Guidance System

Master’s thesis in Marine Technology
Supervisor: Professor Martin Ludvigsen
June 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Abstract

Autonomous underwater vehicle (AUV) has become a popular and important way to do the exploration and exploitation of ocean areas where it's not possible for human to visit such as areas in higher depth. Most of the time, the operation environment of them exists various obstacles, including known static obstacles such as rock structure on seafloor, and unknown dynamic obstacles. Therefore, considering the safe operation condition, the path planning methods with collision avoidance could be seen as the core technology for AUV. This master's thesis presents a literature review introducing the main path planning methods with collision avoidance for AUV applications, including global path planning methods and local path planning methods. By going through all main state-of-the-art path planning methods, a discussion is provided to conclude their advantages and disadvantages. Moreover, further study in Particle Swarm Optimization (PSO) is carried out as well as its modified version Quantum-behaved Particle Swarm Optimization (QPSO) and Gaussian Quantum-behaved Particle Swarm Optimization (G-QPSO). By formulating the path planning problem, a PSO-based optimal path planner is designed using MATLAB. Also, its modified versions with QPSO and G-QPSO algorithms are also designed and deployed using MATLAB. 3 different scenarios are simulated to test the performance of the path planner. The simulation result shows that with well-tuned parameters, PSO-based path planner is qualified for path planning for AUV. But the tuning process is difficult and it is hard to avoid its convergence to local optimal solution. With the modified path planner, the tuning process becomes much easier. The performance and convergence ability to global optimum of QPSO and GQPSO-based path planner are better than that of PSO-based path planner. In conclusion, the GQPSO-based optimal path planner is a high-performance path planner for AUV.

Prefaces

This report is written based on the work of my master thesis in marine cybernetics during the spring of 2022 at the Norwegian University of Science and Technology (NTNU). The research work is based on the project thesis during the fall of 2021, which gives an overview of state-of-the-art path planning methods and their applications.

The main research topics of the master's thesis are the design and development of an optimal path planner based on particle swarm optimization method with waypoint guidance system.

It is assumed that the reader of this thesis retains basic knowledge within marine technology and control systems.

Hantong Liu

Trondheim, June, 2022

Acknowledgements

I have put great effort behind this master's thesis in this semester. In this process I have also gained lots of knowledge. Along the way, I have received great help and assistance from others. Here, I'd like to thank my supervisor, professor Martin Ludvigsen. I am thankful for his efforts in this whole academic year since he has answered my questions, given me ideas when facing challenges and provided feedback on my work. Furthermore, I would like to thank people who have helped me solve technical problems during the work.

Contents

Abstract . . . . **iii**
Prefaces . . . . **v**
Acknowledgements . . . . **vii**
Contents . . . . **ix**
Figures . . . . **xi**
Tables . . . . **xiii**
1 Introduction **1**
1.1 Background 1
1.1.1 Autonomous Underwater Vehicles (AUVs) 1
1.1.2 Autonomy 2
1.2 Motivation 3
1.3 Research Topic and Objectives 3
1.4 Scope and Limitations 3
1.5 Structure of the Thesis 4
2 Literature Review **7**
2.1 Main Path Planning Methods for AUV 7
2.1.1 Global Path-planning Methods 8
2.1.2 Local Path-planning Methods 11
2.2 Particle Swarm Optimization (PSO) for Path Planning 14
3 AUVs' Guidance System and Map Processing **17**
3.1 LAUV Thor 17
3.2 Guidance System with Constant Jerk 19
3.3 Map Processing 20
4 Optimization Theory **25**
4.1 Particle Swarm Optimization (PSO) 25
4.1.1 Classical Particle Swarm Optimization (PSO) Algorithm . . . 26
4.1.2 A Deeper Look into the Algorithm and its Meta-parameters . 28
4.2 Quantum-behaved Particle Swarm Optimization (QPSO) 31
4.3 Quantum-behaved Particle Swarm Optimization with Gaussian Mutation (G-QPSO) 34
5 Problem Formulation and Algorithm Design **37**
5.1 Problem Formulation 37
5.1.1 Preliminary Modelling for Optimal Path Planning Problem . 38
5.1.2 Simplification and Modification to Apply PSO Algorithm . . 41

5.1.3 Final Version of Modelling for Path Planning Problem 44
5.2 Algorithm Design 44
5.2.1 Straight-line Method 44
5.2.2 Design of Objective Functions 46
5.2.3 The PSO Algorithm for Path Planning Problem 48
5.2.4 The QPSO and G-QPSO Algorithm for Path Planning Problem 49
6 Results 51
6.1 Path Planning with PSO algorithm 51
6.1.1 2D Path Planning with Different c_1 and c_2 53
6.1.2 2D Path Planning with Different Definitions of Inertia Weight 55
6.1.3 2D Path Planning in Complex Environment with OBB-shape Obstacles 57
6.1.4 3D Path Planning with PSO Algorithm 58
6.2 Path Planning with QPSO Algorithm 62
6.2.1 2D Path Planning with Different c_1 and c_2 62
6.2.2 3D Path Planning with QPSO algorithm 65
6.3 Path Planning with G-QPSO Algorithm 67
6.3.1 2D Path Planning with G-QPSO algorithm 67
6.3.2 3D Path Planning with G-QPSO algorithm 70
7 Discussion 73
7.1 Path Planning with PSO Algorithm 73
7.1.1 Tuning of Cognitive and Social Coefficient c_1 and c_2 74
7.1.2 Choice of Definitions of Inertia Weight w 75
7.1.3 2D and 3D Path Planning with PSO Algorithm 75
7.2 Path Planning with QPSO Algorithm 76
7.2.1 Tuning for Cognitive and Social Coefficient c_1 and c_2 76
7.2.2 3D Path Planning with QPSO Algorithm 77
7.3 Path Planning with G-QPSO Algorithm 77
8 Conclusion 79
8.1 Conclusion 79
8.2 Future Work 80
Bibliography 81
A Matlab Code 87
A.1 Code of GQPSO-based optimal path planner 87
A.1.1 Main.m 87
A.1.2 CostFunction.m 90
A.2 Code of PSO-based optimal path planner 92
A.2.1 Main.m 92
A.3 Code of obstacles setting 95
A.3.1 2D scenario with OBB-shape obstacle 95
A.3.2 3D scenario with sphere-shape obstacles 97

Figures

1.1 Picture of AUV 1

2.1 Path Planning Methods for AUV 8

3.1 Picture of the AUV - LAUV Thor 18
3.2 Jerk, acceleration, velocity and position in transition[49] 19
3.3 Modelling of the obstacles with circumcircle 21
3.4 Original and expanded model of obstacles 21
3.5 Comparison between the circumcircle and oriented bounded block 21
3.6 Graphics illustration for obstacles in circle and rectangular shape . 22

4.1 Flowchart of particle swarm optimization method 28
4.2 Flowchart of Quantum-behaved Particle Swarm Optimization method 33

5.1 Graphics explanation of the distance between path and obstacle . . 40
5.2 Graphics explanation of the turning angle between path segments . 41
5.3 Comparison between paths with 4 and 8 interior points 45
5.4 Graphics illustration for geometric relationship between the point and the rectangular 47

6.1 Performance of the PSO algorithm with different pair of c_1 and c_2 . 54
6.2 Performance of the PSO algorithm with different strategies for inertia weight w 56
6.3 Performance of the PSO algorithm in complex environment with OBB-shape obstacles 57
6.4 Optimal path generated by PSO-based path planner in 3D scenario 59
6.5 Optimal path generated by PSO-based path planner in the NE-plane 59
6.6 Optimal path generated by PSO-based path planner in the DE-plane 60
6.7 The fitness value during iterations by PSO algorithm in 3D environment 60
6.8 Performance of the QPSO algorithm with different pair of c_1 and c_2 in simple 2D scenario 63
6.9 Performance of the QPSO algorithm with different pair of c_1 and c_2 in complex 2D scenario 64
6.10 Optimal path generated by QPSO-based path planner in 3D scenario 65

6.11 Optimal path generated by QPSO-based path planner in the NE-plane 66
6.12 Optimal path generated by QPSO-based path planner in the DE-plane 66
6.13 The fitness value during iterations by QPSO algorithm in 3D environment 67
6.14 Performance of the G-QPSO algorithm with random numbers in Gaussian distribution in simple 2D scenario 68
6.15 PPerformance of the G-QPSO algorithm with random numbers in Gaussian distribution in complex 2D scenario 69
6.16 Optimal path generated by GQPSO-based path planner in 3D scenario 70
6.17 Optimal path generated by GQPSO-based path planner in the NE-plane 71
6.18 Optimal path generated by GQPSO-based path planner in the DE-plane 71
6.19 The fitness value during iterations by GQPSO algorithm in 3D environment 72

Tables

4.1 Strategies for defining inertia weight 30

6.1 Parameter setting for PSO algorithm 52
6.2 Map information in a simple 2D environment 52
6.3 Map information in a complex 2D scenario with OBB-shape obstacles 53
6.4 Map information in 3D scenario with sphere-shape obstacles 53
6.5 Positions of waypoints generated by three pairs of c_1 and c_2 55
6.6 Total length of the paths generated by three pairs of c_1 and c_2 . . . 55
6.7 tested parameters of inertia weight 55
6.8 Positions of waypoints generated by PSO algorithm with different strategies of inertia weight 55
6.9 Total length of the paths generated by PSO algorithm with different strategies of inertia weight 55
6.10 Optimization Parameter in complex environment 57
6.11 Positions of waypoints in feasible and unfeasible path generated by PSO algorithm 58
6.12 The fitness value of the paths generated by PSO algorithm, including feasible and unfeasible path 58
6.13 Optimization Parameter in 3D scenario 58
6.14 Positions of waypoints generated by PSO algorithm in 3D scenario . 61
6.15 Optimization Parameter for β in QPSO algorithm 62
6.16 Total length of the paths with different (c_1, c_2) 63
6.17 Total length of the paths with different (c_1, c_2) 64
6.18 Optimization Parameter in 3D scenario 65
6.19 Positions of waypoints generated by QPSO algorithm in 3D scenario 65
6.20 Optimization Parameter for β in G-QPSO algorithm 67
6.21 Optimization Parameter in 3D scenario 70
6.22 Positions of waypoints generated by QPSO algorithm in 3D scenario 70

Chapter 1

Introduction

1.1 Background

1.1.1 Autonomous Underwater Vehicles (AUVs)

Autonomous underwater vehicles (AUVs) are a subclass of unmanned underwater vehicles (UUVs). As shown in Fig. 1.1, it is a self-propelled, unmanned, untethered underwater vehicle which is able to complete simple tasks with little or even without human intervention. The common task for AUVs is to work as the survey platform to do scientific research such as mapping the seafloor or monitoring and analysing the chemical property of the water.

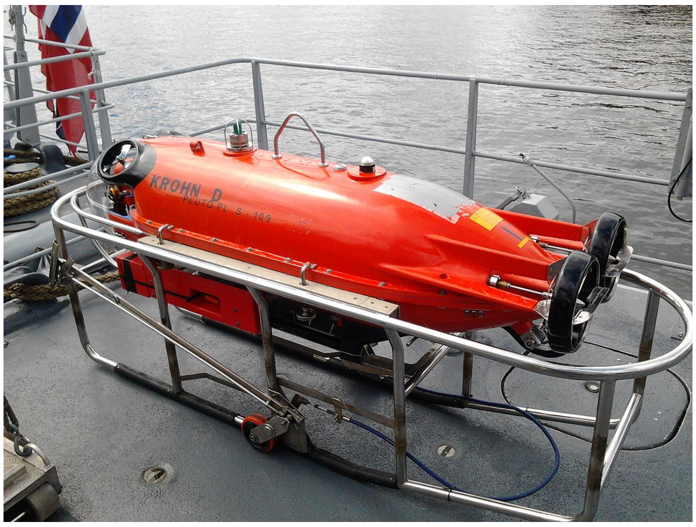


Figure 1.1: Picture of AUV

The first AUV in the world is developed in 1957 by Stan Murphy[1] and with the fast development of techniques, a large variety of AUVs are in existence. Nowadays, AUV has become a popular and important way to explore the ocean area where it is not possible for human to visit, such as areas with higher depth. Also, various subsea tasks in different fields, such as ocean pollution monitoring, marine biology exploration and pipe following and inspection could be achieved by those autonomous vehicles themselves[2]. The level of autonomy is a key parameter for AUV and its definition is introduced later.

1.1.2 Autonomy

When it comes to the concept of autonomy, people always get confused and could not tell the difference between autonomy and automation. Though these 2 words look quite similar, their corresponding systems differ a lot. The system with automation is called automatic system, which could perform well-defined tasks without any human intervention. If the task is not defined in advance, the automatic system is not able to handle it without human's help. While for the system with autonomy, which is also called autonomous system, it could complete these tasks. The autonomous system have more intelligent and adaptive functionalities. These make it possible for the system to learn in the unknown environment, adapt to it and improve itself. For these autonomous systems, there exist several definitions of its autonomy level. US National Institute of Standards and Technology (NIST) proposes one definition of the autonomy level, which divides autonomy into 4 levels shown as follows:

1. **Automatic operation - Human in the loop**: In this level, human intervention is indispensable. In marine applications, the remotely operated underwater vehicles (ROVs) belong to this level.
2. **Management by consent - Human delegated**: In this level of autonomy, the human intervention is not always needed. Some of the operations could be done by system itself and some of them should be done with humans' consent. For example, the optimal heading control in Dynamics positioning system is in this level.
3. **Semi-autonomous - human supervisory control**: Systems in this level of autonomy could handle most tasks without human intervention. But some specific tasks can only be done by humans. The emergency and safety systems are in this level of autonomy.
4. **Highly autonomous - Human out of the loop**: The system in this level could perform complex tasks with high uncertainty without any human intervention and could work well when unexpected events happen. An ideal AUV is in this level of autonomy.

Nowadays, the system of AUV is not fully in the forth level of autonomy. In some specific situation, human intervention is still necessary since the AUV could not handle the unexpected events. Therefore, there is still potential of autonomy for AUV and it is essential to increase its level.

1.2 Motivation

The motivation of this thesis is to increase the level of autonomy for AUVs focusing on path planner based on Particle Swarm Optimization methods. Researchers have claimed that the performance of the path planner is a key factor for achieving persistent autonomy in AUVs. Hence, it is necessary for AUV to have a qualified optimal path planner which could help to find the optimal path without human intervention. In the field of path planning, particle swarm optimization method has been developed for over 20 years and applied to solve engineering problems. Several PSO-based optimal path planners designed for Unmanned Aerial Vehicles (UAVs) have been tested in practical experiment and its performance is satisfying. However, applying it in underwater environment is not popular. Since its performance in UAV's applications has been tested, it can be assumed that with some modification, the PSO algorithm could be used to design optimal path planner for Autonomous Underwater Vehicles (AUVs) with good performances.

1.3 Research Topic and Objectives

In general, this master's thesis is based on the research hypothesis, that is, the research about path planning method is a core technique for achieving high level of autonomy for Autonomous Underwater Vehicles (AUVs) and Particle Swarm Optimization (PSO) method could be utilized to design a high-performance path planner for AUVs. According to this research hypothesis, the objectives of this master's thesis are presented as follows:

- Conduct a literature study about the state-of-the-art path planning methods and study further in the field of PSO algorithm, focused on its development and application on AUVs.
- Define the optimal path planning problem for AUV and propose its corresponding mathematical model.
- Choose a proper guidance system for AUV, such that the optimal path planner could have good cooperation with the control system of AUV.
- Modify the mathematical model for path planning in consideration of engineering aspect.
- Design an optimal path planner with PSO method based on the modified engineering model.
- Examine the performance of the PSO-based optimal path planner in different scenarios.

1.4 Scope and Limitations

The scope of this thesis is designing an optimal path planner with particle swarm optimization (PSO) method and examining its performance in different scenarios by simulations. A deeper literature review is carried out focusing on the development of PSO method. From the classical particle swarm optimization (PSO) method to Quantum-behaved Particle Swarm Optimization method with Gaussian Mutation (G-QPSO), this thesis covers the development of the optimal path planning methods based on swarm evolutionary strategy, as well as the implementation of these methods into practical path planning problem. As a part of the autonomy module in the AUV's control system, the optimal path planner deployed on AUVs is utilized to increase the level of autonomy and enable the AUV to complete scientific tasks without human intervention. The PSO-based path planner is developed using MATLAB and the simulations are all done with MATLAB.

One of the limitations of this thesis is lack of practical experiments. All results shown in this thesis are from simulations in MATLAB. Hence, the performance of the path planner in real marine environment is uncertain. Also, the testing scenarios given in this thesis are not in large scales. In the testing scenarios of 2D and 3D, the modified GQPSO-based path planner shows good performance. However, as for real environment in much larger scale, it is hard to say if the optimal path planner could work well. These limitations could be solved by future work presented in Chapter 8.

1.5 Structure of the Thesis

This section introduces the main structure of this thesis, which is outlined based on the research objectives presented in Section 1.3. The main structure is shown as follows:

Chapter 1 presents an introduction of this master's thesis, including the background, the motivation of the thesis, as well as the research topic and objectives. Also, the scope and limitations of the thesis are presented in this chapter.

Chapter 2 presents a literature review about the state-of-the-art path planning methods as well as their applications in the marine environment. Since the PSO algorithm is used in this thesis, a deeper literature review about it is presented, focusing on its development and applications.

In **Chapter 3**, a practical AUV called LAUV Thor is introduced. Information about its physical characteristics and devices installed on the vehicle is presented. To ensure good cooperation between the path planner and the practical AUV, the guidance system which works as the connection between them is introduced. Also, the mathematical modelling of the obstacles is presented.

Chapter 4 provides the general frameworks of the optimization theory used in the thesis, including PSO, QPSO and G-QPSO method. The basic concepts of an optimization problem is presented, as well as a deeper look into the meta-parameters in these optimization theories.

Chapter 5 presents the process of problem formulation. The preliminary model of path planning problem is introduced and modified to an engineering model which is suitable for applying PSO method. Based on this engineering model, practical algorithm in MATLAB is presented.

Chapter 6 contains the simulation results using the PSO-based path planners presented in Chapter 5. Their performances in three different scenarios including 2D and 3D are tested.

Chapter 7 presents the discussion of the results shown in Chapter 6.

In **Chapter 8**, the conclusions are drawn based on the discussion in Chapter 7. The limitations of the thesis are presented as well as the future work.

Chapter 2

Literature Review

In recent years, due to the fast development of techniques, autonomous underwater vehicle (AUV) has become a popular and important way to explore the ocean area where it is not possible for human to visit, such as areas with higher depth. Also, various subsea tasks in different fields, such as ocean pollution monitoring, marine biology exploration and pipe following and inspection could be achieved by those autonomous vehicles themselves[2]. Researchers all over the world have proposed many path planning methods considering collision avoidance with surrounding obstacles and other vehicles. However, due to the fast development of artificial intelligence (AI), many recent advances and breakthroughs emerge in the field of path planning for AUV. Like autonomous driving, applying autonomy in underwater vehicles has become a hot topic and attracted the attention from both industry and research institutes.

In this chapter, several state-of-the-art path planning methods are introduced and their advantages and disadvantages are also discussed. Focusing on particle swarm optimization (PSO) method, a deeper literature review is presented about its development and applications in the field of path planning, especially for Autonomous Underwater Vehicles (AUVs).

2.1 Main Path Planning Methods for AUV

The mission of path planning for AUV could be considered as a series of translations and heading angle changes from the starting point to the destination point. By analyzing the operation environment of AUV and its main operations like mapping and monitoring, it could be found that most of the time, AUV will operate in the environment which is full of obstacles, which are both static and dynamic.

Considering the simple case, that is the location and shape of the static obstacles could be obtained beforehand by measurement or previous knowledge, a global map of this environment with obstacles information could be achieved before path planner works. Hence this global map could be used to figure out a desired path without collision with obstacles between the starting point and destination point. These methods are called global path planning method. Moreover, these global path planning methods could deal with path planning problem with other objectives, such as path length, energy consumption, as well as problems considering the influence of underwater current.

However, if the information of some static obstacles could not be obtained in advance, which means there are unknown static obstacles, these global path planning methods are not able to figure out a desired collision-free path. They could not solve problems with dynamic obstacles either due to the same reason. Unfortunately, the real underwater environment is always dynamic and uncertain. Hence, it is quite difficult, or even impossible to obtain information of all obstacles in advance, which means the global path planning methods are not totally qualified as a path planner. In these cases, the global path planner is still necessary to figure out a general path, and an additional path planner is needed to help AUV avoid those unknown static and dynamic obstacles, such as reefs, animals like fish or other rock structure on seafloor. Therefore, in this report, the path planning methods for AUV are divided into 2 categories, one for global path planning with known static obstacles, the other for local path planning with unknown static and dynamic obstacles, which are shown in Fig. 2.1

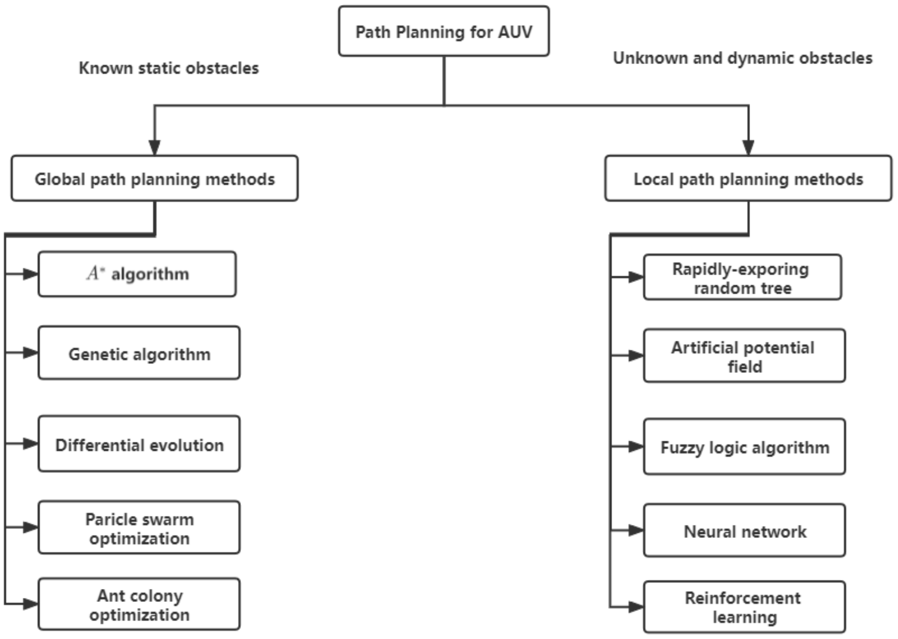


Figure 2.1: Path Planning Methods for AUV

2.1.1 Global Path-planning Methods

In the field of global path-planning methods for AUV, scientists have proposed and developed many popular and commonly used methods in recent decades, which have been proved qualified as global path planner for AUV.

In 1985, Dechter and Pearl[3] firstly proposed the A^* algorithm, which is the most effective direct search algorithm in the environment with known static obstacles. It achieves the desired path by combining heuristic searching and searching based on the shortest path. Considering the characteristics of underwater environment, A^* method are modified in the heuristic function by Carroll[4], by taking operation depth for AUV and corresponding current information into account. Furthermore, in order to reduce time for searching, a sparse A^* method is proposed by adding constraints of maximum path length and turning angle to the original A^* method by Szczerba[5]. Moreover, to make the desired path more suitable for operation of AUV, scientists developed these basic A^* method further. Chen used randomly distributing points to construct the search space of obstacles and added a constraints of maximum turning radius[6], which makes the path smoother and easier to track for AUV. An obvious shortcoming of A^* method is time-consuming. To shorten the search time, a multi-directional A^* method is proposed by Li and Zhang[7], which reduces the number of searching nodes and then shortens the searching time.

Not like the A^* method, another famous global path-planning method, genetic algorithm is proposed by Cobb and Grefenstette in 1993[8] with the inspiration of natural selection and evolution and has been widely used when solving optimization problems. In order to make genetic algorithms adapt to particularity of AUV and underwater environment, these algorithms are commonly modified in the part of genetic operator and evaluation factor. Alvarez and Caiti added a new genetic operator to the genetic algorithm, which enables the ability of global convergence, especially when there are several different local optimum in the field[9]. Furthermore, in order to enhance the capacity in environment with strong current, they made an improvement on the previous genetic algorithm by adding an iterative operator and a random migration operator[10], which are able to have a better control on the initial population and mutation rate. Moreover, the converging rate and energy consumption of the path could be improved through modification on genetic algorithms. Sun and Zhang's work[11] proves that using ocean current as an evaluation factor could achieve a better path with lower energy consumption. Cao [12] proved that improvement of initial population generation method could greatly fasten the converging rate. All of these work have shown that the genetic algorithm has achieved a great success in global path planning for AUV with a global map in advance, which contains all information about the static obstacles.

Similar to the genetic algorithm introduced above, the differential evolution method could be considered as a global path planning method for AUV. It was proposed by Storn and Price in 1997 and consists of mutation, crossover and selection operation. The difference between these 2 methods is, in differential evolution, the mutation vector is formed by the parent generation's difference vector, which intersects with the parent generation's individual vector to generate a new individual vector[13]. Hence, it could be expected that the differential evolution method have a better performance than the genetic algorithm in the field of path planning, especially when considering the efficiency. To make the algorithm suitable to scenarios of AUV application and improve its performance, researchers always modified its cost function and combing it to the kinematics of AUV's model. Zhang applied adaptive strategy to the basic differential evolution method[14]. He used penalty methods on the cost function and considered the curvature constraints, path length and energy consumption. These considerations are used to adaptively adjust the parameters based on the size and position of the obstacles[14]. In some simple case, this method has been verified. Li implemented the differential evolution algorithm on a simulation platform with task for obstacle avoidance. The proposed method has been verified by conducting experiments with single and multiple obstacles on a simulation platform[15].

Different from the method above, Eberhart and Kennedy proposed particle swarm optimization method in 1995, which uses evolutionary computation technology based on random population[16]. Kind of like genetic algorithm, the inspiration of this method comes from the foraging behavior of birds. When birds are seeking for food, they don't know the exact position of the food, instead they know where they are now and how far away the food is. Their search approach is to follow the bird that is closest to food. In the application field for AUV, the particle swarm optimization has been widely used, as well as its most famous modification, quantum particle swarm optimization method. Yang and Zhang proposed an adapted inertia-weight particle swarm optimization algorithm which takes the current's speed and direction into account in the fitness function[17]. By considering these factors, the AUV is able to complete the tasks with strong current. Moreover, for off-line path planning of AUV, Lim presented a particle swarm optimization method with selected differential evolution. Selecting the best particle for differential evolution hybridization can cut down on computation time significantly[18]. Also, this method was tested and verified in an environment with known obstacles and time-invariant non-uniform currents.

Similar to the particle swarm optimization method, the inspiration of ant colony optimization, which is a heuristic optimization method proposed by Dorigo in 1991, comes from the foraging behavior of any colony in nature[19]. The basic principle behind the ant colony method is that, it represents one feasible solution to the path optimization problem by using the behavior of a single ant, and the behavior of the entire ant colony defines the problem's solution space. It works well in solving optimization problem and could also work well after some modification and adaptation to the application of AUV. Wang and Wei improved this method and implemented it on AUV applications[20]. By adding a cutting operator and insertion point operator, a smooth desired path could be achieved quickly and meet the requirement of obstacle avoidance. In order to reduce invalid search, Ma added an alarm pheromone in the original algorithm[21]. When an ant reaches the inaccessible region, it sends out alarm signals the ants behind it. This enhances its ability of obstacle avoidance. In addition to the objective of a collision-free path, ant colony optimization method could solve problems with more objectives. A multi-objective any colony optimization method was proposed by Hu and Zhang[22], which also considers the influence of current on energy consumption and safety performance of the AUV.

In summary, a general discussion about the advantages and disadvantages are done and their pros and cons are concluded, as well as their suitable operation environment.

A^* method is the most effective algorithm using direct search strategy. In A^* algorithm, a shortest and collision-free path could be achieved without any operation in advance. Though it is easy to understand and implement, A^* method still has some problem, especially when solving large-scale problems. Due to its own searching strategy, it has to go through all nodes in the searching area and calculate the cost function of these child nodes. This process is quite time-consuming and influences the convergence rate especially when the task is given in a large-scale problem. The genetic algorithm and differential evolution method have similar process structure. Hence they suffer from the same disadvantages, that is, the optimization process will occupy a large storage space and more parameters should be calculated and adjusted. Though they have these drawbacks, these 2 methods are still widely used because of their strong ability of global searching for optimal solution, as well as their superior robustness. Particle swarm optimization and ant colony optimization are both inspired by the foraging behaviors of animals in nature. With this property, they could adapt to the environment in a short time. But the convergence rate of these 2 methods are different due to the difference in algorithm structure. At the beginning, the searching period of ant colony optimization method is quite long which results in a low convergence rate. As time goes, the convergence rate becomes higher. But for particle swarm optimization, its convergence rate is higher in the initial stage. However, the particle swarm optimization method has higher tendency to converge at a local minimum, which is a significant disadvantages.

2.1.2 Local Path-planning Methods

Unknown and dynamic obstacles could not be solved by global path-planning methods, thus they are always tough problems for AUV's path planning. Researchers try to solve this problem all the time and some classical methods were proposed and verified, such as RRT method, artificial potential field method and fuzzy logic algorithm. In recent years, With quick development of artificial intelligence (AI), more local path-planning methods with AI tech such as neural network and reinforcement learning appear and attract researchers' attention.

Considering dynamic constraints, Tan proposed the rapidly-exploring random trees (RRT) algorithm, which a suitable candidate to solve local path-planning problem in high-dimensional unknown environment[23]. In Tan's algorithm, they considered not only the algebraic constraints from interaction with obstacles, but also the differential constraints with dynamics of AUV's model. By taking both of them into account, a desired path without any collision could be derived from this method effectively between the starting point and destination point. To achieve higher efficiency, Hernández proposed an modification method which is called homotopy rapidly-exploring random trees (HRRT) algorithm[24]. HRRT algorithm could achieve high efficiency by reducing unnecessary space exploration. Furthermore, the transition rapidly-exploring random trees algorithm is modified and implemented in AUV application[25], which has been proved qualified for exploration tasks near seabed. Based on different purposes, researchers have developed several modification of RRT algorithms, such as liveness-based RRT (Li-RRT) by adding a liveness index to describe the nodes' effectiveness[26], smooth-RRT algorithm using greedy strategy to get smooth path[27], closed-loop rapidly-exploring random tree (CL-RRT) algorithm solving the kinematic constraints from the characteristics of AUV and obstacles[28], as well as RRT^* algorithm. All of these methods have some progress in specific part of the path planning performance and have been proved in simulation environments.

Artificial potential field method is a real-time path planning method proposed by Khatib in 1986[29]. Initially, it was widely used in mobile robot because of its advantages of low cost. In 2005, Ding introduced this method to solve the path planning problem for AUV[30]. They added an additional virtual force in the algorithm and successfully avoided the defect of original algorithm. Moreover, considering both the dynamic obstacle and influence of the current, Cheng modified the artificial potential field method and achieved good performance of overcoming the influence of the environment and collision avoidance with dynamic obstacles[31]. Also, this method could be used for multi-AUV scenario. Ge improved this method and made AUV succeed in completing the searching task and avoiding collision[32].

The concept of fuzzy logic algorithm was proposed by Cordón in 1996[33]. It utilizes the expert's knowledge to figure out a nonlinear mapping from the state space to the control space. It is not a pure path planner but a controller which is able to avoid the collision. Khanmohammadi applied this method to AUV and did a simulation with this fuzzy logic algorithm. The simulation result show that it achieved success in avoiding dynamic obstacles[34]. Even though it has the same function as a local path planner, but actually the fuzzy logic algorithm works as a controller in the system. Therefore, it will not be introduced too much in this report.

With the fast development of artificial intelligence, the neural network is proposed and well developed in recent years. Researchers have applied it into path planning for AUV. In a traditional way, a well-trained neural network could figure out the signals to avoid collision with data from sensors as input. However, the training process is quite hard, time-consuming and even costly. In order to overcome this, Yang and Meng proposed a bio-inspired neural network in 2003, which has good performance without any prior training process[35]. Due to the particularity of AUV and underwater environment, the training process is more expensive and dangerous than other cases. Therefore, the bio-inspired neural network is widely used in the local path planning for AUV. With this convenient neural network for AUV applications, several attempts are done by researchers by using neural network as a the local path planner for AUV. Yan and Zhu proposed an improved bionic neural network to solve full coverage path planning problem for AUV[36]. Zhu tested this bio-inspired neural network in an unknown dynamic environment, by combining it to a map planning method[37]. Ni also proposed an improved version of this neural network and the experiment results show that it achieves success in solving path planning problem in large environment[38]. In addition, this bio-inspired neural network could be used for multi-AUV obstacle avoidance problems by using a improved version called leader–follower biological inspired neural network[39].

Similar to neural network, reinforcement learning (RL) has experienced a fast development in recent years and achieved success in path planning field. The main idea of reinforcement learning is to help the agent learn a optimal policy by interacting with the environment via trial, which could guide the agent performing best actions to complete the task. In the field of path planning for AUV, the usage of reinforcement learning enables it to learn through its own experience and gradually adapt to the environment without knowing the complete prior knowledge or even the prior knowledge at all[40]. Kawano and Ura applied reinforcement learning to path planning for AUV, by using a Q-learning algorithm, which is a basic algorithm with reinforcement learning tech, and combing with teaching method and Bayesian network[41]. Furthermore, in order to improve the learning speed, they proposed a modified version using hierarchical reinforcement learning approach and reached their goal. Considering the scenario with unknown and dynamic obstacles, reinforcement learning tech has been proved qualified for the path planning work. Gore's work shows that by using reinforcement learning tech, AUV can derive an optimal path with minimum deviation from obstacles from the space by taking corresponding actions[42]. Moreover, researchers have modified this method to adapt to more realistic problems. Bhopale proposed a modified Q-learning algorithm which is suitable for large dimensional problems[2]. Also, modern method combining reinforcement learning and deep learning achieves great success, these method are called deep reinforcement learning (DRL). All of these new methods give more opportunities to solve the local path planning problem. With the introduction of reinforcement learning and neural network, improving the autonomous level of path planning method for AUV has become a hot topic and attracted researchers' attention. Although many methods are implemented and tested, there are still a lot of potential problems which need to be solved.

Not like global path planning methods, local path planning methods have less similarities between each other and have distinct advantages and disadvantages. Here is a discussion of local path planning methods introduced above. Firstly, the RRT method takes the algebraic constraints of obstacles as well as the differential constraints into account, which achieves a suitable path considering the AUV's dynamics. Also, the searching strategy of RRT method makes it easier to explore the unknown region, which is quite common in marine applications. These strong ability of exploring unknown region makes it more suitable for path planning in underwater environment. However, this strong exploration ability becomes a burden when considering the real-time computing performance. The artificial potential field method achieves success in path planning because of its simple mathematical structure and easy implementation. However, not like the RRT method, it does not consider the differential constraints derived from dynamics model of AUV. And in some specific scenario such as multiple obstacles with same size, it will have a bad performance. The fuzzy logic algorithm is more like a controller instead of a path planner. Moreover, it is highly dependent on the expert's knowledge to construct the fuzzy rules. Hence, it is not a proper choice for local path planner in unknown environment.

Neural network and reinforcement learning tech have experienced a fast development in recent years. Both of them belongs to the field of artificial intelligence and could achieve a good performance after sufficient learning. However, in marine applications, the cost of collecting data and training model is extremely high and full of risk. In order to get rid of this, researchers have proposed many methods which do not need too much training process in advance such as bio-inspired neural network. Also, with the development of the simulation software, the process of collecting data and training model may become easier.

2.2 Particle Swarm Optimization (PSO) for Path Planning

As mentioned in Section 2.1.1, Kennedy and Eberhart proposed a population-based evolutionary computation algorithm called particle swarm optimization (PSO) in 1995. Though the inspiration of it is quite similar to that of genetic algorithm, the computational efficiency of them differs a lot. Researchers have proved that the PSO algorithm is more efficient than the genetic algorithm in terms of computational cost, even though their effectiveness are almost the same on average[43]. Since the convergence ability to global optimum of the PSO algorithm is not sufficiently strong, some modified versions of it are proposed. The most well-known one is called Quantum-behaved Particle Swarm Optimization (QPSO) algorithm, which is a new PSO algorithm based on the quantum mechanics of the particles. The nature of the particles in QPSO algorithm is totally different from that in classical PSO algorithm. In quantum mechanics, the motion of the particle is not described by the velocity vector, but by a δ-potential. By utilizing this, the convergence ability to global optimum becomes much stronger than that with classical PSO algorithm[44].

In order to improve the performance of the QPSO algorithm, lots of research has been done by scientists and some modified versions of it are proposed. Sun proposed several improved QPSO algorithms with the purpose of better global convergence ability as well as the accuracy[45]. The GQPSO algorithm, which is short for Quantum-behaved Particle Swarm Optimization with Gaussian mutation, has been proposed by using Gaussian potential. The GQPSO algorithm makes it possible to prevent premature convergence, which refers to convergence to local optimal solution instead of the global optimal one. It has been utilized to solve engineering problem by Coelho in 2007[46]. Moreover, by introducing a weight coefficient, a modified version called Weighted QPSO (W-QPSO) is proposed by researchers. The weight coefficient is utilized in calculating the mean best position ($Mbest$) in QPSO in order to render the importance of particles in population when they are evolving[47]. In W-QPSO algorithm, the particles in the whole swarm are ranked according to its fitness value (the value of the objective function), and then a linearly decreasing weight coefficient is applied[47].

In the field of path planning, the particle swarm optimization algorithm has achieved great success and many scientists are trying to apply this algorithm to solving different kinds of optimal path planning problem. An online path (trajectory) planning method for Unmanned Aerial Vehicles (UAVs) based on QPSO algorithm was proposed by Guo in 2009, which utilizes quadrinomial and quintic polynomials to describe the path[48]. The main objectives in this application are minimizing the total length of the path as well as the time expense.

In summary, the particle swarm optimization (PSO) method has been proposed and developed for over 20 years. In order to overcome the drawbacks of the original PSO algorithm such as weak global convergence ability, many modified versions of it have been proposed by researchers. In the engineering aspects, PSO algorithm and its modified versions have been used to solve path planning problem. The PSO-based path planner has achieved success in some way. Therefore, the particle swarm optimization method could be seen as a good choice for AUVs' path planning.

Chapter 3

AUVs' Guidance System and Map Processing

The research of this thesis is based on the NTNU's AUR-Lab's property, the AUV called LAUV Thor. In this chapter, an introduction about this vehicle is presented, including its physical characteristics, the devices installed on the vehicle as well as a glance of its structure. Since the path planning method talked about in this thesis is used to output an optimal path which is also the input of the guidance system, a common type of guidance system is introduced to demonstrate that the optimal path generated by path planner could be used as the input to the guidance system. Moreover, in the real underwater environment, the obstacles detected by the sensor come in many shapes, some of them might have complicated shapes which are difficult to model. Hence, the strategy of obstacles modelling is proposed and presented in this chapter.

3.1 LAUV Thor

The Autonomous underwater vehicle LAUV Thor shown in Fig. 3.1 is the asset of the NTNU research center for underwater robotics, Applied Underwater Robotics Laboratory (AUR-Lab). The LAUV system of it was originally developed by the Underwater Systems and Technology Laboratory (LSTS) from the Porto University and has been further developed in cooperation with OceanScan - Marine Systems & Technology, Lda.

The LAUV, which is short for Light Autonomous Underwater Vehicle, is a modular platform integrated with a set of different sensors and sonars with light weight. The vehicle is designed to complete tasks about cost-effective oceanographic, environmental and inspection surveys and the modular design makes it suitable for a wide range of scientific and civilian applications.

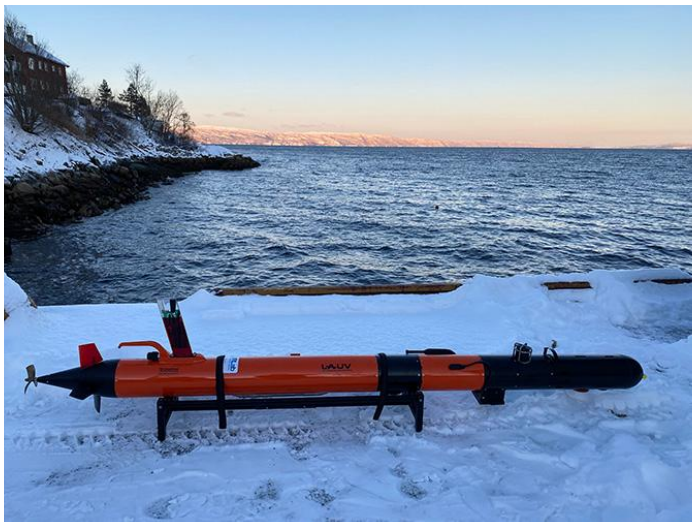


Figure 3.1: Picture of the AUV - LAUV Thor

With the target of fulfilling various applications, many types of sensors have been integrated with the vehicle. The devices installed here are shown as follows:

1. Environmental sensors measuring Conductivity, Temperature, Depth (CTD), sound speed, fluorescence, turbidity
2. Forward looking sonar
3. Acoustic transducers (Long baseline (LBL) and Acoustic modem)
4. Illumination module
5. Multi-beam echo-sounder
6. DVL (Doppler Velocity Log)
7. IMU (Inertial Measurement Unit) or INS (Inertial Navigation System)
8. On-board CPU & solid-state hard disk
9. Communication and Navigation boards (Wi-Fi, GPS, GSM, Iridium, Compass)
10. Batteries and cameras
11. Emergency pinger

The total length of this vehicle is 226 cm and its weight in air is 35 kg. The maximum operational depth is 100 m and its speed is limited in the range of 0.5 to 2.0 m/s. The main structure of LAUV is designed to achieve a balance of many factors, including its weight and buoyancy as well as the space for modular devices installed on the AUV. Also, sufficiently good robustness is required.

3.2 Guidance System with Constant Jerk

In the motion control system of the AUVs, the guidance module is used to generate desired states for the whole system and send the desired states as input to the actuator module. Since the optimal path in this thesis consists of a sequence of waypoints, a suitable guidance system which could complete the task based on waypoints should be selected. The constant jerk guidance system for position reference could be a good choice. Constant jerk is a guidance scheme based on the waypoints. With this system, the optimal path generated by the path planner does not need any modification and could be used as the input to the guidance system, which could save the computational cost and improve the efficiency.

Given the desired path with a sequence of waypoints, the constant jerk guidance system could generate the desired states and ensure smooth transition in position and velocity. In this algorithm, some physical constraints about the velocity, acceleration and jerk are considered. With the known upper bound for these states, the algorithm could ensure that the desired states are practical and not beyond the ability of the actuators.

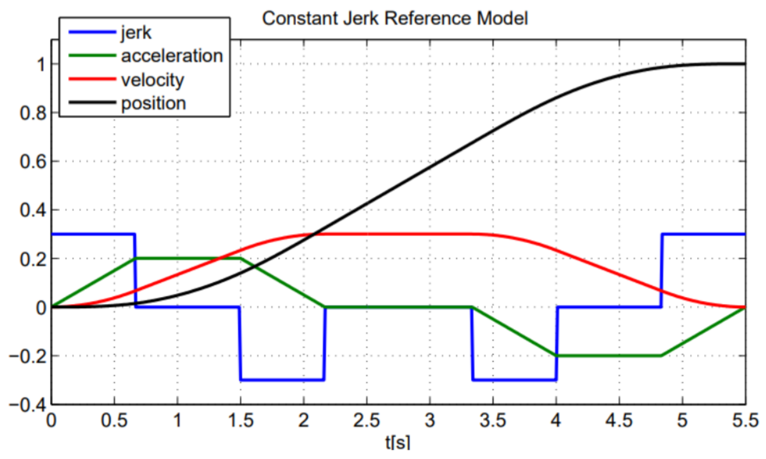


Figure 3.2: Jerk, acceleration, velocity and position in transition[49]

In the transition between 2 adjacent reference waypoints $\mathbf{P}_k$ and $\mathbf{P}_{k+1}$, there are seven phases shown as follows and the graphics illustration is shown in Fig. 3.2.

Phase 1: Start acceleration: constant jerk leads to increase in acceleration

Phase 2: Constant acceleration: jerk is equal to zero, constant acceleration leads to increase in velocity

Phase 3: End acceleration: constant negative jerk leads to decrease in acceleration to zero

Phase 4: Constant velocity: velocity reaches maximum and keeps constant (main transit phase)

Phase 5: Start deceleration: constant negative jerk leads to increase in deceleration

Phase 6: Constant deceleration: jerk is equal to zero, constant deceleration leads to decrease in velocity

Phase 7: End deceleration: constant positive jerk leads to decrease in deceleration as well as velocity to zero

With the algorithm presented above, the guidance system could generate the desired states, including the velocities v_i and positions p_i at each time instant, which describes the transition motion between waypoint $\mathbf{P}_k$ and $\mathbf{P}_{k+1}$. Since the candidate path generated by the path planner consists of a sequence of waypoints, it could be used as the input to the guidance system without any modification. Therefore, the path planner matches well with the guidance system as well as the whole control system.

3.3 Map Processing

One main purpose of the path planning for AUV is to avoid collision with the obstacles and ensure safety. In real underwater environment, the obstacles come in all shapes and sizes. The AUVs identify these obstacles using different sensors like radar and sonar. By observing the images from radar and sonar, the shape of the obstacles identified by the AUVs might be complicated. In general, running the path planner based on these unprocessed information of the obstacles, the computation process costs a lot and the computation time is pretty long. In order to save the computational cost and shorten the computation time, it is necessary to pre-process the map before deploying the path planning algorithm. In this thesis, the obstacles are all represented by algebraic models and the bounding volume is used to simplify the shape of them. For obstacles with small length-width ratio, the circumcircle is used. For that with large length-width ratio, it is modelled as a Oriented Bounded Block (OBB).

For obstacles with small length-width ratio, using circumcircle to simplify its shape leads to little loss of feasible path, Moreover, in the process of solving path planning problem, the circle-shape model of the obstacles helps save computational cost since judging if a point is inside a circle/sphere is always a one-dimensional problem. The size of the circumcircle is defined by calculating the length of the lines between each pair of vertices. The diameter of the circumcircle is the length of the longest line and the center of it is the midpoint of the longest line. Some examples are shown in Fig. 3.3.

Considering the uncertainties in the working process, such as uncertainty of movement, actuators and the navigation system, models of the obstacles are expanded. A safety threshold ϵ is used to expand the obstacles. The original and expanded models are shown in Fig. 3.4.

Though the circumcircle model of the obstacles works well and is easy to apply, it still has some disadvantages and is not suitable for all obstacles. As the Figure 3.5 shows, in this specific scenario, if the obstacle with large length-width ratio is modelled by its circumcircle, the safe path in green line is not feasible anymore,

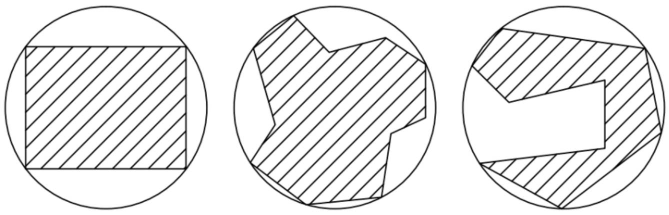


Figure 3.3: Modelling of the obstacles with circumcircle

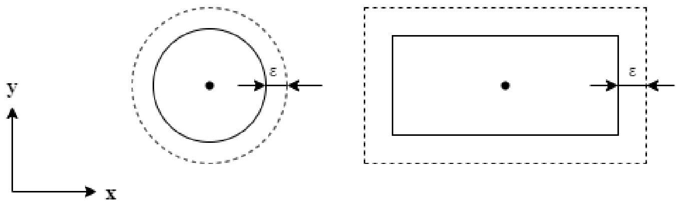


Figure 3.4: Original and expanded model of obstacles

while by using a oriented bounded box to model the obstacle, the path in green line becomes feasible.

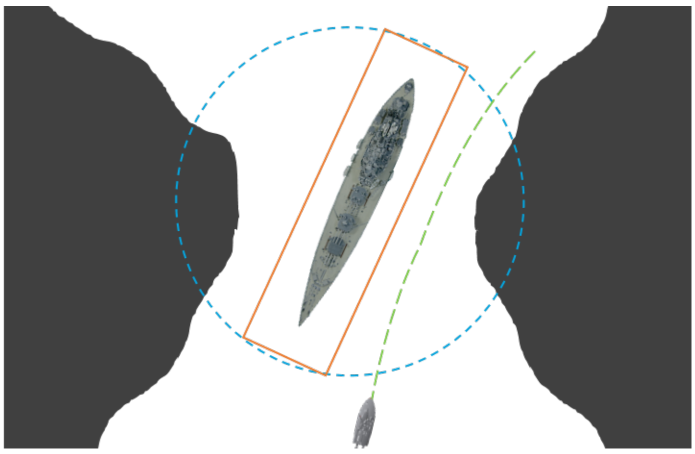


Figure 3.5: Comparison between the circumcircle and oriented bounded block

Since in the marine environment, some common types of obstacles such as large fish and ship have large length-width ratio. If these obstacles are modelled as a circle, there could be much loss of the feasible region and paths. Therefore, for these kind of obstacles, an oriented bounded box (OBB) is used. Though this model leads to more computational cost in the practical algorithm, the feasible region with it is larger and better paths could be generated. With the same consideration of uncertainty, this OBB model is also expanded and shown in Fig. 3.4.

The mathematical representations of the 2-dimensional obstacles are shown in Fig. 3.6.

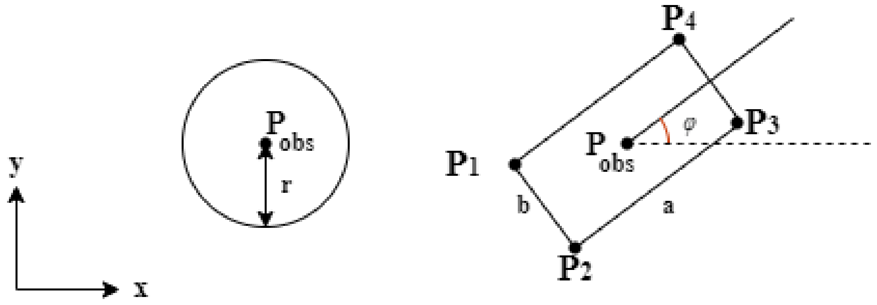


Figure 3.6: Graphics illustration for obstacles in circle and rectangular shape

For a circle-shape obstacle, it is represented by two parameters, the position of the center and the radius, which are expressed as:

$$\mathbf{P}^i_{obs} = [x^i_{obs}, y^i_{obs}] \quad \text{and} \quad r^i_{obs} \tag{3.1}$$

where i is the index of the circle shape obstacle.

For obstacles modelled as oriented bounded block, 4 parameters are used to describe the obstacle, the length a and width b, the position of the center $\mathbf{P}_{obb}$ and the orient angle ϕ, which are expressed as follows:

$$\mathbf{P}^i_{obb} = [x^i_{obb}, y^i_{obb}]\ ,\ r^i_{obb}\ , a^i \text{ and } b^i \tag{3.2}$$

where i is the index of the obstacle modelled as OBB. The positions of 4 vertices could be generated with the parameters in Eq. 3.2 and the mathematical expression is shown as follows:

$$\mathbf{P}^i_1 = [x^i_{obb} - \frac{a}{2}, y^i_{obb} + \frac{b}{2}] \cdot \mathbf{R}(\phi^i) \tag{3.3a}$$

$$\mathbf{P}^i_2 = [x^i_{obb} - \frac{a}{2}, y^i_{obb} - \frac{b}{2}] \cdot \mathbf{R}(\phi^i) \tag{3.3b}$$

$$\mathbf{P}^i_3 = [x^i_{obb} + \frac{a}{2}, y^i_{obb} - \frac{b}{2}] \cdot \mathbf{R}(\phi^i) \tag{3.3c}$$

$$\mathbf{P}^i_4 = [x^i_{obb} + \frac{a}{2}, y^i_{obb} + \frac{b}{2}] \cdot \mathbf{R}(\phi^i) \tag{3.3d}$$

where **R** is a 2×2 rotation matrix and its definition is expressed in Eq. 3.4.

$$\mathbf{R}(\phi^i) = \begin{vmatrix} \cos\phi^i & \sin\phi^i \\ -\sin\phi^i & \cos\phi^i \end{vmatrix} \tag{3.4}$$

Chapter 4

Optimization Theory

This chapter aims to provide general frameworks of the optimization theories used in this thesis, including Particle Swarm Optimization (PSO), Quantum Behaved Particle Swarm Optimization (QPSO) and Gaussian Quantum Behaved Particle Swarm Optimization (G-QPSO). The basic concepts of optimization problem are reviewed and classical algorithm with these techniques are also presented in this chapter. Section 4.1 - 4.3 introduce these three techniques separately. Also, a deeper look into the algorithms and meta-parameters is done in order to find out their shortcomings.

4.1 Particle Swarm Optimization (PSO)

In science, the term "optimization" refers to the process of identifying the best element (according to one or more user-specified criteria) among a set of possible alternatives[50]. When using mathematical languages for illustrating this process , this is always accomplished by defining a goal in terms of a parameterized function f, which is also called cost function or objective function. With this parameterized function f, the process of optimization comes down to finding out the specific values of these parameters, which minimize or maximize objective function f. The choice for maximization or minimization depends on the task and the requirement but maximization of f is equal to minimization of $-f$. Therefore, only minimization problem is discussed in this chapter. Therefore, the optimization problem could be defined as the following form.

$$\begin{aligned} &Given \quad f : \mathbb{R}^n \longrightarrow \mathbb{R} \\ &Find \quad \mathbf{x}_{optimal} | f(\mathbf{x}_{optimal}) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^n \end{aligned} \tag{4.1}$$

The parameter n in Eq .4.1 refers to the dimension of this optimization problem. The vector $\mathbf{x}$ is the candidate solutions. These solutions are in a n dimensional domain, which is called search space and characterized as $\mathbb{R}^n$. $\mathbf{x}_{optimal}$ is the optimal solution minimizing the objective function f. The function f, which is also called cost function or objective function, quantifies the fitness of the candidate solutions for the specific problem.

4.1.1 Classical Particle Swarm Optimization (PSO) Algorithm

Particle swarm optimization (PSO) method uses evolutionary computation technology based on random population. The inspiration of this method comes from the foraging behavior of birds. When birds are seeking for food, they don't know the exact position of the food, instead they know where they are now and how far away the food is[51]. Their search approach is to follow the bird that is closest to food. In other words, they find out food location through studying themselves and exchanging information with others.

In theory, individuals of a group or swarm, may profit from the prior discoveries and experiences of all group members. Based on this, a hypothesis is posed to develop the PSO, that is, the exchange of information among individuals in a group provides an evolutionary advantage[16].

In PSO algorithm, each candidate solution is called a "particle" and represents a point in the search space. The whole swarm is constituted of N particles. With these particles, PSO could explore the search space in the same way of other population-based algorithms. In the process of searching for the optimal solution of the problem, two important kinds of information are utilized, which are their own experience and experiences from other particles. With their own experience, each particle could explore around itself and find out how much progress these movements could make. Experiences from other particles could help it know other particles' best choices.

When searching for the optimal solution, each particle keeps track of its positions in the search space by updating itself based on the following equation:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{4.2}$$

In Eq .4.2, t and $t+1$ are the indices of two successive iterations in the PSO algorithm. It can be found that the most important thing in this updating rule is $\mathbf{v}_i(t+1)$, which governs how particles move in the search space. This velocity vector $\mathbf{v}_i(t+1)$ is called particle velocity and associated with three terms. The first one refers to the inertia of the particle. The second one is about the best choice it has achieved so far from its own experience and the last one is associated with the overall best choice from experiences of all particles, which is the global version of the second one. The updating rule for the particle velocity is defined as below:

$$\mathbf{v}_i(t+1) = w\mathbf{v}_i(t) + c_1[\mathbf{p}_i(t) - \mathbf{x}_i(t)]\mathbf{R}_1 + c_2[\mathbf{p}_g(t) - \mathbf{x}_i(t)]\mathbf{R}_2 \tag{4.3}$$

In Eq .4.3, w is the inertia weight. $\mathbf{p}_i$ is the personal best ($pbest$) of the i_{th} particle, which is based on its own experience, While $\mathbf{p}_g(t)$ is the global best ($gbest$), which is the overall best solution from the previous experience of all particles. c_1 and c_2 are constants and called cognitive and social coefficient, which will be talked about in section 4.1.2. $\mathbf{R}_1$ and $\mathbf{R}_2$ are two random diagonal matrices which will also be introduced later.

By using the updating rule 4.2 and 4.3, each particle changes its position in the search space and updates its velocity in each iteration. By using its own experience $pbest$ and others' experience $gbest$, the particles tend to find the optimal solution after sufficient iterations.

The basic elements of the classical PSO algorithm are stated and defined as follows:

- Particle $\mathbf{x}_i(t)$, $i = 1, ..., n$: It is the basic component in PSO algorithm representing the candidate solution in the form of a n-dimensional vector, n is the dimension of the problem which is always the number of optimization variables
- Swarm: It is the sum of moving particles distributed in the search space disorderly and randomly. The components in the swarm tend to cluster together while the moving direction of each particle differs from one to others.
- Personal best position $\mathbf{p_i(t)}$, $i = 1, ..., n$: It represents the best position for the i_{th} particle which corresponds to the minimum value of the cost function (fitness value). As the particle moves, it compares the value of the cost function at the current position to the minimum value it has achieved before, and updates the best position in each iteration.
- Global best position $\mathbf{p_g(t)}$: It is the best position with the minimum fitness value among personal best positions for all particles achieved so far.
- Particle velocity $\mathbf{v_i(t)}$, $i = 1, ..., n$: It is the velocity of the i_{th} moving particle which is also in the form of a n-dimensional vector. The particle velocity is updated with the rule in eq. 4.3 based on the previous speed, personal best position and global best position. After obtaining the new velocity vector, the movement of the particle is known and the position is updated with eq. 4.2

The scheme of the classical PSO algorithm is given by the following steps and the flowchart is shown in Fig. 4.1

Step 1: Initialization of particles' positions and velocities in the whole swarm: Initialize a pre-defined population of particles randomly, including their positions and velocities in the n-dimensional search space. A common way is to use some uniform probability distribution function to obtain these values.

Step 2: Evaluation of particle's fitness: Calculate the value of the objective function (fitness value) for each particle. In this part, the main goal is minimizing the objective function rather than maximizing.

Step 3: Comparison to fitness of $pbest$ (personal best position): Compare each particle's fitness with the fitness of the personal best position. If the value at current position is less than that at $pbest$, then set the current position as the $pbest$ position and update the fitness value of $pbest$ with the current fitness value.

Step 4: Comparison to fitness of $gbest$ (global best position): Compare the fitness of the current position with fitness of the population's overall best positions achieved before. If the value at current space is less than that at $gbest$, then reset the global best position as the current position and do the same for the fitness value.

Step 5: Updating of each particle's velocity and position: Calculate the velocity vector $\mathbf{v_i}(\mathbf{t}+\mathbf{1})$ and update the particle's position $\mathbf{x_i}(\mathbf{t}+\mathbf{1})$ according to Eqs. 4.3 and 4.2.

Step 6: Repeat of the evolutionary cycle: Return to Step 2 until a stopping criterion is satisfied. Usually, if the fitness of the particles is good enough or the number of iterations reaches the upper bound, the iteration is stopped.

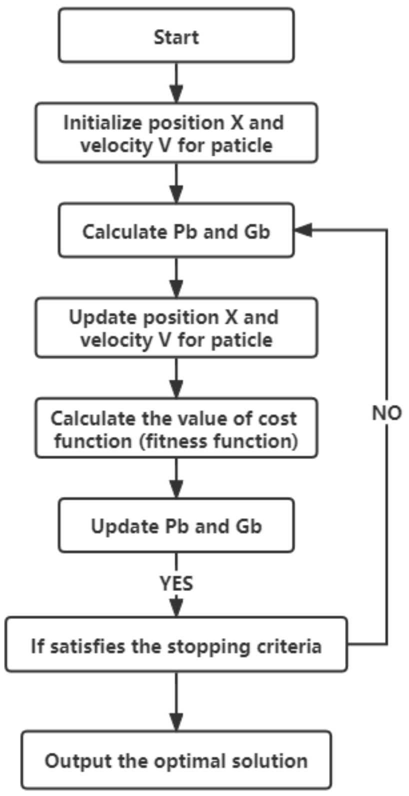

Start
Initialize position X and velocity V for paticle
Calculate Pb and Gb
Update position X and velocity V for paticle
Calculate the value of cost function (fitness function)
Update Pb and Gb
YES
NO
If satisfies the stopping criteria
Output the optimal solution

Figure 4.1: Flowchart of particle swarm optimization method

4.1.2 A Deeper Look into the Algorithm and its Meta-parameters

By analysing the scheme of the PSO algorithm and its updating rule, it can be found that there exist several meta-parameters which need to be decided in advance, such as the constants c_1 and c_2. Like other heuristic methods, these meta-parameters play an important rule in the algorithm and highly influence the performance and efficiency when solving optimization problem. Therefore, the influence of these parameters on the algorithm performances, including the convergence property and computation efficiency is concerned and discussed in this section. Moreover, the initialization process also plays an determinant role. In this process, a good estimate of the particles' positions and velocities could highly improve the efficiency and lead to fast convergence. Hence, the strategy for initialization is also talked about here.

As the flowchart of the PSO algorithm shown in Fig. 4.1, the initialization process is the first step in the algorithm. In this step, not only the particles' positions and velocities are initialized, an initial estimate of these two values are also required, which affect the convergence property of the solution. When talking about the initialization, there is a common way to complete this task, which is also a general agreement in the literature, that is initializing particles' positions which are spread uniformly over the whole search space. In a mathematical way, this initialization is:

$$x_{ij}(0) \sim U(x_{j.min}, x_{j.max}) \tag{4.4}$$

where i refers to the index of particle, j refers to the index of optimization variable, $x_{j.min}$ and $x_{j.max}$ are the lower and upper bound of the j_{th} optimization variable, and U represents the uniform distribution. Scientists have proved that this way of initialization ensures a good initial convergence property of the search space and increase the exploration property of the algorithm[52]. These two properties lead to a relatively fast convergence rate of the PSO algorithm.

For the initialization strategy for the particles' velocities, the same way as the position done, that is, setting the velocity using a uniform distribution so that the velocities cover the search space as uniformly as possible, is suggested by many scientists. However, this strategy does not have a good performance and leads to a severe diversity of the whole swarm. This makes sense since in the updating rule shown in Eq. 4.3, the inertia term dominates the velocity in the beginning, leads to large initial step sizes and finally destroys the convergence property. Moreover, larger step size makes particles violate the boundary of the search space more frequently and causes the divergence of algorithm, which is also called "velocity explosion". This is a general risk and highly influences the algorithm's performance. Therefore, imperative modification should be done to get rid of it. There are two popular approaches which are commonly used to solve this problem. The first one is called "velocity clamping". As its name shows, this approach utilizes a velocity threshold as the upper bound of velocity. By introducing this threshold, the velocities are limited in a range as:

$$\begin{aligned} &\text{if} \quad v_{ij}(t+1) > v_j^{max} && \text{then} \quad v_{ij}(t+1) = v_j^{max} \\ &\text{if} \quad v_{ij}(t+1) < -v_j^{max} && \text{then} \quad v_{ij}(t+1) = -v_j^{max} \end{aligned} \tag{4.5}$$

This strategy has a simple structure and many scientists have proved that it's a efficient way to solve common problems and provide a good trade-off between the exploration and exploitation, which leads to a good exploration ability[53]. However, it's not easy to get the proper velocity threshold, which is highly dependent on the specific problem. There does not exist a general criterion or even rule of thumb to choose accurate value of the threshold. This is the main drawback of the velocity clamping strategy. Even though this is a difficult task, using the interval of the search space to estimate the threshold is agreed on by some authors, which is represented as:

$$v_j^{max} = k \times \frac{x_j^{max} - x_j^{min}}{2}, \quad k \in (0,1] \tag{4.6}$$

The second way to overcome "velocity explosion" is introducing an inertia weight w. Since the domination of the inertia term finally leads to the divergence, adding a weighting coefficient to limit its influence is a good way to solve this problem. By adding the inertia weight, the contribution of velocity term in previous iteration is controlled and could be modulated as the iteration goes. Several approaches to define the inertia weight have been used by scientists and proved to improve the convergence property. Some common approaches for defining the inertia weight is reported in Table. 4.1.

Strategy	Definition of inertia weight
Constant inertia weight	$w(t) = w = constant$
Random inertia weight	$w(t) = 0.5 + \frac{r}{2}, \quad r \sim U(0,1)$
Linearly decreasing inertia weight	$w(t) = w_{max} - \frac{w_{max} - w_{min}}{t_{max}} \times t$

Table 4.1: Strategies for defining inertia weight

Like the "velocity clamping" strategy, implementation of the inertia weight still needs some pre-defined parameters which are dependent on the specific problem. However, the choice of inertia weight is not as tough as for the velocity threshold. Several scientists suggests that using $w_{max} = 0.9$ and $w_{min} = 0.4$ could achieve good performances generally[54].

Although several modification introduced above could overcome the shortcoming of "velocity explosion", the simplest way, that is setting the initial velocities to very small random values, is considered as a good choice. In this way, the risk of velocity explosion could be significantly reduced, meanwhile the exploration ability of the algorithm is still guaranteed[54].

Besides the initialization process, by observing the updating rule for particles' velocity in eq. 4.3, there are still two parameters which need to be decided in advance, which are two acceleration constants c_1 and c_2. These two parameters represents how much contribution the personal best and global best make to the particles' movement. The values of the constants c_1 and c_2 determine the extent to which the particles move towards the best position from its own experience and experience of all particles. In other words, these two parameters modulate the relative contributions of the social and cognitive terms[54]. Focusing on their influence on the movement of particles and the convergence property of the whole PSO algorithm, many investigations of these two parameters have been done. The result shows that, larger c_1 and c_2 we choose, larger oscillation frequency we get around the optimal solution. Moreover, smaller values lead to some inadequate result, i.e. sinusoidal patterns. In general, there is a common choice of c_1 and c_2 which has been proved to work well for most of the applications[55], that is shown as follows:

$$c_1 = c_2 = 2 \tag{4.7}$$

In summary, the classical particle swarm optimization method could provide good convergence property and efficiency after some modification such as introducing inertia weight and "velocity clamping". But the main drawback can not be solved thoroughly, that it, there always exist some problem-based parameters which are critical and hard to decide. To get rid of this disadvantages, similar approaches are proposed and a popular one which called Quantum-behaved Particle Swarm Optimization (QPSO) is introduced later.

4.2 Quantum-behaved Particle Swarm Optimization (QPSO)

In terms of classical PSO algorithm, there are two elements representing the state of each particle, the position vector $\mathbf{x_i}$ and velocity vector $\mathbf{v_i}$. These two elements depict the movement of each particle and determine particle's trajectory. Like the updating rule in Eq. 4.2, in PSO algorithm the particle moves in Newtonian mechanics. However, this mechanism does not work for the case in quantum mechanics. In quantum world, previous updating rule for particles' position is meaningless, since $\mathbf{x_i}$ and $\mathbf{v_i}$ of a particle cannot be determined simultaneously according to uncertainty principle. Therefore, if the quantum behavior of particles' movement is taken into consideration in a PSO system, the updating rule in classical PSO algorithm does not work and the particles have to move in a different way.

In the quantum form of classical PSO problem, which is named Quantum-behaved Particle Swarm Optimization (QPSO), instead of the position and velocity vector, the state of each particle is represented by a wave function $\psi(x,t)$ in Schrödinger equation[56]. With illustration of the Schrödinger equation, the dynamic behavior of the particle is totally different from that one considered in classical PSO algorithm, since $\mathbf{x_i}$ and $\mathbf{v_i}$ of a particle cannot be determined simultaneously according to uncertainty principle. In this context, a probability density function $|\psi(x,t)|^2$ is defined to quantify the probability of the particle's appearing in position $\mathbf{x_i}$.

According to the Monte Carlo method, the updating rule of the particles' position is shown in the following iterative equations[57]:

$$\begin{cases} \mathbf{x_i(t+1)} = \mathbf{p} + \beta \cdot |Mbest_i - \mathbf{x_i(t)}| \cdot \ln(1/u), & \text{if} \quad k \geq 0.5 \\ \mathbf{x_i(t+1)} = \mathbf{p} - \beta \cdot |Mbest_i - \mathbf{x_i(t)}| \cdot \ln(1/u), & \text{if} \quad k < 0.5 \end{cases} \tag{4.8}$$

where β is a design parameter called contraction–expansion coefficient[58], u and k are random values in the range $[0,1]$, usually achieved by some uniform probability distribution functions.

$$u \sim U(0,1) \quad k \sim U(0,1) \tag{4.9}$$

Since the $Mbest_i$ term represents the global point defined as the mean value of personal best positions of all particles found so far in the swarm , it is named as Mainstream Thought or Mean Best[59], and its mathematical form is shown as follows:

$$Mbest = \frac{1}{N}\sum_{i=1}^{N} P_{i,d}(t) \tag{4.10}$$

where $P_{i,d}$ represents the personal best position for the i_{th} particle in the swarm, $P_{g,d}$ is the global best position among all personal best positions and g is the index of the best particle.

Another term in the updating rule Eq. 4.8, $\mathbf{p}$, is called particles' local attractor, which helps to achieve the convergence of the QPSO algorithm by using the following equation:

$$\mathbf{p} = \frac{c_1 \cdot \mathbf{p_{i,d}} + c_2 \cdot \mathbf{p_{g,d}}}{c_1 + c_2} \tag{4.11}$$

The scheme of the classical QPSO algorithm is given by the following steps and the flowchart is shown in Fig. 4.2.

Step 1. Initialization of particles' positions and velocities in the whole swarm: Initialize a pre-defined population of particles randomly with their positions in the n-dimensional search space. A common way is to use some uniform probability distribution function to obtain these values.

Step 2. Evaluation of particle's fitness: Calculate the value of the objective function (fitness value) for each particle. In this part, the main goal is minimizing the objective function rather than maximizing.

Step 3. Updating of the global point: Calculate the $Mbest$ using Eq. 4.10.

Step 4. Comparison to fitness of $pbest$ (personal best position): Compare each particle's fitness with the fitness of the personal best position. If the value at current position is less than that at $pbest$, then set the current position as the $pbest$ position and update the fitness value of $pbest$ with the current fitness value.

Step 5. Comparison to fitness of $gbest$ (global best position): Compare the fitness of the current position with fitness of the population's overall best positions achieved before. If the value at current space is less than that at $gbest$, then reset the global best position as the current position and do the same for the fitness value.

Step 6. Updating of each particle's velocity and position: Update the particle's position $\mathbf{x_i(t+1)}$ according to Eqs. 4.8.

Step 7. Repeat of the evolutionary cycle: Return to Step 2 until a stopping criterion is satisfied. Usually, if the fitness of the particles is good enough or the number of iterations reaches the upper bound, the iteration is stopped.

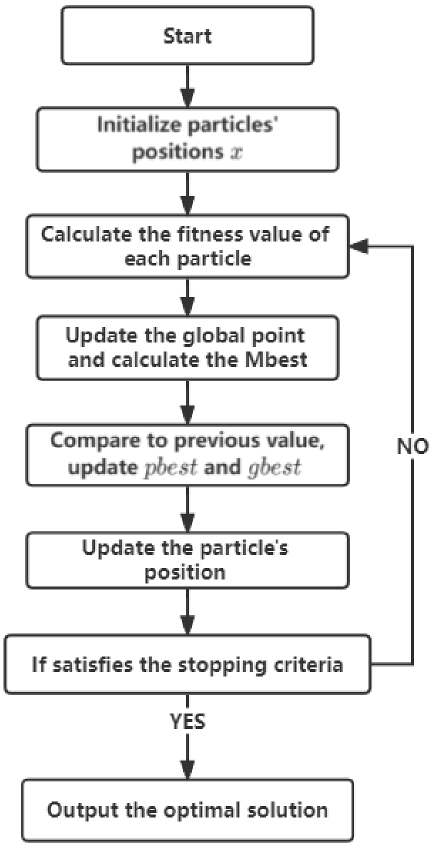

Start
Initialize particles' positions x
Calculate the fitness value of each particle
Update the global point and calculate the Mbest
Compare to previous value, update $pbest$ and $gbest$
Update the particle's position
If satisfies the stopping criteria
NO
YES
Output the optimal solution

Figure 4.2: Flowchart of Quantum-behaved Particle Swarm Optimization method

By observing the updating rule for the particle's positions shown in eq. 4.8, it can be found that there still exist some problem-based parameters including c_1, c_2 and β. For the parameter β, it can be chosen in the same way as for the inertia weight w, whose common strategies are shown in Table. 4.1. But it is still difficult to choose proper values for other two parameters. In order to get rid of this, a modified version of Quantum-behaved particle swarm optimization using Gaussian mutation is proposed by scientists and its details are introduced later.

4.3 Quantum-behaved Particle Swarm Optimization with Gaussian Mutation (G-QPSO)

In the last few years, various versions of QPSO have been proposed and used to solve practical optimization problems. In most of these applications, the uniform probability distribution is deployed to set parameters as random values, including c_1, c_2, u, and k in Eqs. 4.8. However, besides the uniform probability distribution, other ways of distribution are also used to generate random values for these parameters, such as Gaussian, Cauchy and exponential probability distribution. In this section, following the same line of study, a modified QPSO algorithm with mutation operator using Gaussian probability distribution is presented, which is named as Gaussian Quantum-behaved Particle Swarm Optimization (G-QPSO).

Compared with uniform probability distribution, Gaussian probability distribution has better effect on the choice of parameters because of its mathematical properties, that is, Gaussian distribution sequences have zero mean and unit variance. Applied to the stochastic coefficients in QPSO algorithm, this property results in a good trade-off between the probability of generating movement with small amplitudes around the current space and that with large amplitudes. The former is considered as fine tuning and the latter is considered as mutation, which may generate movement with large amplitude and make the particle escape from the local minimum.

In this content, firstly, random numbers are generated by using the absolute value of the Gaussian probability distribution with zero mean and unit variance $abs(N(0,1))$. The updating rule with QPSO approach combined with Gaussian mutation operator is presented as follows:

$$\begin{cases} \mathbf{x_i(t+1)} = \mathbf{p} + \beta \cdot |Mbest_i - \mathbf{x_i(t)}| \cdot \ln(1/u), & \text{if} \quad k \geq 0.5 \\ \mathbf{x_i(t+1)} = \mathbf{p} - \beta \cdot |Mbest_i - \mathbf{x_i(t)}| \cdot \ln(1/u), & \text{if} \quad k < 0.5 \end{cases} \tag{4.12}$$

where $G = abs(N(0,1))$, $k \sim U(0,1)$

Moreover, to avoid choosing parameters c_1 and c_2, a new definition of p without these two meta-parameters is proposed, by using random numbers instead of them. The new definition of p is shown in the following equation:

$$\mathbf{p} = \frac{G \cdot \mathbf{p_{i,d}} + g \cdot \mathbf{p_{g,d}}}{G + g} \tag{4.13}$$

where $g = abs(N(0,1))$, $G = abs(N(0,1))$, $k \sim U(0,1)$

Overall, the new updating rule of Gaussian Quantum-behaved Particle Swarm Optimization (G-QPSO) is shown in Eq. 4.8. It can be found that there is only one parameter β which need to be decided in advance. Since it can be treated as the inertia weight, there are several ways to choose it. Therefore, in the G-QPSO algorithm, the main drawbacks of PSO and QPSO are overcome. The tuning process of parameters is extremely shortened. Since the scheme of G-QPSO is quite similar to QPSO algorithm with a small change of the updating rule, it is not presented here.

Chapter 5

Problem Formulation and Algorithm Design

In chapter 3 and 4, the AUV's guidance system and proposed optimization methods, including PSO, QPSO and G-QPSO methods are introduced. In order to solve the path planning problem by using these methods and make the solution suitable for the AUV's guidance system, a proper mathematical modelling for the path planning problem is indispensable. Moreover, in consideration of the characteristics for path planning, some new strategies are added to the basic algorithm to improve efficiency.

In this section, firstly the problem formulation is presented, including the preliminary modelling for path planning problem, its simplification and modification as well as the final version of modelling. Later, the algorithm design is introduced. The new strategy to define the dimension of the problem, the design of objective function as well as the practical algorithms are presented in the second part.

5.1 Problem Formulation

As mentioned in chapter 4, the standard form of an optimization problem is shown as follows:

$$\begin{aligned} & \min_{\mathbf{x}\in\mathbb{R}^n} \quad \mathbf{f}(x) \\ s.t. \quad & c_i(x) = 0,\ i \in \mathcal{I} \\ & c_i(x) \geq 0,\ i \in \mathcal{E} \end{aligned} \tag{5.1}$$

where $\mathbf{x}$ is the optimization variable in a n-dimensional space, $\mathbf{f}(x)$ represents the objective function, which is also called cost function or fitness. In general, only one objective function is taken into consideration and $\mathbf{f}(x)$ changes to a scalar form $f(x)$, but when it comes to multiple objective functions, this form is still suitable. $c_i(x)$ is called constraints, including equality constraints with indices $i \in \mathcal{I}$ and inequality constraints with indices $i \in \mathcal{E}$.

With this standard form of optimization problem, which is called standard constrained optimization problem, the path planning problem could be modelled and transferred to this standard form according to our purposes.

5.1.1 Preliminary Modelling for Optimal Path Planning Problem

In the process of preliminary modelling, the first to do is define the optimization variables $\mathbf{x}$ and its search space $\mathbb{R}^n$. In order to fulfill the desired properties of AUV's guidance system, the desired output of the optimal path planner should be a sequence of waypoints. Therefore, an optimal path which consists of $n+1$ waypoints is defined as the form of output. Among these $n+1$ points, there are two known and pre-defined points, which are the start position and destination position of the AUV. In a 3D case, these 2 positions are defined as follows:

$$\mathbf{p}_0 = (x_0, y_0, z_0) \tag{5.2a}$$

$$\mathbf{p}_{n+1} = (x_{n+1}, y_{n+1}, z_{n+1}) \tag{5.2b}$$

Besides these 2 known points, the internal points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ need to be generated by solving the optimization problem. The number of the internal points, denoted by n, is a design parameter and need to be determined in advance. Thus, the desired output, which is the mathematical representation of the path is shown in Eq. 5.3.

$$\mathcal{P} = [\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n, \mathbf{p}_{n+1}] \tag{5.3}$$

For the search space $\mathbb{R}^n$, it is defined based on the specific environment and map information. In general, in a 3-dimensional environment for AUV, the search space is defined within a global upper and lower bound in 3 directions including north, east and down direction, which is shown as follows:

$$N: \quad x_i \in [x_{min}, x_{max}] \tag{5.4a}$$

$$E: \quad y_i \in [y_{min}, y_{max}] \tag{5.4b}$$

$$D: \quad z_i \in [z_{min}, z_{max}] \tag{5.4c}$$

Commonly, the lower bound in down direction z_{min} can be set as 0, which corresponds to the sea level and the upper bound could be set as $z_{seafloor}$, the depth of the seafloor.

After defining the optimization variables, the next thing, which is also the most important thing in problem formulation comes to the front, that is defining the objective functions. When it comes to the path planning problem for AUV, two aspects should be taken into consideration. One is efficiency, the other one is safety. In the aspect of efficiency, the total path length could be a good choice of objective function. By minimizing the total path length, the travelling time and energy consumption could reach minimum point with some assumption like constant velocity. Hence, a good efficiency could be achieved. In consideration of safety, a critical requirement is that there is no collision with obstacles during the travel. Therefore, a objective function for collision avoidance with obstacles is indispensable. Moreover, considering the physical property of the AUV, large turning angle during the travel should be avoided. This helps smooth the path and avoid wear and tear on the actuators. In the preliminary modelling of the path planning problem, these three objectives shown above are concerned and the mathematical definitions are presented later.

Objective Function I: Path Length

Given the mathematical representation of the candidate path shown in Eq. 5.3, the total length of the path which consists of n path segments and n unknown waypoints could be achieved by using the Pythagorean theorem in 3D as follows:

$$f_1(\mathcal{P}) = \sum_{k=0}^{n} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2 + (z_{k+1} - z_k)^2} \tag{5.5}$$

With this objective function $f_1(\mathcal{P})$ in Eq. 5.5, the optimization method could quantify the value of the total path length and candidate solutions with lower values of $f_1(\mathcal{P})$ are preferred.

Objective Function II: Safety Margin

In consideration of safety in underwater environment, the second objective function concerning collision avoidance with obstacles is proposed. The margin of safety, denoted by m,is pre-defined according to the safety requirement and physical property of the vehicles. Since the optimization method searches for the minimum point in the space, negative value is generated by this objective function if the distance between the candidate path and obstacles is greater than the safety margin. When the distance between the candidate path and obstacles is less than this safety margin, a positive penalty function is added to the objective function. The mathematical definition of the second objective function $f_2(\mathcal{P})$ is shown as follows:

$$f_2(\mathcal{P}) = \begin{cases} m - r_{min} & , \quad \text{if} \quad r_{min} > m \\ e^{r_{min} - m} & , \quad \text{otherwise} \end{cases} \tag{5.6}$$

where m is a pre-defined parameter according to the environment and AUV, r_{min} represents the minimum distance between the obstacle and path segments, whose mathematical representation is presented later. With this objective function, it can be found that the greater r_{min} is, the more negative value $f_2(\mathcal{P})$ generates. Hence, the optimization method could output safety path with negative value of $f_2(\mathcal{P})$.

As mentioned in chapter 3, after the map processing, the obstacles are classified to 2 types, circle/sphere and oriented bounding box (OBB) with known positions and other information. For the circle-shape obstacles in a 2D environment, whose graphics explanation is shown in Fig. 5.1, the minimum distance

r_{min} could be calculated as follows:

$$r_{min} = \min_{k=1...n} \frac{|(\mathbf{p}_{obs} - \mathbf{p}_k) \times (\mathbf{p}_{obs} - \mathbf{p}_{k+1})|}{|(\mathbf{p}_{k+1} - \mathbf{p}_k)|} \tag{5.7}$$

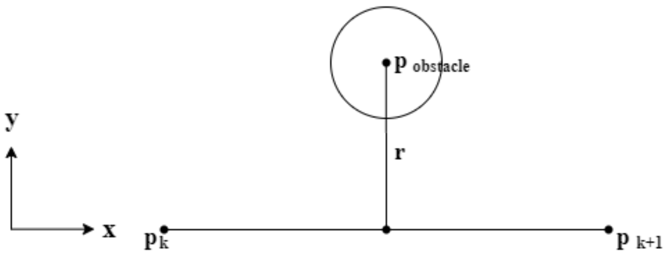


Figure 5.1: Graphics explanation of the distance between path and obstacle

Given the objective function as Eq. 5.6, If the distance is less than the safety margin, the objective function increases exponentially and works well as a penalty function. Hence, the optimization method could output a path with more negative value of $f_2(\mathcal{P})$.

The graphics illustration in Fig. 5.1 is considered in 2D environment. For the sphere-shape obstacles in 3-dimensional environment, this mathematical representation still works. However, for those obstacles with rectangular shape and modelled as oriented bounded box, this definition is not suitable. Thus, some modifications should be applied to solve this problem, which are introduced in section 5.1.2.

Objective Function III: Avoidance of sharp turning angle

Besides the path length and collision avoidance with obstacles, the smoothness of the path is also a consideration. In the underwater environment, the turning action for AUV can not be easily taken because of the dynamics and physical limitation on actuators. Moreover, sharp turns which will lead to wear and tear on the power equipment of AUV. Hence, the objective function about the turning angle is proposed.

Given four consecutive internal waypoints $\mathbf{p_k}$, $\mathbf{p_{k+1}}$, $\mathbf{p_{k+2}}$ and $\mathbf{p_{k+3}}$, as well as three path segments between two adjacent waypoints $\mathbf{l}_{k+1}$, $\mathbf{l}_{k+1}$ and $\mathbf{l}_{k+2}$, the turning angle, which is the path's changes in heading, is defined in Eq. 5.8. The graphics illustration is shown in Fig. 5.2.

$$\psi_k = \arccos\left(\frac{\mathbf{l}_k \cdot \mathbf{l}_{k+1}}{|\mathbf{l}_k||\mathbf{l}_{k+1}|}\right) \tag{5.8}$$

With the mathematical representation of the turning angle in Eq. 5.8, the third objective function $f_3(\mathcal{P})$ could be defined as the maximum turning angle during the travel. Since there are $n+2$ waypoints in the total path and $n+1$ path segments,

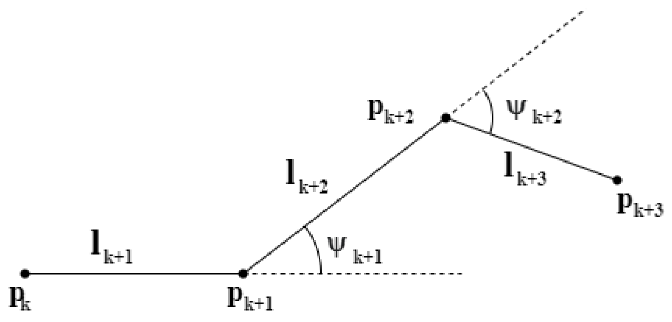


Figure 5.2: Graphics explanation of the turning angle between path segments

the number of turning angles between adjacent path segments is n. Therefore, the mathematical representation of $f_3(\mathcal{P})$ is expressed as follows:

$$f_3(\mathcal{P}) = \max_i \psi_i, \quad i = 1 \dots n \tag{5.9}$$

In summary, three objective functions are taken into consideration in the preliminary modelling for optimal path planning problem. The first one is the total path length, the second one is about safety margin in order to avoid collision with obstacles, and the third one is used to smooth the path and avoid sharp turns. The optimization variables is a sequence of waypoints, whose upper and lower bound are determined according to the specific problem. The preliminary mathematical model is shown as follows:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \quad f_i(x), \quad i = 1,2,3 \tag{5.10}$$

where

$$f_1(\mathcal{P}) = \sum_{k=0}^{n} \sqrt{(x_{k+1}-x_k)^2 + (y_{k+1}-y_k)^2 + (z_{k+1}-z_k)^2} \tag{5.11a}$$

$$f_2(\mathcal{P}) = \begin{cases} m - r_{min} & , \text{ if } \quad r_{min} > m \\ e^{r_{min}-m} & , \text{ otherwise} \end{cases} \tag{5.11b}$$

$$f_3(\mathcal{P}) = \max_i \psi_i, \quad i = 1 \dots n \tag{5.11c}$$

5.1.2 Simplification and Modification to Apply PSO Algorithm

In the preliminary modelling, three objective functions in Eqs. 5.11 are defined to optimize the path and the path planning problem is modelled as an unconstrained multi-objective optimization problem. However, with the classical PSO algorithm introduced in chapter 4, only unconstrained optimization problem with single objective function is taken into consideration and solved by deploying PSO method. Therefore, in order to use PSO method solving this path planning problem, some simplification and modification is necessary.

The first thing to do is reduce the number of objective functions. Since the main goal is to search for the optimal path with shortest path length, the objective function $f_1(\mathcal{P})$ is reserved and used as new objective function, which is expressed as follows:

$$f(\mathcal{P}) = \sum_{k=0}^{n} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2 + (z_{k+1} - z_k)^2} \tag{5.12}$$

where $\mathcal{P}$ represents the optimization variables expressed as Eq. 5.13.

$$\mathcal{P} = [\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_n, \mathbf{p}_{n+1}] \tag{5.13}$$

For the second objective function $f_2(\mathcal{P})$ about the collision avoidance, it can be simplified as an inequality constraint and then the number of objective function is reduced. Since the purpose of $f_2(\mathcal{P})$ is to ensure enough distance between the path segment and obstacles, the same effect could be achieved by using an inequality constraint $g_i(\mathcal{P})$ which makes the distance between path segment and obstacles not less than a certain value. The mathematical representation of $g(\mathcal{P})$ is expressed as follows:

$$g_i(\mathcal{P}) = r_i(\mathcal{P}) \geq c \quad , i = 1, \dots, n \tag{5.14}$$

where

$$r_i = \frac{|(\mathbf{p}_{obs} - \mathbf{p}_i) \times (\mathbf{p}_{obs} - \mathbf{p}_{i+1})|}{|(\mathbf{p}_{i+1} - \mathbf{p}_i)|} \tag{5.15}$$

The graphics explanation for Eq. 5.15 is shown in Fig. 5.7 and c is a pre-defined distance threshold according to the environment and physical characteristics of AUV.

Similarly, the third objective function $f_3(\mathcal{P})$ could also be transferred to an inequality constraint $q_i(\mathcal{P})$ which limits the turning angle using a specific threshold as the upper bound. The new inequality constraint $q_i(\mathcal{P})$ is defined as follows:

$$q_i(\mathcal{P}) = \psi_i \leq \psi_{threshold} \quad i = 1, \dots, n \tag{5.16}$$

where

$$\psi_i = \arccos\left(\frac{\mathbf{l}_i \cdot \mathbf{l}_{i+1}}{|\mathbf{l}_i||\mathbf{l}_{i+1}|}\right) \tag{5.17}$$

The graphics illustration for Eq. 5.17 is shown in Fig. 5.2 and $\psi_{threshold}$ is also a pre-defined parameter based on the turning ability of AUV and its physical limitation.

With these two simplifications, the preliminary model of the path planning problem has been changed to a constrained optimization problem with single objective function, which is expressed as follows:

$$\begin{aligned} & \min_{\mathcal{P}} \quad f(\mathcal{P}) \\ s.t. \quad & g_i(\mathcal{P}) \geq c \qquad\quad , i = 1, \dots, n \\ & q_i(\mathcal{P}) \leq \psi_{threshold} \ , i = 1, \dots, n \end{aligned} \tag{5.18}$$

When deploying PSO, QPSO and G-QPSO methods to solve this simplified path planning problem in Eqs. 5.18, an indispensable problem is how to handle the constraints since the classical PSO algorithm is only suitable for unconstrained optimization problem. Various methods have been proposed to handle these constraints within evolutionary algorithm and swarm intelligence approaches[60], and could be classified into several categories, including methods preserving solution feasibility, methods with penalty function and other hybrid approaches[61].

In this thesis, a penalty-based method is used to handle the inequality constraints in the simplified optimization problem. In penalty-based method, a penalty function is defined and added to the objective function when the constraints is not fulfilled, which corresponds to the unfeasible solutions. Since there are two inequality constraints, two penalty function p_{safety} and p_{turn} is proposed and the modified objective function is expressed as follows:

$$f_{mod}(\mathcal{P}) = \begin{cases} f(\mathcal{P}) & \text{, if } \mathcal{P} \in \mathcal{F} \\ f(\mathcal{P}) + p_{safe}(\mathcal{P}) & \text{, if } \mathcal{P} \in \mathcal{U}_1 \\ f(\mathcal{P}) + p_{turn}(\mathcal{P}) & \text{, if } \mathcal{P} \in \mathcal{U}_2 \\ f(\mathcal{P}) + p_{safe}(\mathcal{P}) + p_{turn}(\mathcal{P}) & \text{, if } \mathcal{P} \in \mathcal{U}_1 \cap \mathcal{U}_2 \end{cases} \tag{5.19}$$

where $\mathcal{F}$ is the feasible set, $\mathcal{U}_1$ is the unfeasible set for the first constraint about safety, $\mathcal{U}_2$ is the unfeasible set for the first constraint about turning angle. The penalty function p_{safety} and p_{turn} are equal to zero when the corresponding constraints are not violated by the candidate solution, and their mathematical representation are shown in Eqs. 5.20.

$$p_{safe}(\mathcal{P}) = \sum_{i=1}^{n} p^i_{safe}(\mathcal{P}) \tag{5.20a}$$

$$p_{turn}(\mathcal{P}) = \sum_{i=1}^{n} p^i_{turn}(\mathcal{P}) \tag{5.20b}$$

where

$$p^i_{safe}(\mathcal{P}) = \begin{cases} 0 & \text{, if } g_i(\mathcal{P}) - c \geq 0 \\ \beta \cdot (c - p_i(\mathcal{P})), & \text{otherwise} \end{cases} \tag{5.21}$$

$$p^i_{turn}(\mathcal{P}) = \begin{cases} 0 & \text{, if } q_i(\mathcal{P}) - \psi_{threshold} \leq 0 \\ \beta \cdot (\psi_{threshold} - q_i(\mathcal{P})), & \text{otherwise} \end{cases} \tag{5.22}$$

where β is a pre-defined parameter which is used to enlarge the violation of the constraints and quantify the penalty. In general, it is generated with a large value like 500, 1000. The expressions for $p_i(\mathcal{P})$ and $q_i(\mathcal{P}$ are shown in Eqs. 5.14 and 5.16.

5.1.3 Final Version of Modelling for Path Planning Problem

With the simplification and modification in chapter 5.1.2, the preliminary model with multiple objective functions has changed a lot. The new version is an unconstrained optimization model with single objective function using penalty-based method, which is suitable for PSO, QPSO and G-QPSO approches. The final version of modelling for the path planning problem is shown as follows:

$$\min_{\mathcal{P}} \quad f(\mathcal{P}) + p_{safe}(\mathcal{P}) + p_{turn}(\mathcal{P}) \tag{5.23}$$

where

$$\mathcal{P} = [\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_n, \mathbf{p}_{n+1}] \tag{5.24a}$$

$$f(\mathcal{P}) = \sum_{k=0}^{n} \sqrt{(x_{k+1}-x_k)^2 + (y_{k+1}-y_k)^2 + (z_{k+1}-z_k)^2} \tag{5.24b}$$

$$p_{safe}(\mathcal{P}) = \sum_{i=1}^{n} p_{safe}^{i}(\mathcal{P}) \tag{5.24c}$$

$$p_{turn}(\mathcal{P}) = \sum_{i=1}^{n} p_{turn}^{i}(\mathcal{P}) \tag{5.24d}$$

$$p_{safe}^{i}(\mathcal{P}) = \begin{cases} 0 & ,\text{if} \quad g_i(\mathcal{P}) - c \geq 0 \\ \beta \cdot (c - g_i(\mathcal{P})), & \text{otherwise} \end{cases} \tag{5.24e}$$

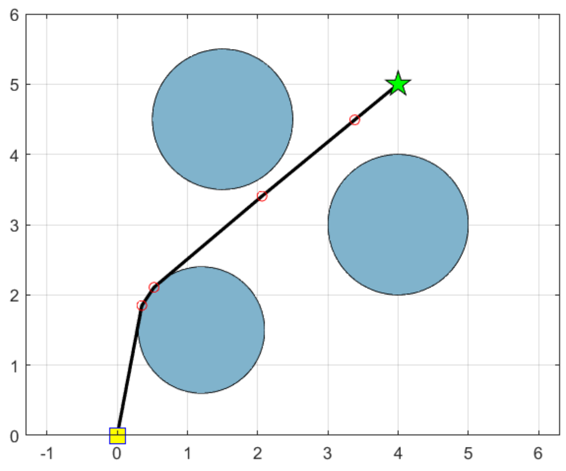
$$p_{turn}^{i}(\mathcal{P}) = \begin{cases} 0 & ,\text{if} \quad q_i(\mathcal{P}) - \psi_{threshold} \leq 0 \\ \beta \cdot (\psi_{threshold} - q_i(\mathcal{P})), & \text{otherwise} \end{cases} \tag{5.24f}$$

5.2 Algorithm Design

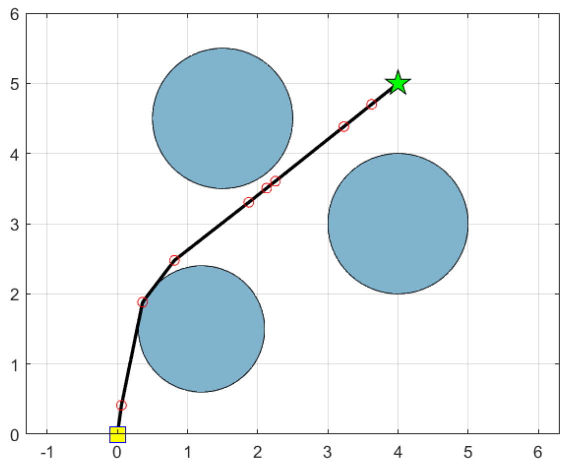
Given the modified model of the path planning problem in Eqs. 5.23, the classical PSO, QPSO and G-QPSO algorithm presented in chapter 4 could be deployed to solve this optimization problem. Based on the mathematical model, corresponding engineering model for MATLAB is designed, including the objective function and penalty function. In this part, the engineering model in MATLAB is presented and a strategy to define the dimension of the problem is introduced, as well as the pseudo-code of these algorithms.

5.2.1 Straight-line Method

As mentioned in section 5.1, the candidate path which consists of a sequence of waypoints including known starting and destination point and n interior points is defined as the optimization variable. The number of unknown points denoted by n, which is also the dimension of the optimization problem, need to be determined in advance. Since any path segment between two adjacent waypoints is a continuous path and could be discretized, there could be infinite number of points to be optimized to depict the best path. To solve this problem, in general, the number of unknown points is selected as smaller as possible. Thus, the optimization problem changes to a reduced dimension and the computational cost reduces as well. The examples shown in Fig. 5.3 corresponds to a possible path 8 interior points and 4 interior points. It can be found that these 2 paths both fulfill the requirement and the computation process for the path with 4 unknown points is clearly faster than that with 8 points. Therefore, a proper number of optimization variables is essential to the algorithm's performance and efficiency.



(a) Path with 4 interior points



(b) Path with 8 interior points

Figure 5.3: Comparison between paths with 4 and 8 interior points

In theory, if a path segment between 2 adjacent points collides with an obstacle in convex shape, setting one additional turning point in the path segment could avoid the collision. Thus, the maximum number of unknown points required to generate a path without collision with all obstacles is the number of obstacles in the search space. Any path with more unknown points is unnecessary and wasteful of computational resource. However, some obstacles could be far away from the path and impossible to collide with it. It's also a waste. Therefore, a simple method to determine the number of unknown points is proposed by drawing a straight line between the starting point and destination point. The number of unknown points is equal to the number of obstacles crossed by the straight line. The experience from other researchers shows that when the number of interior points is less than 3, some bad results such as path with much longer length and even unfeasible paths could appear[62]. Therefore, at least 4 unknown points is guaranteed and the pseudo-code is shown as follows:

Algorithm 1 Straight-line Method

```
1: Draw a straight line between the staring and destination point
2: if The straight line collides with obstacles then
3:     Determine the number of obstacles crossed by the line, denoted by $n_{obs}$
4:     if $n_{obs} < 4$ then
5:         $n = 4$
6:     else
7:         $n = n_{obs}$
8:     end if
9: end if
```

5.2.2 Design of Objective Functions

In the mathematical modelling of the path planning problem shown as Eqs. 5.23, the objective function consists of three parts, the total length of path $f(\mathcal{P})$ and two penalty function $p_{safe}(\mathcal{P})$ and $p_{turn}(\mathcal{P})$. In this section, the engineering models of them in MATLAB are presented.

For the first part, the total length of the path, it is calculated in the same way shown in section 5.1, by using the Pythagorean theorem. The expression in 2D case is shown as follows:

$$f(\mathcal{P}) = \sum_{k=0}^{n} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2} \tag{5.25}$$

In terms of the penalty function of safety $p_{safe}(\mathcal{P})$, there are some changes with engineering concern. Since the mathematical model of it is not suitable for obstacles in rectangular shape, another way to quantify the safety margin is used. Instead of calculating the distance between the obstacle and the path segment, linear interpolation is used to discrete the path segment and the penalty function is described by the relationship between the interpolation points and the obstacles.

For the obstacle in circle/sphere shape, it is easy to judge if the interpolation point is inside the obstacle by calculating the distance between it and the center of the obstacle. The engineering model of the penalty function is expressed in Eqs. 5.28.

$$p_{safe,1}(\mathcal{P}) = \sum_{i=1}^{n}\sum_{j=1}^{m} p_{safe,1}^{i,j}(\mathcal{P}) \tag{5.26}$$

where i is the index of the path segment, j is the index of the interpolation point, and

$$p_{safe,1}^{i,j}(\mathcal{P}) = \begin{cases} 0 & \text{, if} \quad g_{i,j}(\mathcal{P}) - c \geq 0 \\ \beta \cdot (c - g_i(\mathcal{P})), & \text{otherwise} \end{cases} \tag{5.27a}$$

$$g_{i,j}(\mathcal{P}) = \sqrt{(x_{i,j} - x_{obs})^2 + (y_{i,j} - y_{obs})^2} \tag{5.27b}$$

The piecewise-defined function in Eqs. 5.29 could expressed by a maximum function and the new form is shown as follows:

$$p_{safe,1}^{i,j}(\mathcal{P}) = \beta \cdot \max\left(0,\, 1 - \frac{g_{i,j}(\mathcal{P})}{c}\right) \tag{5.27c}$$

For the obstacles in rectangular shape, the model presented above is no longer applicable. There are several ways to judge if a certain point is inside a rectangular, such as ray casting. In this thesis, the judgement is based on the geometric relationship between the interpolation point and two parallel sides of the rectangular.

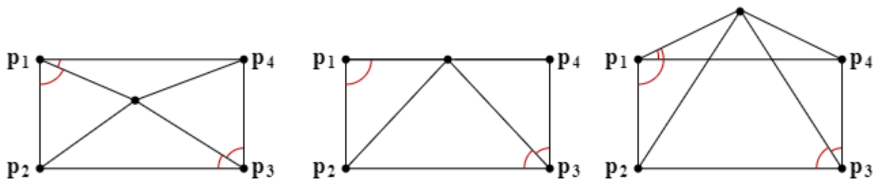


Figure 5.4: Graphics illustration for geometric relationship between the point and the rectangular

As shown in Fig. 5.4, it can be found that if the point is inside the rectangular, the angles marked by red curve are all acute angles. If the point is on the side of rectangular or outside, some of these angles become equal to, even greater than $\pi/2$. The mathematical expression of this criterion is shown as follows:

$$p_{safe,2}(\mathcal{P}) = \sum_{i=1}^{n}\sum_{j=1}^{m} p_{safe,2}^{i,j}(\mathcal{P}) \tag{5.28}$$

where i is the index of the path segment, j is the index of the interpolation point, and

$$p^{i,j}_{safe,2}(\mathcal{P}) = \begin{cases} 0 & ,\text{if} \quad \phi^k_{i,j} \in (0, \frac{\pi}{2}) \qquad k = 1, 2, 3, 4 \\ \beta \cdot (c - g_i(\mathcal{P})), & \text{otherwise} \end{cases} \tag{5.29a}$$

$$g_{i,j}(\mathcal{P}) = \sqrt{(x_{i,j} - x_{obs})^2 + (y_{i,j} - y_{obs})^2} \tag{5.29b}$$

The angle could be calculated as the same way in Eq. 5.8 and the final expression for the penalty function is shown as follows:

$$p_{safe}(\mathcal{P}) = p_{safe,1}(\mathcal{P}) + p_{safe,2}(\mathcal{P}) \tag{5.30}$$

For the second penalty function about the turning angle, the engineering model of it is quite similar to $p_{safe,1}(\mathcal{P})$ and shown as follows:

$$p_{turn}(\mathcal{P}) = \sum_{i=1}^{n} p^i_{turn}(\mathcal{P}) \tag{5.31a}$$

$$p^i_{turn}(\mathcal{P}) = \begin{cases} 0 & ,\text{if} \quad q_i(\mathcal{P}) - \psi_{threshold} \leq 0 \\ \beta \cdot (\psi_{threshold} - q_i(\mathcal{P})), & \text{otherwise} \end{cases} \tag{5.31b}$$

$$q_i(\mathcal{P}) = \arccos\left(\frac{\mathbf{l}_k \cdot \mathbf{l}_{k+1}}{|\mathbf{l}_k||\mathbf{l}_{k+1}|}\right) \tag{5.31c}$$

The piecewise-defined function in Eqs. 5.31 could expressed by a maximum function and the new form is shown as follows:

$$p^i_{turn}(\mathcal{P}) = \beta \cdot \max\left(0, \frac{q_i(\mathcal{P})}{\psi_{threshold}} - 1\right) \tag{5.31d}$$

In summary, three parts of the objective function are modified in the consideration of engineering aspect. The new model solves some previous problems such as unfitness to obstacles in rectangular shape. Also, with the discretization of the path segments,it is easy to deploy these functions into MATLAB code. The practical algorithm and code work are introduced later.

5.2.3 The PSO Algorithm for Path Planning Problem

As mentioned in chapter 4, the first step in PSO algorithm is initialization of the swarm. In the classical PSO algorithm, the swarm is initialized by using random distribution function. Since the straight line method has drawn a candidate path which is a line between starting and destination point with the shortest path length, it could be used as the initial values of the swarm by linear interpolation along this path. The mathematical expression of the initialization is:

$$\mathbf{P}_i = \mathbf{P}_0 + i \cdot \frac{\mathbf{P}_{n+1} - \mathbf{P}_0}{n}, \; i = 1, 2, ..., n \tag{5.32}$$

Algorithm 2 Proposed PSO Algorithm

```
1: Define the dimension of problem $n$ with Straight-line method in Algorithm 1
2: Initialize the swarm with eqs. 5.32 and 5.33
3: Evaluate the fitness of the swarm and set as the initial $pbest$ and $gbest$
4: for k = 1:K do
5:   for i = 1:N do
6:     Update the velocity $v$ by using eq. 4.3
7:     Check if the velocity violates the velocity bounds
8:     Update the particle's position with eq. 4.2
9:     Check if the new position violates the position bounds
10:    Evaluate the fitness value $F(\mathcal{P}_i)$ of the particle in new position with eqs. 5.23
11:    if $F(\mathcal{P}_i) < pbest_i$ then
12:      $pbest_i = F(\mathcal{P}_i)$
13:    end if
14:    if $F(\mathcal{P}_i) < gbest$ then
15:      $gbest = F(\mathcal{P}_i)$
16:    end if
17:  end for
18: end for
```

For the initialization of the velocity v, it is equal to zero.

$$\mathbf{v}_i = 0\,,\; i = 1, 2, ..., n \tag{5.33}$$

The pseudo-code of the proposed PSO algorithm is presented in Algorithm 2.

In this algorithm, K is the maximum iterations and N is the population of the swarm. There are many parameters such as c_1, c_2, w and the velocity bound v_{min}, v_{max} which need to be defined in advance.

5.2.4 The QPSO and G-QPSO Algorithm for Path Planning Problem

The initialization process in QPSO and G-QPSO algorithm is the same as the one in PSO algorithm. Since the difference between QPSO and G-QPSO algorithm is quite small, only the pseudo-code of QPSO is presented and the difference is added. The pseudo-code of the proposed algorithm is presented in Algorithm 3.

Algorithm 3 Proposed QPSO Algorithm

```
1: Define the dimension of problem $n$ with Straight-line method in Algorithm 1
2: Initialize the swarm with eqs. 5.32 and 5.33
3: Evaluate the fitness of the swarm and set as the initial $pbest$ and $gbest$
4: for k = 1:K do
5:    for i = 1:N do
6:       Evaluate $Mbest$ by using eq. 4.10
7:    end for
8:    for i = 1:N do
9:       Calculate $\beta$ by using eq. 4.3 and generate random number $k$
10:      if $k < 0.5$ then
11:         Update the particle's position with positive sign in eq. 4.8(G-QPSO
            with eq. 4.12)
12:      else
13:         Update the particle's position with negative sign in eq. 4.8(G-QPSO
            with eq. 4.12)
14:      end if
15:      Check if the new position violates the position bounds
16:      Evaluate the fitness value $F(\mathcal{P}_i)$ of the particle in new position with eqs.
         5.23
17:      if $F(\mathcal{P}_i) < pbest_i$ then
18:         $pbest_i = F(\mathcal{P}_i)$
19:      end if
20:      if $F(\mathcal{P}_i) < gbest$ then
21:         $gbest = F(\mathcal{P}_i)$
22:      end if
23:   end for
24: end for
```

Chapter 6

Results

This chapter contains the results from different phases of the development of an optimal path planner based on swarm evolutionary strategy, including the classical PSO algorithm, QPSO algorithm and G-QPSO algorithm. Note that the practical path planning problem is represented by the mathematical model in Eqs. 5.23 and engineering model in section 5.2.2, which is solved by deploying the PSO/QPSO/G-QPSO algorithm presented in section 5.2. Overall, the results can be divided into 3 parts with three different algorithms and a comparison part is put at the end.

1. Results of the Matlab-coded PSO-based path planner, with the tuning process of c_1 and c_2, as well as the comparison between strategies about inertia weight w, including both 2D case and 3D case.
2. Results of the Matlab-coded QPSO-based path planner, with the tuning process of c_1 and c_2, only 2D case is considered.
3. Results of the Matlab-coded GQPSO-based path planner, only 2D case is considered and no tuning process is done with G-QPSO algorithm.
4. The comparison among the results achieved before in a 2D case.

6.1 Path Planning with PSO algorithm

In this section, the tuning process of c_1 and c_2 is shown and different strategies of inertia weight w are tested. After choosing the best c_1, c_2 and w in 2D environment with circle-shape obstacles, the path planner is tested in a complex environment with both circle-shape and OBB-shape obstacles. Finally, a 3D case is done and the performance in 3D environment is tested.

In all results shown in this section, the general PSO parameter setting are as given in Table 6.1.

Parameter	**Symbol**	**Value**
Population size	N	150
Number of iterations	K	150
Coefficient for velocity bound	k	0.2
Number of segments between two adjacent points	n	300
Safety threshold for obstacle expansion	ϵ	5.52 (m)

Table 6.1: Parameter setting for PSO algorithm

The map information in a 2D environment including the sizes and positions of the obstacles, the starting point and destination point is shown in Table 6.2.

Parameter	**Symbol**	**Value**	**Unit**
Starting point	(x_0, y_0)	$(0,0)$	$[m]$
Destination point	(x_{n+1}, y_{n+1})	$(80,100)$	$[m]$
Obstacle 1 position	(x^1_{obs}, y^1_{obs})	$(30,90)$	$[m]$
Obstacle 2 position	(x^2_{obs}, y^2_{obs})	$(80,60)$	$[m]$
Obstacle 3 position	(x^3_{obs}, y^3_{obs})	$(24,30)$	$[m]$
Obstacle radius	$(r^1_{obs}, r^2_{obs}, r^3_{obs})$	$(20,20,18)$	$[m]$

Table 6.2: Map information in a simple 2D environment

For a complex scenario with OBB-shape obstacles, the map information is shown in Table 6.3.

Parameter	**Symbol**	**Value**	**Unit**
Starting point	(x_0, y_0)	$(0,0)$	$[m]$
Destination point	(x_{n+1}, y_{n+1})	$(80,100)$	$[m]$
Obstacle 1 position	(x^1_{obs}, y^1_{obs})	$(30,90)$	$[m]$
Obstacle 2 position	(x^2_{obs}, y^2_{obs})	$(80,60)$	$[m]$
Obstacle 3 position	(x^3_{obs}, y^3_{obs})	$(24,30)$	$[m]$
Obstacle radius	$(r^1_{obs}, r^2_{obs}, r^3_{obs})$	$(20,20,18)$	$[m]$
Obstacle 4 position in OBB shape	$x_{obb}, y_{obb}, \phi_{obb}$	$(38,10,-\frac{\pi}{6})$	$[m,m,rad]$
Obstacle 4 length and width	(a,b)	$(10,10)$	$[m]$

Table 6.3: Map information in a complex 2D scenario with OBB-shape obstacles

As for the 3-dimensional scenario, only sphere-shape obstacles are taken into account and the map information is shown in Table 6.4.

Parameter	**Symbol**	**Value**	**Unit**
Starting point	(x_0, y_0, z_0)	$(0,0,0)$	$[m]$
Destination point	$(x_{n+1}, y_{n+1}, z_{n+1})$	$(100,100,100)$	$[m]$
Obstacle 1 position	$(x^1_{obs}, y^1_{obs}, z^1_{obs})$	$(15,45,15)$	$[m]$
Obstacle 2 position	$(x^2_{obs}, y^2_{obs}, z^2_{obs})$	$(40,30,30)$	$[m]$
Obstacle 3 position	$(x^3_{obs}, y^3_{obs}, z^3_{obs})$	$(12,15,20)$	$[m]$
Obstacle 4 position	$(x^4_{obs}, y^4_{obs}, z^4_{obs})$	$(60,70,70)$	$[m]$
Obstacle 5 position	$(x^5_{obs}, y^5_{obs}, z^5_{obs})$	$(50,60,50)$	$[m]$
Obstacle radius	$(r^1_{obs}, r^2_{obs}, r^3_{obs}, r^4_{obs}, r^5_{obs})$	$(15,15,8,10,15)$	$[m]$

Table 6.4: Map information in 3D scenario with sphere-shape obstacles

6.1.1 2D Path Planning with Different c_1 and c_2

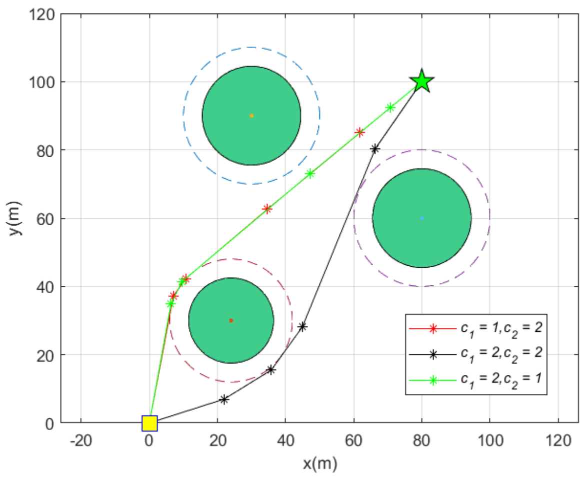
Firstly, three pairs of c_1 and c_2 are applied to the PSO algorithm shown in Algorithm 2. The results are shown in Fig. 6.1.

$$1.\quad c_1 = 1, \quad c_2 = 2 \tag{6.1}$$

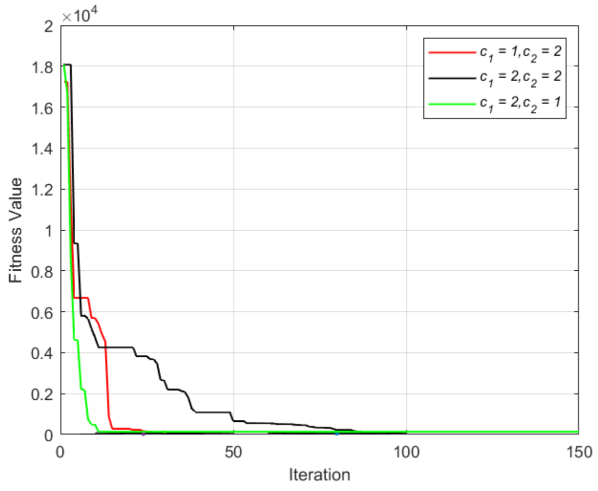
$$2.\quad c_1 = 2, \quad c_2 = 2 \tag{6.2}$$

$$3.\quad c_1 = 2, \quad c_1 = 1 \tag{6.3}$$

The information of obstacles is shown in Table 6.2.



(a) Optimal paths generated by PSO algorithm with different pairs of c_1 and c_2



(b) The fitness value during iterations by PSO algorithm with different pairs of c_1 and c_2

Figure 6.1: Performance of the PSO algorithm with different pair of c_1 and c_2

Figure 6.1 shows the optimal paths generated by the PSO-based path planner with different pairs of c_1 and c_2. The changes of the fitness value during the iterations are also shown, which could be used to check its convergence property. Detailed information about the optimal paths, including the position of waypoints and total path length is shown in Table 6.5 and 6.6

	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$
$(c_1, c_2) = (1,2)$	$(7.05, 37.20)$	$(10.63, 42.26)$	$(34.65, 62.60)$	$(61.78, 84.98)$
$(c_1, c_2) = (2,2)$	$(21.92, 6.98)$	$(35.78, 15.48)$	$(44.90, 28.11)$	$(66.35, 80.37)$
$(c_1, c_2) = (2,1)$	$(6.38, 35.03)$	$(9.58, 41.39)$	$(47.16, 73.02)$	$(70.61, 92.27)$

Table 6.5: Positions of waypoints generated by three pairs of c_1 and c_2

	$(c_1, c_2) = (1,2)$	$(c_1, c_2) = (2,2)$	$(c_1, c_2) = (2,1)$
Total path length $[m]$	134.32	135.23	134.34

Table 6.6: Total length of the paths generated by three pairs of c_1 and c_2

6.1.2 2D Path Planning with Different Definitions of Inertia Weight

As mentioned in chapter 4, there are several methods to define the inertia weight w, which are shown in Table 4.1. In this part, two strategies are chosen and tested, that is, constant inertia weight and linearly decreasing inertia weight, whose mathematical representation is expressed as follows:

$$\text{Constant inertia weight:} \quad w(t) = w \tag{6.4}$$

$$\text{Linearly decreasing inertia weight:} \quad w(t) = w_{max} - \frac{w_{max} - w_{min}}{t_{max}} \times t \tag{6.5}$$

where t is the index of the iteration and the tested parameters are shown in Table 6.7.

w	w_{max}	w_{min}	t_{max}
0.65	0.65	0.2	150

Table 6.7: tested parameters of inertia weight

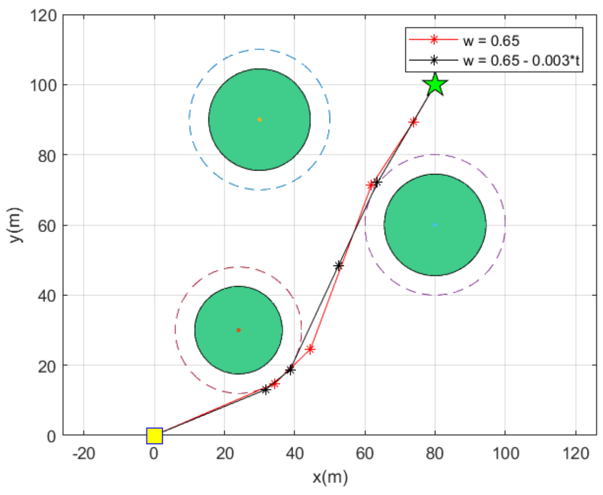
The results with these 2 strategies of inertia weight are shown in Fig. 6.2. The detailed information about the optimal paths, including the position of waypoints and total path length is shown in Table 6.8 and 6.9

	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$
constant w	$(34.39, 14.84)$	$(44.50, 24.64)$	$(61.81, 71.22)$	$(73.71, 89.21)$
$w(t)\, in\, Eq.6.5$	$(31.81, 13.11)$	$(38.71, 18.78)$	$(52.50, 48.46)$	$(63.56, 72.07)$

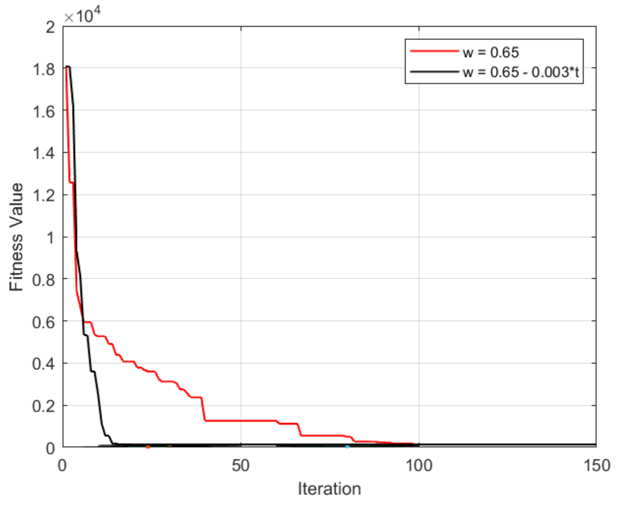
Table 6.8: Positions of waypoints generated by PSO algorithm with different strategies of inertia weight

	constant w	linearly decreased w
Total path length $[m]$	135.27	134.54

Table 6.9: Total length of the paths generated by PSO algorithm with different strategies of inertia weight



(a) Optimal paths generated by PSO algorithm with different definition of inertia weight w



(b) The fitness value during iterations by PSO algorithm with different definition of inertia weight w

Figure 6.2: Performance of the PSO algorithm with different strategies for inertia weight w

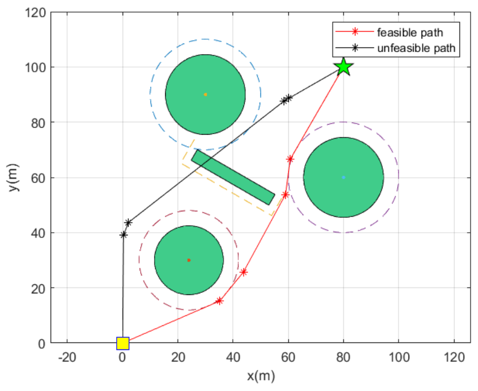
Figure 6.2 shows the optimal paths generated by the PSO-based path planner with different strategies of inertia weight w. The changes of the fitness value during the iterations are also shown, which could be used to check its convergence property. The discussion is presented later in chapter 7.

6.1.3 2D Path Planning in Complex Environment with OBB-shape Obstacles

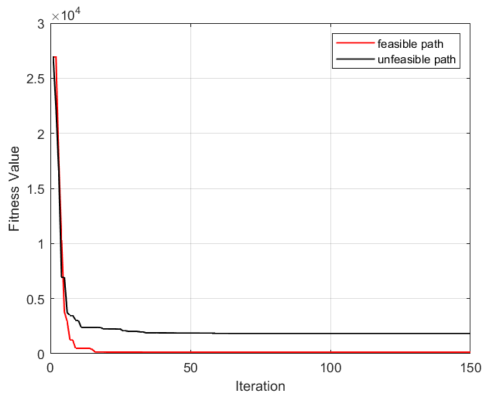
Based on the results of previous two tests, the optimization parameter used in this section is set as the one which has best performance in the former tests and shown in Table 6.10. The map information is presented in Table 6.3.

c_1	c_2	w_{max}	w_{min}	t_{max}
1	2	0.65	0.20	150

Table 6.10: Optimization Parameter in complex environment



(a) Optimal path and unfeasible path generated by PSO algorithm in complex environment with OBB-shape obstacles



(b) The fitness value during iterations by PSO algorithm in complex environment with OBB-shape obstacles

Figure 6.3: Performance of the PSO algorithm in complex environment with OBB-shape obstacles

Figure 6.3 contains two paths generated by the PSO algorithm, one in red is feasible and the other one in black is not feasible and violates the safety constraint defined in chapter 5. The reason about this will be introduced in chapter 7. The change of the fitness value is also shown in this figure. The detailed information about the optimal paths, including the position of waypoints and total path length is shown in Table 6.11 and 6.12.

	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$
feasible path	$(35.21, 15.27)$	$(43.77, 25.77)$	$(59.00, 53.81)$	$(60.70, 66.56)$
unfeasible path	$(0.345, 39.18)$	$(2.185, 43.73)$	$(58.32, 87.53)$	$(60.17, 88.62)$

Table 6.11: Positions of waypoints in feasible and unfeasible path generated by PSO algorithm

	feasible path	unfeasible path
Fitness value	135.26	1849.3

Table 6.12: The fitness value of the paths generated by PSO algorithm, including feasible and unfeasible path

6.1.4 3D Path Planning with PSO Algorithm

Besides the tests in 2-dimensional scenario, the performance of PSO-based path planner in 3-dimensional scenario is tested and analyzed. The map information is shown in Table 6.4. Based on the results of previous tests, the optimization parameter used in this section is set as the one which has best performance in the former tests and shown in Table 6.13.

The optimal path generated by the PSO-based path planner in 3D scenario is shown in Fig. 6.4. The views of the path in North-East (NE) Plane and East-Down (ED) Plane are shown in Fig. 6.5 and 6.6.

c_1	c_2	w_{max}	w_{min}	t_{max}
1	2	0.65	0.20	150

Table 6.13: Optimization Parameter in 3D scenario

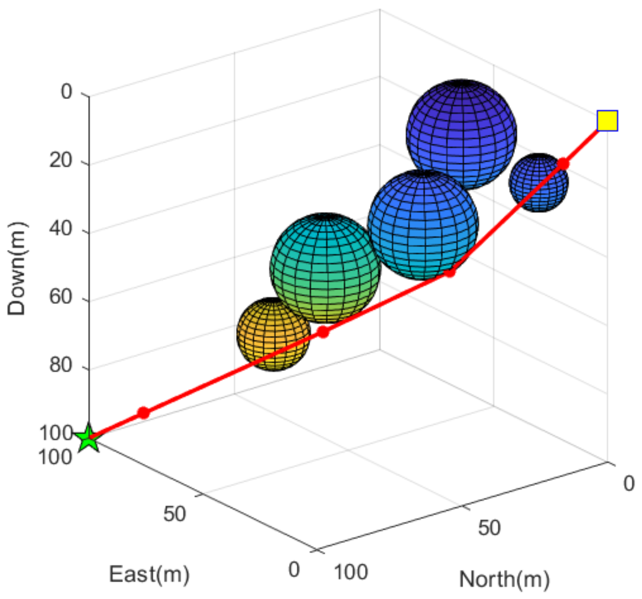


Figure 6.4: Optimal path generated by PSO-based path planner in 3D scenario

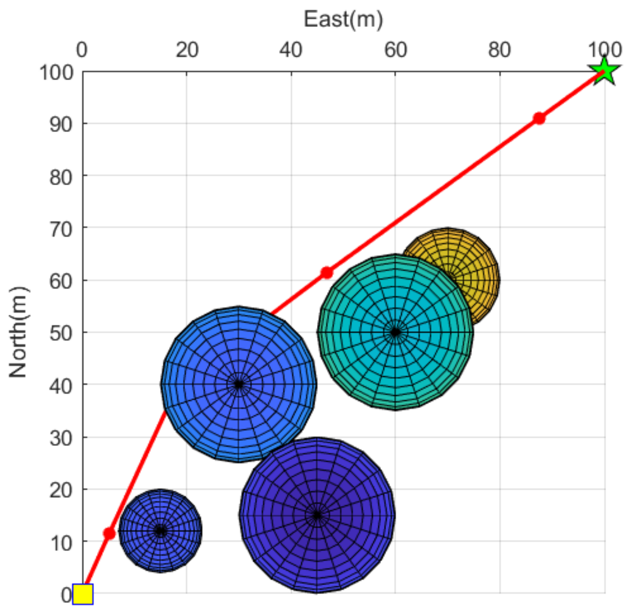


Figure 6.5: Optimal path generated by PSO-based path planner in the NE-plane

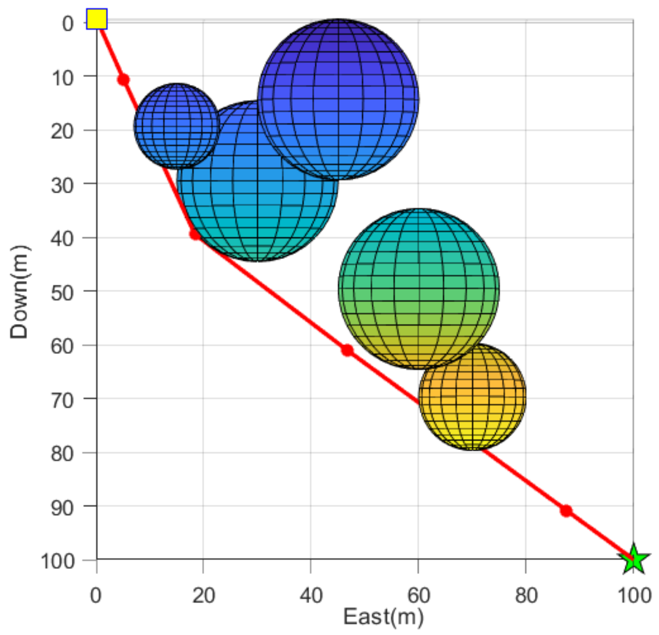


Figure 6.6: Optimal path generated by PSO-based path planner in the DE-plane

The change of the fitness value is also shown in Fig. 6.7. The detailed information about the optimal paths, including the position of waypoints is shown in Table 6.14.

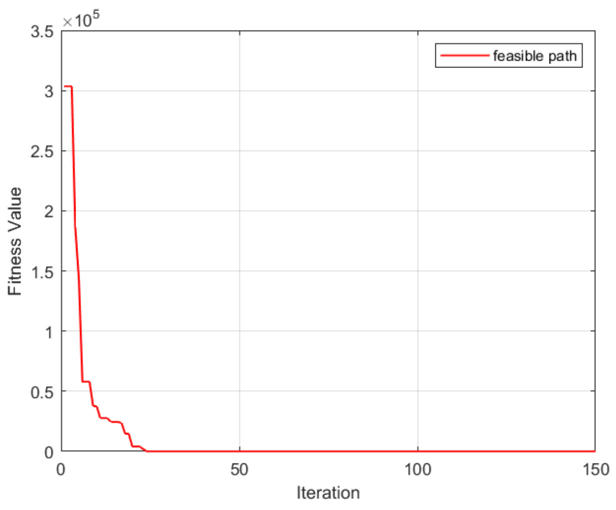


Figure 6.7: The fitness value during iterations by PSO algorithm in 3D environment

	$x\,[m]$	$y\,[m]$	$z\,[m]$
$\mathbf{P}_1$	11.40	5.131	11.40
$\mathbf{P}_2$	39.88	18.35	39.88
$\mathbf{P}_3$	61.36	46.74	61.36
$\mathbf{P}_4$	90.90	87.46	90.90

Table 6.14: Positions of waypoints generated by PSO algorithm in 3D scenario

The fitness value of the optimal path, which is also the total path length of a feasible path is shown as follows:

$$\mathcal{F}(\mathcal{P}) = L_{total} = 177.18 \quad [m] \tag{6.6}$$

6.2 Path Planning with QPSO Algorithm

In this section, the tuning process of c_1 and c_2 is shown and different strategies of choosing c_1 and c_2 are tested. After choosing the best c_1, c_2 in 2D environment with circle-shape obstacles, the path planner is tested in a complex environment with both circle-shape and OBB-shape obstacles. Finally, a 3D case is done and the performance in 3D environment is tested.

The QPSO-based path planner is based on the QPSO algorithm introduced in chapter 5, the Algorithm 3. The parameter β in the algorithm is defined as the same way of inertia weight w, and its mathematical representation is expressed as follows:

$$\beta = w_{max} - \frac{w_{max} - w_{min}}{t_{max}} \times t \tag{6.7}$$

where the parameters are shown in Table 6.15.

w_{max}	w_{min}	t_{max}
0.65	0.20	150

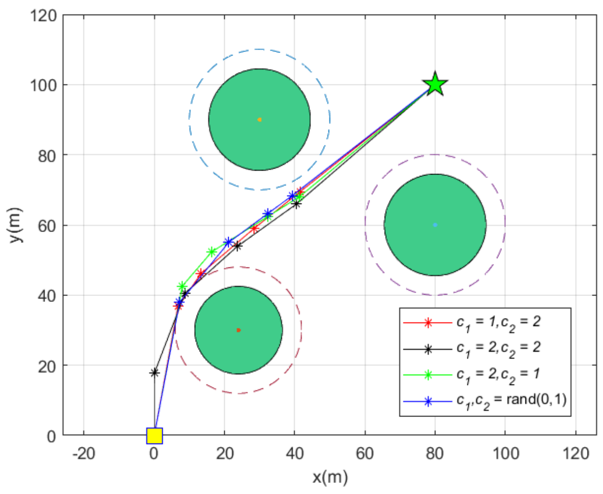
Table 6.15: Optimization Parameter for β in QPSO algorithm

6.2.1 2D Path Planning with Different c_1 and c_2

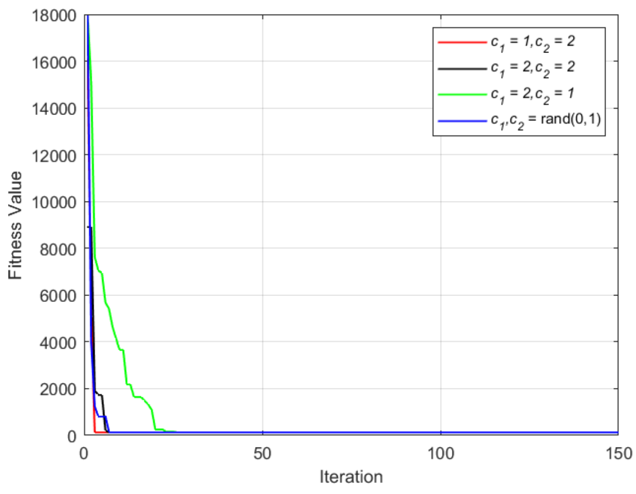
As mentioned in chapter 4, the parameters c_1 and c_2 modulate the relative contribution of the social and cognitive terms to finding the optimum. Similarly, three different pairs of c_1 and c_2 are tested. In addition, in order to simplify the tuning process, c_1 and c_2 are set in the range of $(0,1)$ and their results are presented and compared with previous ones. The parameter setting is shown as follows:

1. $c_1 = 1, \quad c_2 = 2$ (6.8)
2. $c_1 = 2, \quad c_2 = 2$ (6.9)
3. $c_1 = 2, \quad c_1 = 1$ (6.10)
4. $c_1 = rand(0,1), \quad c_2 = rand(0,1)$ (6.11)

In a simple 2D environment with circle-shape obstacles, whose information is shown in Table. 6.2, the optimal paths generated by QPSO-based path planner with different pairs of c_1 and c_2 are shown in Fig. 6.8, as well as the fitness value during the iterations.



(a) Optimal paths generated by QPSO algorithm with different pairs of c_1 and c_2



(b) The fitness value during iterations by QPSO algorithm with different pairs of c_1 and c_2

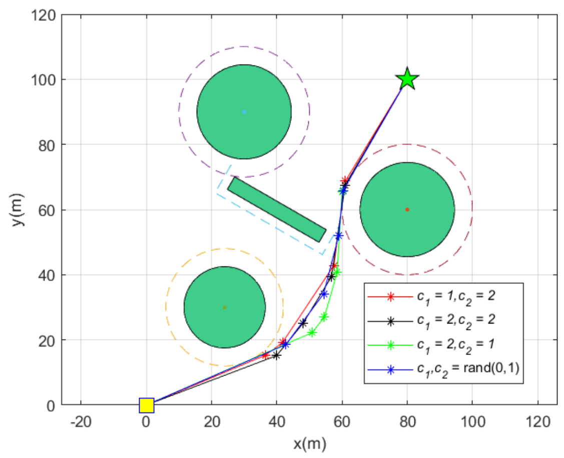
Figure 6.8: Performance of the QPSO algorithm with different pair of c_1 and c_2 in simple 2D scenario

The fitness values, which is equal to the path length for feasible paths, of the paths shown in Fig. 6.8 is given in Table 6.16.

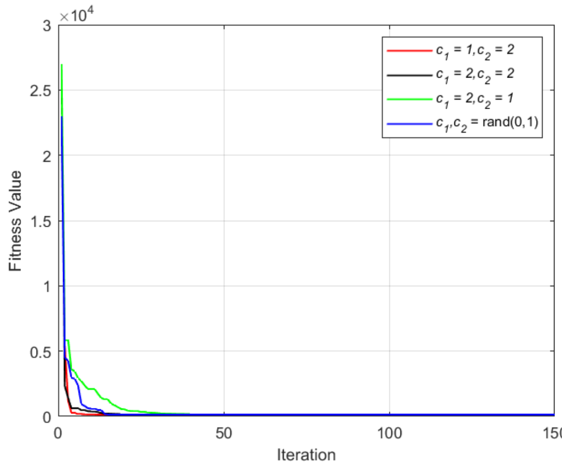
(c_1, c_2)	$(1,2)$	$(2,2)$	$(2,1)$	$rand(1,2)$
L_{total}	134.54	134.98	135.81	134.70

Table 6.16: Total length of the paths with different (c_1, c_2)

In a complex 2D environment with OBB-shape obstacles, whose information is shown in Table. 6.3, the optimal paths generated by QPSO-based path planner with different pairs of c_1 and c_2 are shown in Fig. 6.9, as well as the fitness value during the iterations. The final fitness values are shown in Table 6.17.



(a) Optimal paths generated by QPSO algorithm with different pairs of c_1 and c_2



(b) The fitness value during iterations by QPSO algorithm with different pairs of c_1 and c_2

Figure 6.9: Performance of the QPSO algorithm with different pair of c_1 and c_2 in complex 2D scenario

(c_1, c_2)	$(1,2)$	$(2,2)$	$(2,1)$	$rand(1,2)$
L_{total}	137.16	144.13	151.34	137.77

Table 6.17: Total length of the paths with different (c_1, c_2)

6.2.2 3D Path Planning with QPSO algorithm

In a simple 3D environment with sphere-shape obstacles, whose information is shown in Table. 6.4, the optimal paths generated by QPSO-based path planner with different pairs of c_1 and c_2 are shown in Fig. 6.10, 6.11 and 6.12. The changes of fitness value during the iterations are shown in Fig. 6.13.

c_1	c_2	w_{max}	w_{min}	t_{max}
rand(0,1)	rand(0,1)	0.65	0.20	150

Table 6.18: Optimization Parameter in 3D scenario

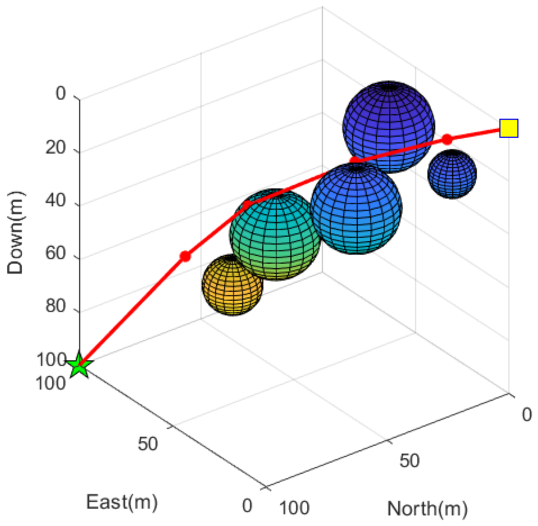


Figure 6.10: Optimal path generated by QPSO-based path planner in 3D scenario

The detailed information about the optimal paths, including the position of waypoints is shown in Table 6.19.

	$x\,[m]$	$y\,[m]$	$z\,[m]$
$\mathbf{P}_1$	13.98	15.29	6.288
$\mathbf{P}_2$	34.08	38.50	18.53
$\mathbf{P}_3$	58.22	64.08	38.00
$\mathbf{P}_4$	74.11	77.08	57.57

Table 6.19: Positions of waypoints generated by QPSO algorithm in 3D scenario

The fitness value of the optimal path, which is also the total path length of a feasible path is shown as follows:

$$\mathcal{F}(\mathcal{P}) = L_{total} = 177.99 \quad [m] \tag{6.12}$$

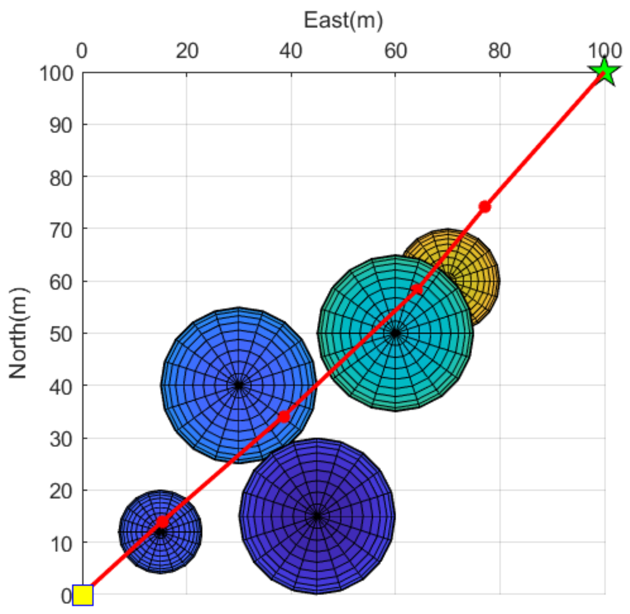


Figure 6.11: Optimal path generated by QPSO-based path planner in the NE-plane

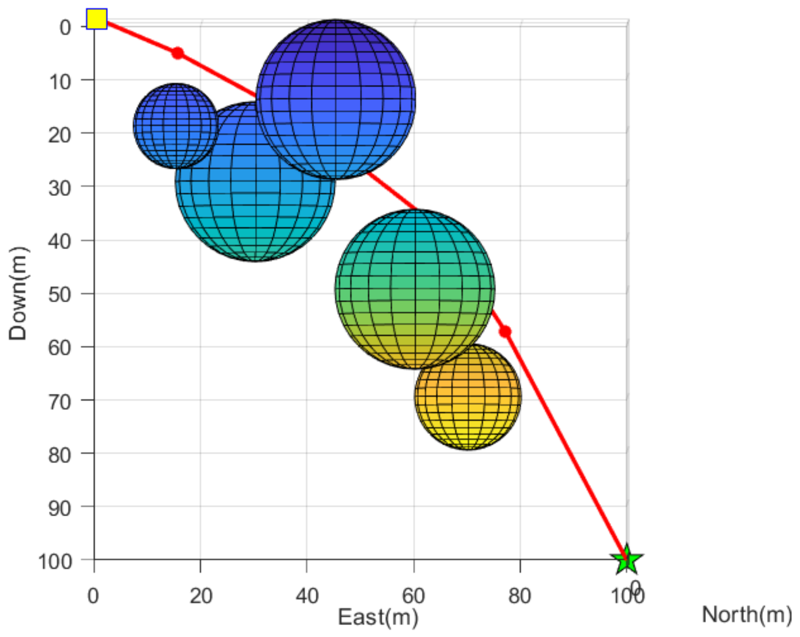


Figure 6.12: Optimal path generated by QPSO-based path planner in the DE-plane

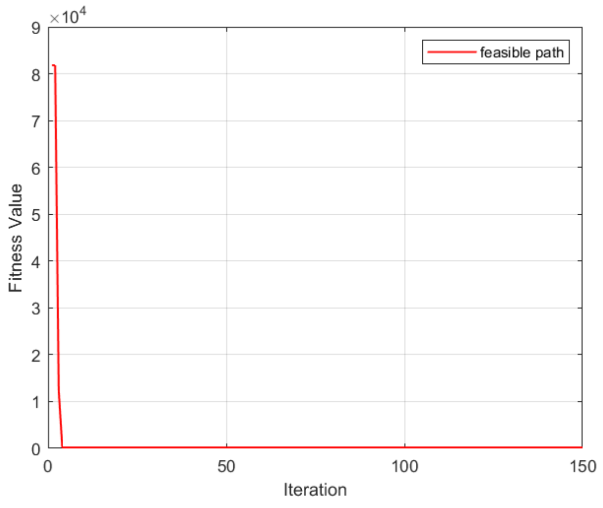


Figure 6.13: The fitness value during iterations by QPSO algorithm in 3D environment

6.3 Path Planning with G-QPSO Algorithm

In this section, no tuning process is done since there is no parameters need to be defined except for β. A 2D simple scenario is considered and then the path planner is tested in a complex environment with both circle-shape and OBB-shape obstacles. Finally, a 3D case is done and the performance in 3D environment is tested.

The G-QPSO-based path planner is based on the G-QPSO algorithm introduced in chapter 5, the Algorithm 3. The parameter β in the algorithm is defined as the same way of inertia weight w, and its mathematical representation is expressed as follows:

$$\beta = w_{max} - \frac{w_{max} - w_{min}}{t_{max}} \times t \tag{6.13}$$

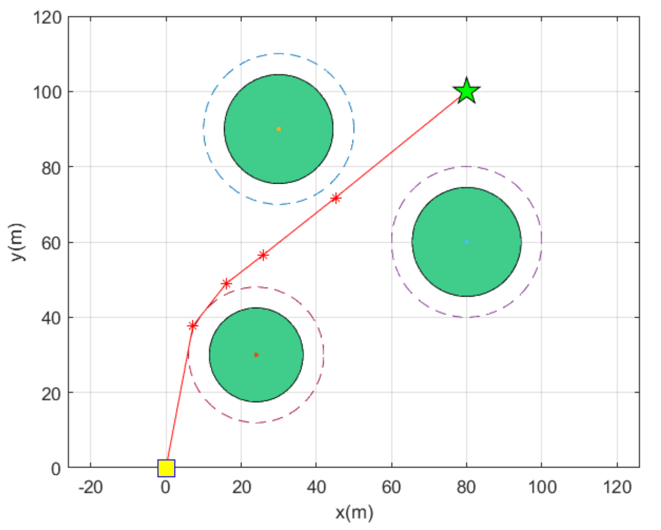
where the parameters are shown in Table 6.15.

w_{max}	w_{min}	t_{max}
0.65	0.20	150

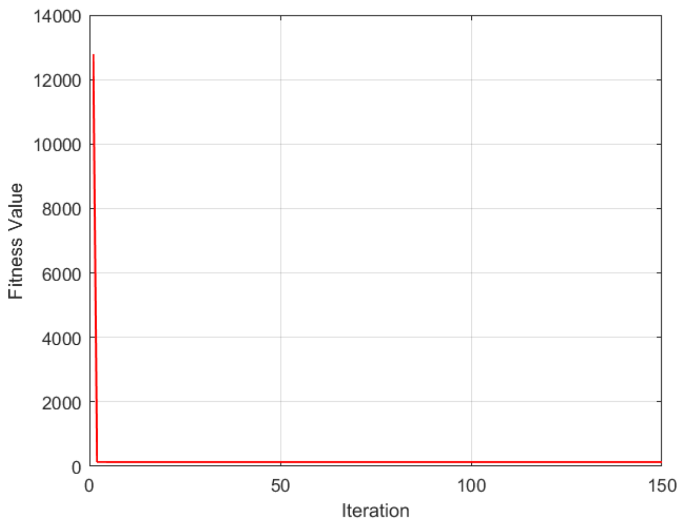
Table 6.20: Optimization Parameter for β in G-QPSO algorithm

6.3.1 2D Path Planning with G-QPSO algorithm

In a simple 2D environment with circle-shape obstacles, whose information is shown in Table. 6.2, the optimal path generated by GQPSO-based path planner with random numbers in Gaussian distribution is shown in Fig. 6.14, as well as the fitness value during the iterations.



(a) Optimal paths generated by G-QPSO algorithm



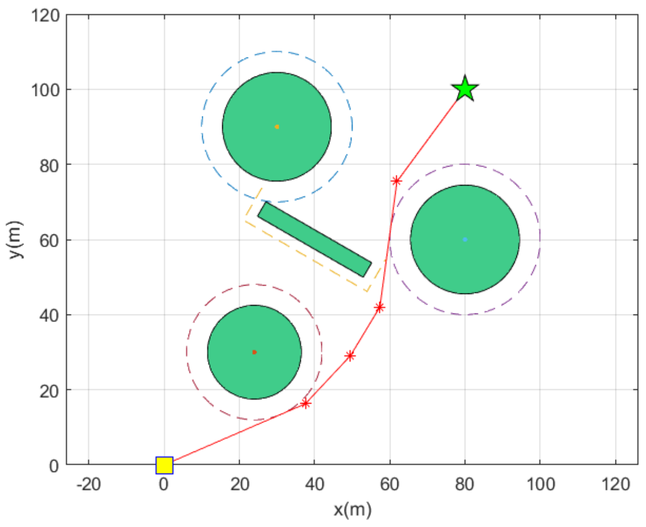
(b) The fitness value during iterations by G-QPSO algorithm

Figure 6.14: Performance of the G-QPSO algorithm with random numbers in Gaussian distribution in simple 2D scenario

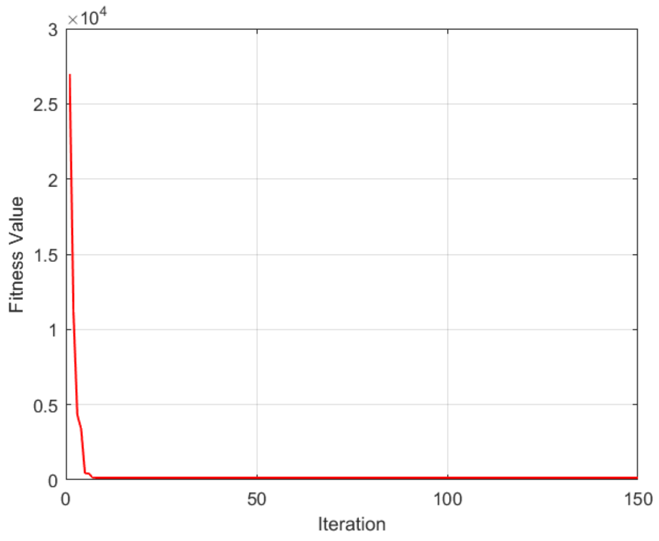
The fitness value of the optimal path, which is also the total path length of a feasible path is shown as follows:

$$\mathcal{F}(\mathcal{P}) = L_{total} = 134.53 \quad [m] \tag{6.14}$$

In a complex 2D environment with OBB-shape obstacles, whose information is shown in Table. 6.3, the optimal path generated by QPSO-based path planner is shown in Fig. 6.15, as well as the fitness value during the iterations.



(a) Optimal paths generated by G-QPSO algorithm



(b) The fitness value during iterations by G-QPSO algorithm

Figure 6.15: PPerformance of the G-QPSO algorithm with random numbers in Gaussian distribution in complex 2D scenario

The fitness value of the optimal path, which is also the total path length of a feasible path is shown as follows:

$$\mathcal{F}(\mathcal{P}) = L_{total} = 137.75 \quad [m] \tag{6.15}$$

6.3.2 3D Path Planning with G-QPSO algorithm

In a simple 3D environment with sphere-shape obstacles, whose information is shown in Table. 6.4, the optimal paths generated by QPSO-based path planner with random numbers in Gaussian distribution are shown in Fig. 6.16, 6.17 and 6.18. The changes of fitness value during the iterations are shown in Fig. 6.19.

G	g	w_{max}	w_{min}	t_{max}
$\lvert randn(0,1)\rvert$	$\lvert randn(0,1)\rvert$	0.65	0.20	150

Table 6.21: Optimization Parameter in 3D scenario

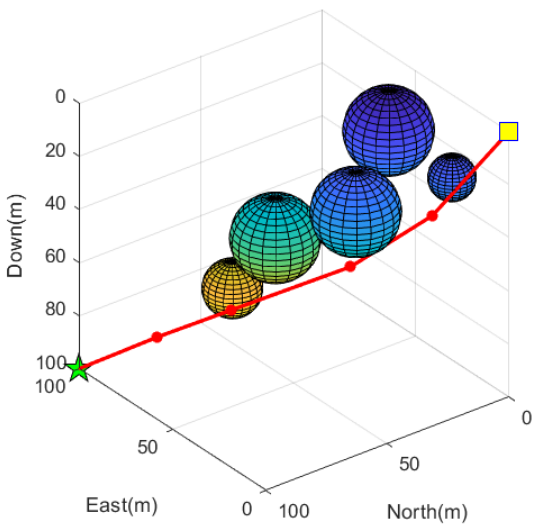


Figure 6.16: Optimal path generated by GQPSO-based path planner in 3D scenario

The detailed information about the optimal paths, including the position of waypoints is shown in Table 6.22.

	$x\,[m]$	$y\,[m]$	$z\,[m]$
$\mathbf{P}_1$	13.98	15.29	6.288
$\mathbf{P}_2$	34.08	38.50	18.53
$\mathbf{P}_3$	58.22	64.08	38.00
$\mathbf{P}_4$	74.11	77.08	57.57

Table 6.22: Positions of waypoints generated by QPSO algorithm in 3D scenario

The fitness value of the optimal path, which is also the total path length of a feasible path is shown as follows:

$$\mathcal{F}(\mathcal{P}) = L_{total} = 176.32 \quad [m] \tag{6.16}$$

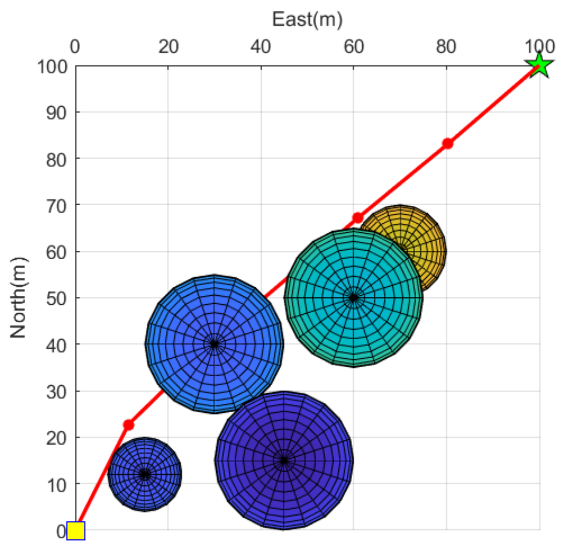


Figure 6.17: Optimal path generated by GQPSO-based path planner in the NE-plane

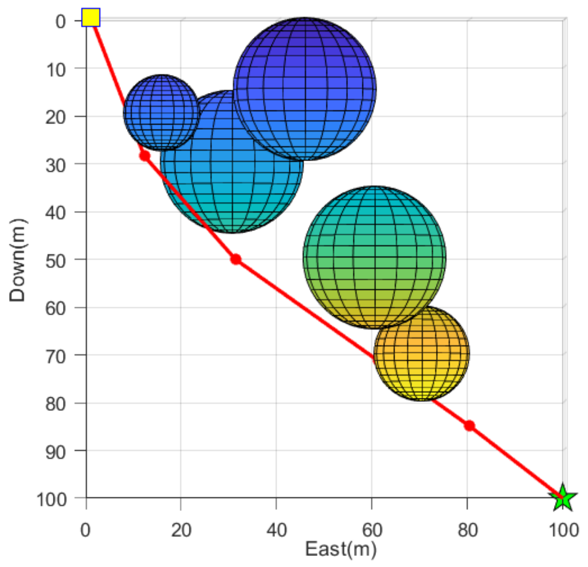


Figure 6.18: Optimal path generated by GQPSO-based path planner in the DE-plane

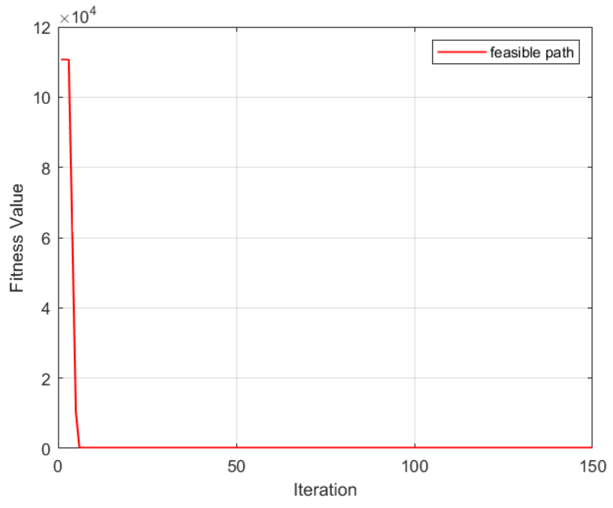


Figure 6.19: The fitness value during iterations by GQPSO algorithm in 3D environment

Chapter 7

Discussion

In this chapter, the discussion on the results shown in Chapter 6 is given. Corresponding to the structure of Chapter 6, the discussion part is also divided into 3 categories which are given as follows:

- Discussion on the performance of the PSO-based path planner based on the results presented in Section 6.1, as well as the tuning advice for multiple parameters. This part is given in Section 7.1.
- Discussion on results generated by the QPSO-based path planner and comparison between them and the results with PSO algorithm. This part is given in Section 7.2.
- Discussion on the performance of the GQPSO-based path planner based on the results presented in Section 6.3 and comparison to other algorithms applied before. This part is given in Section 7.3.

7.1 Path Planning with PSO Algorithm

Firstly, some attention should be paid on the general choices of PSO parameters shown in Table 6.1. This set of parameters is used to solve the optimal path planning problems both in 2-dimensional and 3-dimensional scenario. Moreover, all of these algorithms used in Chapter 6 utilize this set of parameters. Hence, a good choice of them is necessary and could save lots of computational cost.

In this thesis, the population size denoted by N is set as 150, which means there are 150 particles in the swarm. In some way, this parameter has an influence on the performance of the PSO algorithm as well as the PSO-based path planner. If the population of the swarm is pretty large, the probability that some of the particles drop on the optimal position in the initialization process is also large, since the particles are initialized randomly in the search space. Though this initialization process is not used in the path planner given in Chapter 5, large population still increases the diversity of the swarm and the exploration ability is increased at the same time, as well as the convergence ability. However, large population is not always beneficial because it will cause increase of the computational cost. Therefore, there should be a trade-off. Scientists have proved that, in most cases, if the population size is greater than 50, the benefit it gives becomes quite small and the PSO algorithm is not sensitive to its change[]. In this thesis, since the testing scenario is not too complex, the population size could be chosen as a large number in order to ensure the convergence. If it is applied to much more complex scenario, the population size should be redesigned to achieve a balance.

For the number of iterations denoted by K, it is set as 150. This is used to limit the times of iterations and as a part of the stopping criterion. Since the PSO algorithm is easy to get stuck in some local optimum, which corresponds to unfeasible paths, this parameter could help quit the loop and try again. Based on the results given in Chapter 6, the path planner always converges to a candidate solution quickly within 50 iterations. Just to be safe, the number of iterations is set as 150 and it works well in all cases shown in Chapter 6.

The safety threshold ϵ is set as two times of the AUVs' length with the consideration of the uncertainty during the travel and used to expand the obstacles. Since the vehicle is considered as a mass point in the simulation, the expansion process of the obstacles is necessary.

The coefficient for velocity bound denoted by k is an important parameter in PSO-based path planner. As mentioned in Chapter 4, this coefficient defines the velocity bound based on Eqs. 4.5 and limits the step size of the particle's movement. At the beginning, it was set as 0.4 and caused violation of the search space. The result is, the output path was always near the boundary of the search space, which increased the length of the path significantly. After some modification, it is set as 0.2 and the new coefficient works well both in the 2D and 3D scenario.

7.1.1 Tuning of Cognitive and Social Coefficient c_1 and c_2

As mentioned in Chapter 4, the cognitive and social coefficients c_1 and c_2 play an important role in PSO algorithm. These two parameters represent how much contribution the personal best and global best make to the particles' movement. The values of the constants c_1 and c_2 determine the extent to which the particles move towards the best position from its own experience and experience of all particles. In order to show its influence and find the best choice of them for different scenarios, three sets of c_1 and c_2 are tested and the results are shown in Fig. 6.1.

By comparing the total path length shown in Table 6.6, there is not too much difference between them, which means all three pairs of c_1 and c_2 could generate a feasible path with short length. But when looking at the change of fitness value during the iterations, it can be found that the convergence abilities of them are different. For the second pair of c_1 and c_2, it converges to the final path much later than other two pairs. It makes sense since it is a conservative choice by making c_1 equal to c_2. It means the experience from particle itself and the whole swarm makes equal contribution to the particle's movement. However, these two parts of experience are not always good at the same time. Sometimes, the particle's own experience is better than that from others, then the setting with larger c_1 could have better performance and faster convergence. That's why the first pair converges faster than the second one, same for the third pair.

Considering both the convergence speed and the final path length, the first pair of c_1 and c_2 is chosen as the best option and used in the following tests.

$$(c_1, c_2) = (1, 2) \tag{7.1}$$

Also, it can be found that the final path generated by the PSO-based path planner is not always the global optimal one even though it's feasible. The path in black, which is generated by the second pair of c_1 and c_2, is quite different from others and its length is slightly greater than others. This path is called local optimal path. Convergence to local optimum is a main drawback for PSO algorithm. Since the difference of path length is quite small and the local optimal path is also feasible, this PSO-based path planner could be seen as a good path planner in some way.

7.1.2 Choice of Definitions of Inertia Weight w

As mentioned in chapter 4, there are several methods to define the inertia weight w, which are shown in Table 4.1. In this thesis, two strategies are chosen and tested, that is, constant inertia weight and linearly decreasing inertia weight.

By observing the final paths generated by path planner with different inertia weight in Fig. 6.1, it can be found that path planner with the linearly decreasing inertia weight converges to the optimal path much faster than the one with constant inertia weight. It makes sense since the inertia weight is used to limit the contribution of velocity term in previous iteration. At the beginning, because of lack of exploration, the step size generated based on the experience from its own and the whole swarm is not quite useful. As the iteration goes, the particle becomes more and more experienced and the step size generated based on the experience becomes more useful. If the inertia weight is constant, even though this step size becomes better, its contribution to particle's movement does not change. With the linearly decreasing inertia weight, the more iterations goes, the more contribution it gives to the particle. Thus, the path planner converges to the optimal path much faster with linearly decreasing inertia weight.

Considering both the convergence speed and the final path length, the linearly decreasing inertia weight is chosen as the best option.

$$w(t) = w_{max} - \frac{w_{max} - w_{min}}{t_{max}} \times t \tag{7.2}$$

7.1.3 2D and 3D Path Planning with PSO Algorithm

By using the best choice of (c_1, c_2) and inertia weight w, the path planner is tested in more complex scenarios, including a 2D scenario with OBB-shape obstacle and a 3D scenario. The results are shown in Fig. 6.3 and 6.4.

Figure 6.3 contains two paths, the one in black, which is an unfeasible path in collision with obstacles, the other one in red which is a feasible path. Similarly, this figure also shows the main disadvantages of PSO-based path planner, that is, it is easy to get stuck in local optimal solution. In previous test, though the final path is not the optimal solution, it is still a feasible path without collision. However, in this scenario with rectangular shape obstacles, it is stuck in a dangerous path which does not fulfill the requirement. It can be also observed by the fitness value. The fitness value of the unfeasible path is greater than 1000 because the penalty function works when the path collides with obstacles.

In a 3D scenario with sphere-shape obstacles, the performance of the path planner is good and the final path is satisfying and the convergence ability is good enough.

In summary, with well-tuned parameters c_1 and c_2 as well as the linearly decreasing inertia weight w, the PSO-based path planner could generate a feasible path with short total length. However, the main drawback of it may lead to some unfeasible paths in complex scenario. Also, the tuning process of it is difficult and problem-based. In different scenarios, the proper parameters could differ a lot. This also causes some inconvenience. These two disadvantages makes its performance turn down generally.

7.2 Path Planning with QPSO Algorithm

Since QPSO algorithm is a modified version of PSO algorithm with better convergence ability to the global optimal solution, the scenarios simulated by the QPSO-based path planner are the same as those for PSO-based path planner. The results are shown in Figure. 6.8, 6.9 and 6.10 corresponding to the 2D simple scenario, 2D complex scenario with OBB-shape obstacles and 3D scenario.

As mentioned in Chapter 4, even though the updating rule in QPSO algorithm is different from the one in PSO algorithm, the cognitive and social coefficient c_1 and c_2 still represent how much contribution the personal best and global best make to the particles' movement. The values of the constants c_1 and c_2 determine the extent to which the particles move towards the best position from its own experience and experience of all particles. Therefore, the tuning choice of these two parameter is discussed.

7.2.1 Tuning for Cognitive and Social Coefficient c_1 and c_2

In order to compare its performance with PSO-based path planner, same setting of the cognitive coefficients is tested and the results are shown in Fig. 6.8. In addition, a new strategy of choosing c_1 and c_2 is tested, that is, generating random c_1 and c_2 with uniform probability distribution function in the range of (0,1).

By observing the optimal paths generated with different pairs of c_1 and c_2, it can be found that, the convergence ability to global optimal solution of QPSO-based path planner is better than that of PSO-based path planner, since in the same scenario with the same c_1 and c_2, the PSO-based path planner gets stuck in the local optimal solution as the line in black in Fig. 6.8 shows, while the global optimal path is achieved by QPSO-based path planner as the line in black in Fig. 6.8 shows. Also, in the complex 2D scenario with OBB-shape obstacles, the better convergence ability to global optimal solution could be proved. In Fig. 6.9, all paths are feasible.

By comparing the path in blue, generated by random pair of c_1 and c_2, it can be found that its performance is good enough. The difference between its total length and the shortest length is less than 1 meter. It's sufficiently small for a a long path. Meanwhile, its converges speed is also satisfying. These shows that it is a good way to generate random c_1 and c_2 with uniform probability distribution function which does not lead to much loss in performance and convergence ability.

7.2.2 3D Path Planning with QPSO Algorithm

With the new strategy of choosing c_1 and c_2 by using uniform probability distribution function in the range of (0,1), the QPSO-based path planner is tested in a simple 3D scenario. The results are shown in Fig. 6.10 and 6.13. Same conclusion could be drawn from this result, that is, in 3D scenario, generating random c_1 and c_2 with uniform probability distribution function does not lead to too loss in performance and convergence ability. The difference of path length is also less than 1 meter and the path planner converges quickly.

In summary, the QPSO-based path planner has better convergence property to global optimal solution than PSO-based path planner, both in 2D and 3D scenarios. Also, due to the change in updating rule, some problem-defined parameters disappear in QPSO algorithm such as the velocity bound and its coefficients. Though cognitive coefficients c_1 and c_2 still exist, they could be achieved by generating random numbers with uniform probability distribution function. The results have shown that this new strategy is satisfying in terms of convergence ability and algorithm performance. With this strategy, the tuning process becomes much easier.

7.3 Path Planning with G-QPSO Algorithm

In G-QPSO algorithm, instead of uniform probability distribution, Gaussian probability distribution is used to generate random numbers for c_1 and c_2. Compared with uniform probability distribution, Gaussian probability distribution has better effect on the choice of parameters because of its mathematical properties, that is, Gaussian distribution sequences have zero mean and unit variance. Applied to the stochastic coefficients in QPSO algorithm, this property results in a good trade-off between the probability of generating movement with small amplitudes around the current space and that with large amplitudes. To compare its performance, the scenarios simulated by the GQPSO-based path planner are the same as those for QPSO-based path planner. The results are shown in Figure. 6.14, 6.15 and 6.16

corresponding to the 2D simple scenario, 2D complex scenario with OBB-shape obstacles and 3D scenario.

By observing the total length of the paths and convergence speed in all scenarios, it can be found that the optimal paths achieved by GQPSO-based path planner always have less path length and faster convergence speed than the ones generated by QPSO-based path planner. However the difference is quite small, the total length of the path is shorten by less than 1 meter and the convergence speed is always fast enough.

In summary, the path planner based on G-QPSO method have good performance and strong convergence ability in all 3 testing scenarios including 2D scenarios with circle and OBB shape obstacles, and the 3D scenario with sphere-shape obstacles. Compared to QPSO-based path planner, the improvement of it is not quite large. If the scenario is larger and more complex, the performance of GQPSO-based path planner might be much better. With the advantage of less tuning parameters and strong convergence ability, GQPSO-based path planner could be seen as a high-performance path planner for AUV.

Chapter 8

Conclusion

In this chapter, based on the discussions given in Chapter 7, some conclusion are presented about the performance of the path planner designed in this thesis. Also, the process of problem formulation and algorithm design is summarized and discussed here. Corresponding to the limitation given in section 1.4, the future work of this thesis is presented.

8.1 Conclusion

This master's thesis is based on the research hypothesis, that is, the research about path planning method is a core technique for achieving high level of autonomy for Autonomous Underwater Vehicles (AUVs) and Particle Swarm Optimization (PSO) method could be utilized to design a high-performance path planner for AUVs. The contribution of this thesis is a GQPSO-based optimal path planner, which allows the AUV generates paths without human intervention.

The optimal path planning problem for AUV is modelled as a multi-objective optimization problem at the beginning, concerning the path length, the safety margin and smoothness of the path. In order to apply PSO algorithm, the preliminary model is modified to an unconstrained optimization problem with single objective function. To save computational cost, the obstacles are modelled as circles and oriented bounded boxes in 2D scenarios and sphere in 3D scenario. To ensure good cooperation between the path planner and the control system of AUV, a guidance system with constant jerk is chosen and introduced.

The optimal path planner is developed by applying PSO, QPSO and GQPSO method to the optimal path planning problem, and their performances are examined through simulations in 2D scenarios and 3D scenario. After discussion about the test results, some conclusion could be drawn:

1. With well-tuned parameters, the performance of PSO-based optimal path planner is satisfying, while its weak convergence ability to global optimum can not be overcome.

2. For a PSO-based path planner, using linearly decreasing inertia weight could make the algorithm converge faster.
3. Optimal path planner with QPSO/GQPSO method has stronger convergence ability to global optimum than that with PSO method.
4. Using QPSO and GQPSO method could shorten the tuning process significantly and could not lead to too much loss in path length.
5. The GQPSO-based optimal path planner could be seen as a high-performance path planner for LAUV.

8.2 Future Work

During the work of the master's thesis, the limitations of this research come forward and corresponding topics for future work are determined. In the process of problem formulation, the path planning problem is modelled as a multi-objective optimization problem. In order to apply PSO algorithm, this preliminary model is simplified and changed to an unconstrained optimization problem with single objective function. In some way, this modification has a bad influence on the performance of the path planner. In recent years, a modified version of PSO algorithm called MOPSO method which could handle multiple objective functions is proposed by scientists. In the future work, this modified PSO algorithm could be used to design the optimal path planner.

Another limitation of the research is, all of the results shown in this thesis are obtained from simulations in MATLAB and no practical experiments are done. Though the uncertainty of real environment are taken into consideration, it is hard to say if the PSO-based optimal path planner proposed in this thesis could achieve good performance in real environment. In future work, some practical experiments could be done to test its performance in real world.

Moreover, the testing scenarios in this thesis are not designed at a large scale. The performance of the optimal path planner in large-scale scenarios is uncertain and needs to be tested. Therefore, in the future work, some larger scenarios could be simulated to test its performance.

Bibliography

[1] C. Von Alt, 'Autonomous underwater vehicles,' in *Autonomous Underwater Lagrangian Platforms and Sensors Workshop*, vol. 3, 2003, p. 2.

[2] P. Bhopale, F. Kazi and N. Singh, 'Reinforcement learning based obstacle avoidance for autonomous underwater vehicle,' *Journal of Marine Science and Application*, vol. 18, no. 2, pp. 228–238, 2019.

[3] R. Dechter and J. Pearl, 'Generalized best-first search strategies and the optimality of a,' *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 505–536, 1985.

[4] K. P. Carroll, S. R. McClaran, E. L. Nelson, D. M. Barnett, D. K. Friesen and G. N. William, 'Auv path planning: An a* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones,' in *Proceedings of the 1992 symposium on autonomous underwater vehicle technology*, IEEE, 1992, pp. 79–84.

[5] R. J. Szczerba, P. Galkowski, I. S. Glicktein and N. Ternullo, 'Robust algorithm for real-time route planning,' *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 3, pp. 869–878, 2000.

[6] S. Chen, C. Liu, Z. Huang and G. Cai, 'Global path planning for auv based on sparse a* search algorithm,' *Torpedo Technology*, vol. 20, no. 4, pp. 271–275, 2012.

[7] M. Li and H. Zhang, 'Auv 3d path planning based on a* algorithm,' in *2020 Chinese Automation Congress (CAC)*, IEEE, 2020, pp. 11–16.

[8] H. G. Cobb and J. J. Grefenstette, 'Genetic algorithms for tracking changing environments.,' Naval Research Lab Washington DC, Tech. Rep., 1993.

[9] A. Alvarez and A. Caiti, 'A genetic algorithm for autonomous undetwater vehicle route planning in ocean environments with complex space-time variability,' *IFAC Proceedings Volumes*, vol. 34, no. 7, pp. 237–242, 2001.

[10] A. Alvarez, A. Caiti and R. Onken, 'Evolutionary path planning for autonomous underwater vehicles in a variable ocean,' *IEEE Journal of Oceanic Engineering*, vol. 29, no. 2, pp. 418–429, 2004.

[11] Y. Sun and R. Zhang, 'Research on global path planning for auv based on ga,' in *Mechanical Engineering and Technology*, Springer, 2012, pp. 311–318.

[12] J. Cao, Y. Li, S. Zhao and X. Bi, 'Genetic-algorithm-based global path planning for auv,' in *2016 9th International Symposium on Computational Intelligence and Design (ISCID)*, IEEE, vol. 2, 2016, pp. 79–82.

[13] C. Cheng, Q. Sha, B. He and G. Li, 'Path planning and obstacle avoidance for auv: A review,' *Ocean Engineering*, vol. 235, p. 109 355, 2021.

[14] C.-B. Zhang, Y.-J. Gong, J.-J. Li and Y. Lin, 'Automatic path planning for autonomous underwater vehicles based on an adaptive differential evolution,' in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 2014, pp. 89–96.

[15] X. Li, D. Zhu and Y. Qian, 'A survey on formation control algorithms for multi-auv system,' *Unmanned Systems*, vol. 2, no. 04, pp. 351–359, 2014.

[16] R. Eberhart and J. Kennedy, 'A new optimizer using particle swarm theory,' in *MHS'95. Proceedings of the sixth international symposium on micro machine and human science*, Ieee, 1995, pp. 39–43.

[17] G. Yang and R. Zhang, 'Path planning of auv in turbulent ocean environments used adapted inertia-weight pso,' in *2009 Fifth International Conference on Natural Computation*, IEEE, vol. 3, 2009, pp. 299–302.

[18] H. S. Lim, S. Fan, C. K. Chin, S. Chai and N. Bose, 'Particle swarm optimization algorithms with selective differential evolution for auv path planning,' *International Journal of Robotics and Automation*, vol. 9, no. 2, pp. 94–112, 2020.

[19] A. Colorni, M. Dorigo, V. Maniezzo *et al.*, 'Distributed optimization by ant colonies,' in *Proceedings of the first European conference on artificial life*, Paris, France, vol. 142, 1991, pp. 134–142.

[20] H.-D. Wang, G. D. Stanwood, D. K. Grandy and A. Y. Deutch, 'Dystrophic dendrites in prefrontal cortical pyramidal cells of dopamine d1 and d2 but not d4 receptor knockout mice,' *Brain research*, vol. 1300, pp. 58–64, 2009.

[21] F. Ma, X. Wang, X. Liu, J. Yu, T. Wang and W. Zhang, 'Application of segmentation threshold method and wavelet threshold denoising based on emd in Φ-otdr system,' in *Tenth International Conference on Information Optics and Photonics*, International Society for Optics and Photonics, vol. 10964, 2018, p. 1 096 435.

[22] C. Hu and F. Zhang, 'Research on auv global path planning based on multi-objective ant colony strategy,' in *2019 Chinese Automation Congress (CAC)*, IEEE, 2019, pp. 5512–5517.

[23] C. S. Tan, R. Sutton and J. Chudley, 'An incremental stochastic motion planning technique for autonomous underwater vehicles,' *IFAC Proceedings Volumes*, vol. 37, no. 10, pp. 483–488, 2004.

[24] E. Hernández, M. Carreras, J. Antich, P. Ridao and A. Ortiz, 'A topologically guided path planner for an auv using homotopy classes,' in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 2337–2343.

[25] J. D. Hernández, G. Vallicrosa, E. Vidal, È. Pairet, M. Carreras and P. Ridao, 'On-line 3d path planning for close-proximity surveying with auvs,' *IFAC-PapersOnLine*, vol. 48, no. 2, pp. 50–55, 2015.

[26] Y. Li, Y. Wang and J. Sheng, 'The evolution of cooperation on geographical networks,' *Physica A: Statistical Mechanics and its Applications*, vol. 485, pp. 1–10, 2017.

[27] L. Yu, Z. Wei, Z. Wang, Y. Hu and H. Wang, 'Path optimization of auv based on smooth-rrt algorithm,' in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, IEEE, 2017, pp. 1498–1502.

[28] E. Taheri, M. H. Ferdowsi and M. Danesh, 'Closed-loop randomized kinodynamic path planning for an autonomous underwater vehicle,' *Applied Ocean Research*, vol. 83, pp. 48–64, 2019.

[29] O. Khatib, 'Real-time obstacle avoidance for manipulators and mobile robots,' in *Autonomous robot vehicles*, Springer, 1986, pp. 396–404.

[30] D. Fu-guang, J. Peng, B. Xin-qian and W. Hong-Jian, 'Auv local path planning based on virtual potential field,' in *IEEE International Conference Mechatronics and Automation, 2005*, IEEE, vol. 4, 2005, pp. 1711–1716.

[31] C. Cheng, D. Zhu, B. Sun, Z. Chu, J. Nie and S. Zhang, 'Path planning for autonomous underwater vehicle based on artificial potential field and velocity synthesis,' in *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, IEEE, 2015, pp. 717–721.

[32] L. Ge, H. Li, Q. Wang, G. Wei, Z. Hu, J. Liao and J. Li, 'Design and optimization of annular flow electromagnetic measurement system for drilling engineering,' *Journal of Sensors*, vol. 2018, 2018.

[33] O. Cordón, F. Herrera, E. Herrera-Viedma and M. Lozano, 'Genetic algorithms and fuzzy logic in control processes,' *Archives Of Control Science*, vol. 5, pp. 135–168, 1996.

[34] S. Khanmohammadi and O. Ghaderi, 'Simultaneous coordinated tuning of fuzzy pss and fuzzy facts device stabilizer for damping power system oscillations in multi-machine power system,' in *2007 IEEE International Fuzzy Systems Conference*, IEEE, 2007, pp. 1–6.

[35] S. X. Yang and M.-H. Meng, 'Real-time collision-free motion planning of a mobile robot using a neural dynamics-based approach,' *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1541–1552, 2003.

[36] M. Z. Yan and D. Q. Zhu, 'An algorithm of complete coverage path planning for autonomous underwater vehicles,' in *Key Engineering Materials*, Trans Tech Publ, vol. 467, 2011, pp. 1377–1385.

[37] D. Zhu, W. Li, M. Yan and S. X. Yang, 'The path planning of auv based on ds information fusion map building and bio-inspired neural network in unknown dynamic environment,' *International Journal of Advanced Robotic Systems*, vol. 11, no. 3, p. 34, 2014.

[38] J. Ni, L. Wu, P. Shi and S. X. Yang, 'A dynamic bioinspired neural network based real-time path planning method for autonomous underwater vehicles,' *Computational intelligence and neuroscience*, vol. 2017, 2017.

[39] G. Ding, D. Zhu and B. Sun, 'Formation control and obstacle avoidance of multi-auv for 3-d underwater environment,' in *Proceedings of the 33rd Chinese Control Conference*, IEEE, 2014, pp. 8347–8352.

[40] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[41] H. Kawano and T. Ura, 'Fast reinforcement learning algorithm for motion planning of nonholonomic autonomous underwater vehicle in disturbance,' in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, vol. 1, 2002, pp. 903–908.

[42] R. Gore, K. Pattanaik and S. Bharti, 'Efficient re-planned path for autonomous underwater vehicle in random obstacle scenario,' in *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, IEEE, 2019, pp. 1–5.

[43] R. Hassan, B. Cohanim, O. De Weck and G. Venter, 'A comparison of particle swarm optimization and the genetic algorithm,' in *46th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference*, 2005, p. 1897.

[44] J. Sun, B. Feng and W. Xu, 'Particle swarm optimization with particles having quantum behavior,' in *Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753)*, IEEE, vol. 1, 2004, pp. 325–331.

[45] J. Sun, X. Wu, V. Palade, W. Fang, C.-H. Lai and W. Xu, 'Convergence analysis and improvements of quantum-behaved particle swarm optimization,' *Information Sciences*, vol. 193, pp. 81–103, 2012.

[46] L. Coelho, 'Novel gaussian quantum-behaved particle swarm optimiser applied to electromagnetic design,' *IET science, measurement & technology*, vol. 1, no. 5, pp. 290–294, 2007.

[47] M. Xi, J. Sun and W. Xu, 'An improved quantum-behaved particle swarm optimization algorithm with weighted mean best position,' *Applied Mathematics and Computation*, vol. 205, no. 2, pp. 751–759, 2008.

[48] J. Guo, J. Wang and G. Cui, 'Online path planning for uav navigation based on quantum particle swarm optimization,' in *Advanced Technology in Teaching-Proceedings of the 2009 3rd International Conference on Teaching and Computational Science (WTCS 2009)*, Springer, 2012, pp. 291–302.

[49] F. Dukan, 'Rov motion control systems,' 2014.

[50] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2013.

[51] B. Patle, A. Pandey, D. Parhi, A. Jagadeesh *et al.*, 'A review: On path planning strategies for navigation of mobile robot,' *Defence Technology*, vol. 15, no. 4, pp. 582–606, 2019.

[52] F. van den Bergh *et al.*, 'An analysis of particle swarm optimizers [ph. d. thesis],' *Pretoria: Natural and Agricultural Science Department, University of Pretoria*, vol. 95, 2001.

[53] R. C. Eberhart, Y. Shi and J. Kennedy, *Swarm intelligence*. Elsevier, 2001.

[54] F. Marini and B. Walczak, 'Particle swarm optimization (pso). a tutorial,' *Chemometrics and Intelligent Laboratory Systems*, vol. 149, pp. 153–165, 2015.

[55] J. Kennedy and R. Eberhart, 'Particle swarm optimization,' in *Proceedings of ICNN'95-international conference on neural networks*, IEEE, vol. 4, 1995, pp. 1942–1948.

[56] W. Schweizer, *Numerical quantum dynamics*. Springer Science & Business Media, 2001, vol. 9.

[57] Y. Cai, J. Sun, J. Wang, Y. Ding, N. Tian, X. Liao and W. Xu, 'Optimizing the codon usage of synthetic gene with qpso algorithm,' *Journal of Theoretical Biology*, vol. 254, no. 1, pp. 123–127, 2008.

[58] M. Clerc and J. Kennedy, 'The particle swarm-explosion, stability, and convergence in a multidimensional complex space,' *IEEE transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[59] L. dos Santos Coelho, 'Gaussian quantum-behaved particle swarm optimization approaches for constrained engineering design problems,' *Expert Systems with Applications*, vol. 37, no. 2, pp. 1676–1683, 2010.

[60] C. A. C. Coello, 'Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art,' *Computer methods in applied mechanics and engineering*, vol. 191, no. 11-12, pp. 1245–1287, 2002.

[61] Z. Michalewicz and M. Schoenauer, 'Evolutionary algorithms for constrained parameter optimization problems,' *Evolutionary computation*, vol. 4, no. 1, pp. 1–32, 1996.

[62] P. B. Fernandes, R. C. L. De Oliveira and J. V. F. Neto, 'A modified qpso for robotic vehicle path planning,' in *2018 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2018, pp. 1–7.

Appendix A

Matlab Code

In this appendix, the Matlab-code of the proposed optimal path planner is presented, including PSO-based and GQPSO-based algorithms. Since the code of QPSO-based algorithm is quite similar to that of GQPSO-based algorithm. Therefore, the code of QPSO algorithm is not presented and the difference is marked in the code. The parameter setting is also presented here.

A.1 Code of GQPSO-based optimal path planner

A.1.1 Main.m

```

clc;
clear;
close all;

%% Problem Definition

model=CreateModel();

model.n=4;  % number of Handle Points

CostFunction=@(x) MyCost(x,model);    % Cost Function

nVar=model.n;       % Number of Decision Variables

VarSize=[1 nVar];   % Size of Decision Variables Matrix

VarMin.x=model.xmin;           % Lower Bound of Variables
VarMax.x=model.xmax;           % Upper Bound of Variables
VarMin.y=model.ymin;           % Lower Bound of Variables
VarMax.y=model.ymax;           % Upper Bound of Variables


%% PSO Parameters

MaxIt=150;          % Maximum Number of Iterations

nPop=150;           % Population Size (Swarm Size)
```

```

w1 = 0.2;           % Minimum intertia weight

w2 = 0.65;          % Maximum inertia weight


%% Initialization

% Create Empty Particle Structure
empty_particle.Position=[];
empty_particle.Velocity=[];
empty_particle.Cost=[];
empty_particle.Sol=[];
empty_particle.Best.Position=[];
empty_particle.Best.Cost=[];
empty_particle.Best.Sol=[];

% Initialize Global Best
GlobalBest.Cost=inf;

% Create Particles Matrix
particle=repmat(empty_particle,nPop,1);

% Initialization Loop
for i=1:nPop

    % Initialize Position
    if i > 1
        particle(i).Position=CreateRandomSolution(model);
    else
        % Straight line from source to destination

        xx = linspace(model.xs, model.xt, model.n+2);
        yy = linspace(model.ys, model.yt, model.n+2);

        particle(i).Position.x = xx(2:end-1);
        particle(i).Position.y = yy(2:end-1);
    end

    % Evaluation of the fitness value
    [particle(i).Cost, particle(i).Sol]=CostFunction(particle(i).Position);

    % Update Personal Best
    particle(i).Best.Position=particle(i).Position;
    particle(i).Best.Cost=particle(i).Cost;
    particle(i).Best.Sol=particle(i).Sol;

    % Update Global Best
    if particle(i).Best.Cost<GlobalBest.Cost

        GlobalBest=particle(i).Best;

    end

end

% Array to Hold Best Cost Values at Each Iteration
BestCost=zeros(MaxIt,1);

%% Main Loop of GQPSO-based path planner
```

```

for it=1:MaxIt

    beta = w1 + (w2 - w1)*(MaxIt - it)/MaxIt;           % generate beta
    mbest_x = 0;
    mbest_y = 0;

    for j = 1:nPop
        mbest_x = mbest_x + particle(j).Best.Position.x;
        mbest_y = mbest_y + particle(j).Best.Position.y;
    end

    mbest_x = mbest_x/nPop;                             % Calculate Mbest
    mbest_y = mbest_y/nPop;

    for i=1:nPop

        % x part
        % Generate random numbers with Gaussian distribution fuinction
        phi_x = abs(randn);         % phi_x = rand for QPSO
        G_x = abs(randn);           % G_x   = rand for QPSO
        La_x = (G_x * particle(i).Best.Position.x + phi_x *...
        GlobalBest.Position.x)/(G_x + phi_x);
        A_x = beta * abs(mbest_x - particle(i).Position.x) * log(1/G_x);
        if rand > 0.5
            particle(i).Position.x = La_x + A_x;
        else
            particle(i).Position.x = La_x - A_x;
        end

        % Check if the particle violates the bound of search space
        particle(i).Position.x = max(particle(i).Position.x,VarMin.x);
        particle(i).Position.x = min(particle(i).Position.x,VarMax.x);

        % y part

        phi_y = abs(randn);         %  phi_y = rand for QPSO
        G_y = abs(randn);           %  G_y   = rand for QPSO
        La_y = (G_y * particle(i).Best.Position.y + phi_y  *...
        GlobalBest.Position.y)/(G_y+phi_y);
        A_y = beta * abs(mbest_y - particle(i).Position.y) * log(1/G_y);
        if rand > 0.5
            particle(i).Position.y = La_y + A_y;
        else
            particle(i).Position.y = La_y - A_y;
        end

        % Check if the particle violates the bound of search space
        particle(i).Position.y = max(particle(i).Position.y,VarMin.y);
        particle(i).Position.y = min(particle(i).Position.y,VarMax.y);

        % Evaluation
        [particle(i).Cost, particle(i).Sol]=CostFunction(particle(i).Position);

        % Update Personal Best
        if particle(i).Cost<particle(i).Best.Cost

            particle(i).Best.Position=particle(i).Position;
            particle(i).Best.Cost=particle(i).Cost;
            particle(i).Best.Sol=particle(i).Sol;

            % Update Global Best
            if particle(i).Best.Cost<GlobalBest.Cost
                GlobalBest=particle(i).Best;
            end

        end


    end

    % Update Best Cost Ever Found
    BestCost(it)=GlobalBest.Cost;


    % Show Iteration Information
    if GlobalBest.Sol.IsFeasible
        Flag=' *';
    else
        Flag=[', Violation = ' num2str(GlobalBest.Sol.Violation)];
    end
    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it)) Flag]);

    % Plot Solution
    figure(1);
    PlotSolution(GlobalBest.Sol,model);
    pause(0.01);

end

%% Results

figure;
plot(BestCost,'LineWidth',2);
xlabel('Iteration');
ylabel('Best Cost');
grid on;
```

A.1.2 CostFunction.m

```
function sol2=CostFunction(sol1,model)

    x=sol1.x;
    y=sol1.y;

    xs=model.xs;
    ys=model.ys;
    xt=model.xt;
    yt=model.yt;

    xobs=model.xobs;
    yobs=model.yobs;
    robs=model.robs;
    recobs_x = model.recobs_x;
    recobs_y = model.recobs_y;
    a = model.rec_a;
    b = model.rec_b;
    rec_ori_x = model.rec_ori_x;
    rec_ori_y = model.rec_ori_y;

    XS=[xs x xt];
    YS=[ys y yt];

    k=numel(XS);
    k1 = k-2;
    TS=linspace(0,1,k);

    % linear interpolation along the path segement
    tt=linspace(0,1,100);
    xx=interp1(TS,XS,tt);
    yy=interp1(TS,YS,tt);

    dx=diff(xx);
    dy=diff(yy);

    % Calculate the length of the path
    L=sum(sqrt(dx.^2+dy.^2));

    nobs = numel(xobs); % Number of Obstacles
    Violation = 0;

    % Check if the path collides with circle-shape obstacles

    for k=1:nobs
        d=sqrt((xx-xobs(k)).^2+(yy-yobs(k)).^2);
        v=max(1-d/robs(k),0);
        Violation=Violation+mean(v);
    end

    vec2 = [recobs_x(2)-recobs_x(1),recobs_y(2)-recobs_y(1)];
    vec3 = [recobs_x(4)-recobs_x(1),recobs_y(4)-recobs_y(1)];
    vec5 = [recobs_x(2)-recobs_x(3),recobs_y(2)-recobs_y(3)];
    vec6 = [recobs_x(4)-recobs_x(3),recobs_y(4)-recobs_y(3)];

    Violation_rec = 0;
    v_rec = 0;
    num_bad = 0;
    dist_max = 0.5*sqrt(a^2 + b^2);

    % Check if the path collides with OBB-shape obstacles

    for k = 1:numel(xx)
        v_rec = 0;
        vec1 = [xx(k)-recobs_x(1),yy(k)-recobs_y(1)];
        vec4 = [xx(k)-recobs_x(3),yy(k)-recobs_y(3)];
        if dot(vec1,vec2)<0
            v_rec = v_rec + 1;
        end
        if dot(vec1,vec3)<0
            v_rec = v_rec + 1;
        end
        if dot(vec4,vec5)<0
            v_rec = v_rec + 1;
        end
        if dot(vec4,vec6)<0
            v_rec = v_rec + 1;
        end

        if v_rec > 0
            distance = sqrt((xx(k) - rec_ori_x)^2 +...
                       (yy(k) - rec_ori_y)^2);
            Violation_rec = Violation_rec + max(1 - distance/dist_max,0);
            num_bad = num_bad + 1;

        end
    end

    Violation_rec = Violation_rec/num_bad;

    % Check the turning angle
    penl = 0;
    for k = 1:k1
        vector1 = [XS(k+1)-XS(k),YS(k+1)-YS(k)];
        vector2 = [XS(k+2)-XS(k+1),YS(k+2)-YS(k+1)];
        turn = abs(acos(dot(vector1,vector2)/(norm(vector1)*norm(vector2))));
        penl = penl + max((1-(pi/6)/turn),0);
    end

    sol2.TS=TS;
    sol2.XS=XS;
    sol2.YS=YS;
    sol2.tt=tt;
    sol2.xx=xx;
    sol2.yy=yy;
    sol2.dx=dx;
    sol2.dy=dy;
    sol2.L=L;
    sol2.Violation=Violation + penl + Violation_rec;
    sol2.IsFeasible=(Violation + penl + Violation_rec == 0);

end
```

A.2 Code of PSO-based optimal path planner

A.2.1 Main.m

```
clc;
clear;
close all;

%% Problem Definition

model=CreateModel();

model.n=4;  % number of Handle Points

CostFunction=@(x) MyCost(x,model);    % Cost Function

nVar=model.n;       % Number of Decision Variables

VarSize=[1 nVar];   % Size of Decision Variables Matrix

VarMin.x=model.xmin;           % Lower Bound of Variables
VarMax.x=model.xmax;           % Upper Bound of Variables
VarMin.y=model.ymin;           % Lower Bound of Variables
```

```
VarMax.y=model.ymax;           % Upper Bound of Variables


%% PSO Parameters

MaxIt=150;          % Maximum Number of Iterations

nPop=300;           % Population Size (Swarm Size)

w=1;                % Constant Inertia Weight

w1 = 0.2;           % Minimum Inertia Weight
w2 = 0.65;          % Maximum Inertia Weight

c1=1;               % Personal Learning Coefficient
c2=2;               % Global Learning Coefficient

alpha=0.1;
VelMax.x=alpha*(VarMax.x-VarMin.x);    % Maximum Velocity
VelMin.x=-VelMax.x;                    % Minimum Velocity
VelMax.y=alpha*(VarMax.y-VarMin.y);    % Maximum Velocity
VelMin.y=-VelMax.y;                    % Minimum Velocity

%% Initialization

% Create Empty Particle Structure
empty_particle.Position=[];
empty_particle.Velocity=[];
empty_particle.Cost=[];
empty_particle.Sol=[];
empty_particle.Best.Position=[];
empty_particle.Best.Cost=[];
empty_particle.Best.Sol=[];

% Initialize Global Best
GlobalBest.Cost=inf;

% Create Particles Matrix
particle=repmat(empty_particle,nPop,1);

% Initialization Loop
for i=1:nPop

    % Initialize Position
    if i > 1
        particle(i).Position=CreateRandomSolution(model);
    else
        % Straight line from source to destinatio

        xx = linspace(model.xs, model.xt, model.n+2);
        yy = linspace(model.ys, model.yt, model.n+2);

        particle(i).Position.x = xx(2:end-1);
        particle(i).Position.y = yy(2:end-1);
    end

    % Initialize Velocity
    particle(i).Velocity.x=zeros(VarSize);
    particle(i).Velocity.y=zeros(VarSize);

    % Evaluation
    [particle(i).Cost, particle(i).Sol]=CostFunction(particle(i).Position);

    % Update Personal Best
    particle(i).Best.Position=particle(i).Position;
    particle(i).Best.Cost=particle(i).Cost;
    particle(i).Best.Sol=particle(i).Sol;

    % Update Global Best
    if particle(i).Best.Cost<GlobalBest.Cost

        GlobalBest=particle(i).Best;

    end

end

% Array to Hold Best Cost Values at Each Iteration
BestCost=zeros(MaxIt,1);

%% PSO Main Loop

for it=1:MaxIt

    for i=1:nPop
        % Calculate Linearly Decreasing Inertia Weight
        w = w1 + (w2 - w1)*(MaxIt - it)/MaxIt;

        % x Part

        % Update Velocity
        particle(i).Velocity.x = w*particle(i).Velocity.x ...
        + c1*rand(VarSize).*(particle(i).Best.Position.x-particle(i).Position.x)
        + c2*rand(VarSize).*(GlobalBest.Position.x-particle(i).Position.x);

        % Check Velocity Bounds
        particle(i).Velocity.x = max(particle(i).Velocity.x,VelMin.x);
        particle(i).Velocity.x = min(particle(i).Velocity.x,VelMax.x);

        % Update Position
        particle(i).Position.x = particle(i).Position.x + particle(i).Velocity.x;


        % Check Position Bounds
        particle(i).Position.x = max(particle(i).Position.x,VarMin.x);
        particle(i).Position.x = min(particle(i).Position.x,VarMax.x);

        % y Part

        % Update Velocity
        particle(i).Velocity.y = w*particle(i).Velocity.y ...
        + c1*rand(VarSize).*(particle(i).Best.Position.y-particle(i).Position.y)
        + c2*rand(VarSize).*(GlobalBest.Position.y-particle(i).Position.y);

        % Check Velocity Bounds
        particle(i).Velocity.y = max(particle(i).Velocity.y,VelMin.y);
        particle(i).Velocity.y = min(particle(i).Velocity.y,VelMax.y);

        % Update Position
        particle(i).Position.y = particle(i).Position.y + particle(i).Velocity.y;


        % Check Position Bounds
        particle(i).Position.y = max(particle(i).Position.y,VarMin.y);
        particle(i).Position.y = min(particle(i).Position.y,VarMax.y);

        % Evaluate the fitness value
        [particle(i).Cost, particle(i).Sol]=CostFunction(particle(i).Position);

        % Update Personal Best
        if particle(i).Cost<particle(i).Best.Cost

            particle(i).Best.Position=particle(i).Position;
            particle(i).Best.Cost=particle(i).Cost;
            particle(i).Best.Sol=particle(i).Sol;

            % Update Global Best
            if particle(i).Best.Cost<GlobalBest.Cost
                GlobalBest=particle(i).Best;
            end

        end


    end

    % Update Best Cost Ever Found
    BestCost(it)=GlobalBest.Cost;

    % Show Iteration Information
    if GlobalBest.Sol.IsFeasible
        Flag=' *';
    else
        Flag=[', Violation = ' num2str(GlobalBest.Sol.Violation)];
    end
    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it)) Flag]);

    figure(1);
    PlotSolution(GlobalBest.Sol,model);
    pause(0.01);

end

%% Results

figure;
plot(BestCost,'LineWidth',2);
xlabel('Iteration');
ylabel('Best Cost');
grid on;
```

A.3 Code of obstacles setting

A.3.1 2D scenario with OBB-shape obstacle

```
function model=CreateModel()

```

```
% Starting point
xs=0;
ys=0;

% Destination point
xt=2*40;
yt=2*50;

% Obstacles setting

% Circle-shape obstacle

xobs=2*[15 40 12];
yobs=2*[45 30 15];
robs=2*[10 10 9];
n=3;

% OBB-shape obstacle

Rec_ori_x = 2*[20];
Rec_ori_y = 2*[30];
b = 2*19;
a = 2*5;
Rec_angle = [-pi/6];

Rec_obs_x = zeros(1,4);
Rec_obs_y = zeros(1,4);

% Calculate the position of vertices

Rec_obs_x(1) = Rec_ori_x(1) -  0.5*b*cos(Rec_angle(1)) -...
0.5*a*sin(Rec_angle(1));
Rec_obs_x(2) = Rec_ori_x(1) -  0.5*b*cos(Rec_angle(1)) +...
0.5*a*sin(Rec_angle(1));
Rec_obs_x(3) = Rec_ori_x(1) +  0.5*b*cos(Rec_angle(1)) +...
0.5*a*sin(Rec_angle(1));
Rec_obs_x(4) = Rec_ori_x(1) +  0.5*b*cos(Rec_angle(1)) -...
0.5*a*sin(Rec_angle(1));

Rec_obs_y(1) = Rec_ori_y(1) +  0.5*a*cos(Rec_angle(1)) -...
0.5*b*sin(Rec_angle(1));
Rec_obs_y(2) = Rec_ori_y(1) -  0.5*a*cos(Rec_angle(1)) -...
0.5*b*sin(Rec_angle(1));
Rec_obs_y(3) = Rec_ori_y(1) -  0.5*a*cos(Rec_angle(1)) +...
0.5*b*sin(Rec_angle(1));
Rec_obs_y(4) = Rec_ori_y(1) +  0.5*a*cos(Rec_angle(1)) +...
0.5*b*sin(Rec_angle(1));

Rec_obs_x1 = zeros(1,4);
Rec_obs_y1 = zeros(1,4);

b1 = b - 5.52;
a1 = a -5.52;

Rec_obs_x1(1) = Rec_ori_x(1) -  0.5*b1*cos(Rec_angle(1)) -...
0.5*a1*sin(Rec_angle(1));
Rec_obs_x1(2) = Rec_ori_x(1) -  0.5*b1*cos(Rec_angle(1)) +...
0.5*a1*sin(Rec_angle(1));
Rec_obs_x1(3) = Rec_ori_x(1) +  0.5*b1*cos(Rec_angle(1)) +...
0.5*a1*sin(Rec_angle(1));
```

```
    Rec_obs_x1(4) = Rec_ori_x(1) +  0.5*b1*cos(Rec_angle(1)) -...
    0.5*a1*sin(Rec_angle(1));

    Rec_obs_y1(1) = Rec_ori_y(1) +  0.5*a1*cos(Rec_angle(1)) -...
    0.5*b1*sin(Rec_angle(1));
    Rec_obs_y1(2) = Rec_ori_y(1) -  0.5*a1*cos(Rec_angle(1)) -...
    0.5*b1*sin(Rec_angle(1));
    Rec_obs_y1(3) = Rec_ori_y(1) -  0.5*a1*cos(Rec_angle(1)) +...
    0.5*b1*sin(Rec_angle(1));
    Rec_obs_y1(4) = Rec_ori_y(1) +  0.5*a1*cos(Rec_angle(1)) +...
    0.5*b1*sin(Rec_angle(1));

    % Define search space

    xmin=0;
    xmax= 120;

    ymin=0;
    ymax= 120;

    model.xs=xs;
    model.ys=ys;
    model.xt=xt;
    model.yt=yt;
    model.xobs=xobs;
    model.yobs=yobs;
    model.robs=robs;
    model.recobs_x = Rec_obs_x;
    model.recobs_y = Rec_obs_y;
    model.recobs_x1 = Rec_obs_x1;
    model.recobs_y1 = Rec_obs_y1;
    model.n=n;
    model.xmin=xmin;
    model.xmax=xmax;
    model.ymin=ymin;
    model.ymax=ymax;
    model.rec_a = a;
    model.rec_b = b;
    model.rec_ori_x = Rec_ori_x;
    model.rec_ori_y = Rec_ori_y;

end
```

A.3.2 3D scenario with sphere-shape obstacles

```
function model=CreateModel()

    % Starting point
    xs=0;
    ys=0;
    zs=0;

    % Destination point
    xt=100;
    yt=100;
    zt=100;

    % Position of the center

    xobs=[15 40 12 60 50 ];
    yobs=[45 30 15 70 60 ];
    zobs=[15 30 20 70 50 ];

    % Radius

    robs=[15 15 8 10 15 ];

    n=5;

    % Define the search space

    xmin=-200;
    xmax= 200;

    ymin=-200;
    ymax= 200;

    zmin=-100;
    zmax= 100;


    model.xs=xs;
    model.ys=ys;
    model.zs=zs;

    model.xt=xt;
    model.yt=yt;
    model.zt=zt;

    model.xobs=xobs;
    model.yobs=yobs;
    model.zobs=zobs;
    model.robs=robs;
    model.n=n;

    model.xmin=xmin;
    model.xmax=xmax;
    model.ymin=ymin;
    model.ymax=ymax;
    model.zmin=zmin;
    model.zmax=zmax;

end
```

Hantong Liu