MOBILE ROBOT PATH PLANNING USING GENETIC ALGORITHM GLOBAL PATH PLANNING AND POTENTIAL FIELD PATH ADJUSTING

by

AbdelRahman Mahmoud Eliwa

Submitted in partial fulfilment of the
requirements for the degree of
Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
April 2017

To my Mother Ola and my Father Mahmoud

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The purpose of this thesis was to develop an algorithm which solves the path planning problem for a two-wheeled mobile robot. The algorithm included multiple robots within it. The algorithm was developed by first identifying the problem at hand, expanding on it and then designing features which solved each part of the problem. The motivation behind this study is that the path planning algorithm could be applied to real world applications once perfected to a certain degree. The algorithm was composed of two main sub-algorithms: an off-line component and an on-line one. The off-line component was a global path planner which used a genetic algorithm to find the optimal path between the source point and the goal point while only accounting for static obstacles. The second component was a local path planner utilizing artificial potential fields in order to adjust the path at hand to account for any dynamic objects. Multiple configurations were tested and evaluated. The algorithm was compared to an Ant Colony Optimization algorithm and it proved to be more efficient with respect to the maps used in the study. The algorithm was evaluated for advantages and limitations and future study aspects were identified. Future works included: expanding the algorithm to allow for more robots, implementing the algorithm to a real life application and recreating miscellaneous situations to test the flexibility of the algorithm.

LIST OF ABBREVIATIONS USED

GA	Genetic Algorithm
GEN	Generation
ACO	Ant Colony Optimization
DNA	Deoxyribonucleic acid
C/O	Cross Over
SPCO	Single Point Cross Over
TPCO	Two-Point Cross Over
SCO	Scattered Cross Over
GAITP	Genetic Algorithm with incremental turning points
GAPTP	Genetic Algorithm with the pre-set turning points
SACO	Simple Ant Colony Optimization
ACS	Ant Colony System
ACO-MH	Ant Colony Optimization- Meta Heuristic
TP	Turning Points
NLA	No Leader Assignment
SRS	Simple Ranked System
RRLA	Repeated Random Leader Assignment
CTG	Closest to Goal
FTG	Farthest to Goal
cm	centimeter
sec	seconds

ACKNOWLEDGMENTS

First and foremost, I would like to thank God. Next, I would like to express my gratitude to my supervisor, Dr. Jason Gu for his continuous help, motivation and support throughout this thesis. His guidance has helped me at all stages of the research and during the writing of my thesis.

I would like to thank my parents for their continuous support and for being my rock throughout this journey. Thank you for keeping me going even when I thought I could not.

Afterwards, I would like to thank the rest of my committee members: Dr. William Phillips and Dr. Kamal El-Sankary for their encouragement, constructive feedback and insight.

I would like to thank my fellow lab mates in Dr. Gu's graduate group for their constructive feedback and questions throughout my research process. Having an outside view helped in identifying problems which I couldn't see on my own.

Finally, I would like to thank all my friends in Halifax. Your presence contributed to me keeping my sanity and actually finishing my work.

CHAPTER 1 INTRODUCTION

1.1 INTRODUCTION

This thesis develops a modified path planning algorithm which attempts to solve a global path planning problem for a mobile robot. The algorithm is developed from the ground up based on the requirements presented and the problem identified. The scope of this thesis was made to include static environment with purely non-moving obstacles, as well as changing environments which included both static and dynamic obstacles. The algorithm can be identified as a two-part algorithm. One part is the offline global path planner achieved with a genetic algorithm tool. The second part is an online path adjustor which is based on the artificial potential field method. The algorithm was adjusted to account for both single and multiple robot cases, and could be scaled even further if required.

The goal of this chapter is to act as the introductory portion to this thesis. The problem is introduced. Next, the field of study is related to and the motivation is made clear. This chapter also serves to introduce the necessary background information regarding path planning as a problem. The outline of the thesis is stated in this section.

1.2 BACKGROUND

Mobile robotics is a field which is researched heavily in our world today. This is because of the fact that mobile robots are able to perform man-required tasks through

automation [8]. Specifically, the sub-field of path planning is of interest. This field focuses on how a mobile robot will move within its environment.

The problem at hand is to find a path between two points: the start point and the goal point. The basic requirement for the path is that it must avoid any obstacles within the given environment [11].

Path planning varies from one algorithm to another. Path planning could be classified as two main categories: local and global. Local refers to path planning which is done on the spot, purely on a situational basis [8]. The robot senses an obstacle, and evaluates its local environment and moves in response to that. Global path planning includes the entire map as the environment, giving the robot the information within the entire map to act upon [9]. If both were combined, because of the nature of the global path planning needing the entire map as input, it will be used first before local path planning. As a result, it is safe to say that the better the path achieved through global path planning, the less dependency the algorithm will place on the local path planning component.

While both path planning types have their own advantages and disadvantages, the algorithm developed within this thesis will be primarily built upon global path planning. Local path planning will be still considered and added to the algorithm when needed.

1.3 THESIS MOTIVATION

The main motivation behind this thesis is that mobile robotics could be used in hazardous industrial applications [10]. Industrial applications, could be safer if mobile robotics were to replace the human operator aspect, both for the human and at the same time, achieve better results with high precision [2]. Other applications may also include

areas which are too hazardous for a human operator. These may include: fiery areas, extremely highly pressured areas or other areas with dangerous environmental factors [8].

In addition, more applications could be the basis for the motivation for mobile robots. Other than hazardous areas, there are environments which would benefit from the presence of a robot from an efficiency perspective. This could include environments which need service, but are too difficult for the human operator. An example of this could be air duct cleaning [3]. Due to their tight spaces, a human operator could have difficulty servicing these areas while adhering to the proper safety guidelines. While the size of an adult human operator could prove to be a limitation, a mobile robot could be rebuilt in many size to fit multiple areas, depending on the task at hand.

Another motivation would be the researched field of automated vehicles. The development of self-driving cars has been growing more popular over the years. The better the algorithm, the more it is likely to simulate the human intuition generated solution [8]. With the applications of global positioning systems (GPS) and satellite imagery, a global map for the car's environment could be generated. This global map would include static obstacles such as the road structures or any nearby buildings.

Another possible application for this algorithm would be aerospace travel. Air traffic control towers have an overviewed map of the aerial space and what planes occupy what space. Because of that, global path planning could be applied to this field. However, obstacles in this field could include other objects other than other planes [12]. Obstacles may include: areas to avoid such as windy areas, high turbulence areas, etc…

For these situations, a solution could be given via solving the global path planning problem. This is because an accurate map of the environment could be provided (road

maps, aerospace maps) [13]. That, in addition to a clear, predefined source point where the mobile robot is starting from and a predefined goal point which the robot is trying to reach.

That being said, the real world applications require the consideration of dynamic changes within the environment. In the car situation, moving obstacles such as other cars, pedestrians and other external factors.

In the case of the aerospace situation, other planes moving could represent dynamic obstacles [6]. In addition to this, weather conditions which are constricted within a specific area, could also be defined as an obstacle form. An example of this would be an ongoing storm which is defined within a specific geographical area and is moving within a specific direction that is both predicted and monitored in real time [13]. Current world air traffic controls allow the monitoring of such conditions.

From an engineering point of view, the main fundamental requirement for a self-driven car would be to reach its assigned destination safety. In order to do that, any obstacle collisions must be avoided and prevented. This makes obstacle collision the fundamental requirement in this path planning problem.

After obstacle collision is identified as the primary requirement, secondary requirements could be identified. The path length should be taken into consideration. The path planning problem is aimed for the robot to travel from points A to B. This would imply that the shorter the path, the more plausible the algorithm will be. In addition, the shorter the path, the less time it will take the robot to make the trip. A path planning algorithm should be aimed to generate generically short paths to be taken.

Another secondary requirement for the algorithm is its efficiency. In this case, efficiency refers to the computational cost which the algorithm needs in order to perform its assigned task. While taking obstacle collision and path length into consideration, the computational cost of the algorithm has to be taken into account. If the algorithm is computationally expensive, but generates a path which is not significantly better than its competition, then it loses its advantage. A plausible algorithm should be balanced in terms of the time it takes to execute and the quality of the results which it produces.

1.4 THESIS OUTLINE

Following this introductory chapter, chapter 2 gives an overview on the field in question that is mobile robotics. Afterwards, chapter 3 provides an overview on genetic algorithms, how they operate, and their relation to global path planning as a problem. Chapter 4 includes the experimental setup for the algorithm, as well as the algorithm development order. Chapter 5 presents the results attained from the experiments. Chapter 6 is the conclusion attained from the results, as well as the possible areas of future works on the algorithm.

CHAPTER 2 OVERVIEW ON MOBILE ROBOTICS AND PATH PLANNING

2.1 INTRODUCTION

Mobile robots are a major application in the modern world. The specific type of robots which this thesis focuses on are wheel-type. This would be the robots which have a range of motion similar to that of a car. Within the introduction, an argument was made for the application of the algorithm on aerospace vehicles, that is also the case. The algorithm could be applied on robots which have a similar range of motion. The purpose of this section is to give a descriptive overview on the global path planning problem with regards to differential robot.

2.2 OVERVIEW ON GLOBAL PATH PLANNING AND RELATED WORKS

Global path planning describes a situation in which a robot is given the task to plan and follow a path from points A to B [8]. This task has to be done while avoiding any obstacles present within the environment and while remaining within the boundaries of the environment. What makes global path planning unique is that the map data is fed into the algorithm for the path to be planned out.

While this is ideal for static obstacles, the case of a moving obstacle must also be taken into consideration. In order to do this, the mobile robot must be equipped with at

least one type of sensor in order to sense the location of any obstacles that may have displaced since the initial path has been planned.

Once the differential robot has sensed that it is headed for collision with a displaced obstacle, changes to its path must be made. This is where the path adjusting takes place in order to still reach the goal with the shortest distance possible, while still being able to avoid all obstacles within the environment.

2.3 EVALUATION CRITERION

2.3.1 Evaluation Criterion 1- Obstacle Avoidance

Obstacle avoidance is the primary criterion of evaluation for the path planning algorithm. This is because of the fact that if the robot does indeed collide with an obstacle, it has failed its purpose in finding an obstacle-free path. Avoiding all obstacles is the main motivation behind this type of path planning situation [11].

The robot has to avoid all obstacles, taking into consideration their shape, size and orientation. The algorithm plans a path for the robot to move around the obstacle to avoid it, while still moving towards the goal point. Some algorithms plan a path to trace the obstacle faced until it is no longer obstructing the path to the goal. While other algorithms plan a new path as a response to the presence of the obstacle, avoiding it altogether.

2.3.2 Evaluation Criterion 2- Path Length

Path length is the second criterion of evaluation for the global path planning problem. Ideally, an algorithm producing the shortest possible path between the source and the goal is the desired case [1]. From the previous section on obstacle avoidance, a straight line will not be the common solution due to the presence of obstacles within the

environment. As a result of that, the path planned will most likely have turning points in which the robot changes its direction and its orientation.

Since one of the motivations behind automated path planning is to mimic the most logical path potentially planned by a human, the path has to be intuitively short. For that reason, any useless motions taken by the robot within its path has to be taken into consideration. An example of that would be the robot making a perfect loop and then continuing with its path towards the goal. Unless the loop was done to avoid an obstacle, it increases the path length needlessly.

2.3.3 Evaluation Criterion 3- Algorithm Efficiency

The third most important criterion in evaluating the quality of a path planning algorithm is its efficiency. After the first two criterions are considered, the algorithm at hand is one which produces obstacle-free paths with their lengths taken into account.

The next item to be evaluated is the computational cost of the algorithm. That directly refers to the time the algorithm takes to complete and have a path planned out. This is an important thing to consider because it is possible for an algorithm to have a computational cost outweighing the results it produces. Ideally, it is desired to have an algorithm producing good results with low computational cost. However, that is usually not the case. As such, algorithms have to be weighed in and compared against one another according not only to the quality of the results they produce, but how long it takes them to produce said results.

Global path planning refers to cases where the entire map is given as an input to the algorithm. This initially was the entirety of the study scope of this thesis. However,

due to the simple nature of static obstacle avoidance as a problem, the scope of the thesis was widened in order to include dynamic obstacles as well.

CHAPTER 3 OVERVIEW ON GLOBAL PATH PLANNING USING GENETIC ALGORITHM

3.1 INTRODUCTION

The purpose of this chapter is to provide an overview on genetic algorithm and their applications in path planning. The concept of genetic algorithm is derived from models mimicking real-world evolution processes. The model represents the idea that within a specific species, the ideal candidates for survival are the individuals with the heritable traits suited for the surrounding conditions. These traits are then passed on to the following generation of the species which will be more adaptable to the conditions [14].

3.2 BIOLOGICAL EXAMPLE

An example of this could be given through a population of a moths. The population of moths included two color variations: black and peppered. With the environment being the industrial world, the white moths had a higher chance of being preyed upon. This is due to the fact that visually they stood out more in the environment, unlike the black moths. This resulted in an increase of black moths and a noticeable decrease the in the white moths [4]. Within this example, a population was introduced. Afterwards, the population responded to the environment it was presented with, adapting with it. This particular example was part of a series of experiment performed by Kettlewell in the 1950's [4].

3.3 OVERVIEW ON GENETIC ALGORITHM

The basic concept of a genetic algorithm is to start with an initial population. The population is then evaluated for its fitness, per individual, according to a specific grading criteria. Afterwards, through the application of certain genetic operators, the second generation of the population would be produced. The second generation would supposedly be better than its predecessor when being evaluated for its fitness. This process would repeat, until a specific number of generations is reached, until a certain fitness is reached or another stop condition could be placed on the genetic algorithm [14]. In order to explain the concept of genetic operators, the concepts of phenotypes and genotypes have to be explained.

Phenotype is the physical description of the genetic trait. In other words, phenotype is the "non-coded" description of the individual. Genotype represents the coded information which the phenotype is trying to describe [14]. An example of the difference between these two factors could be shown when describing a frog population. The frog's DNA and genetic markers are considered its genotype [14]. However, its phenotypes would include aspects such as the frog's body color, eye color, limb length, tongue length.

This is applied within a genetic algorithm, through having a population with its individuals genetically marked. An example of this would be having the individual shown below:

Individual 001: 0100110010

The number shown above represents the genetic identifier of the individual. Each number location corresponds to a specific marker. This would be the genotype of the

individual. In the nature of the global path planning problem, an individual will represent a specific path. Each number within its genetic identifier could correspond to a specific locational node which the path crosses if it is 1 and doesn't if it is a 0. This would be one example as to how a path could be genetically encoded into the genetic algorithm.

Afterwards, through identifying the genetic markers of each individual, and evaluating their fitness, it is possible to select which genetic markers give the individual the best fitness value. These traits could then be passed on to offspring individuals which would have mixed traits from its parent paths. Thus is the concept of crossover.

3.3.1 Genetic Operator: Crossover

Crossover is the most basic genetic operator. It is such because it is the one most witnessed within nature within the real world. A real life example would be the genetic heritage of certain traits from one's parents. An example of a genetic trait would be eye color. The eye colors of both parents contribute directly to the eye color of the offspring. While some traits are recessive, requiring both parents to have it for it to be passed on, or even dominant, requiring only one parent to have it, the concept of crossover is allowing all of it to happen. A more fitting example would be having two parent individuals, each with a specific genetic advantage. From an evolutionary point of view, the ideal crossover would exchange the genes of both parents in a constructive fashion. This would allow the offspring to be born with both genetic advantages, achieved from both parents [14].

3.3.1.1 Single Point Crossover

An example of this could be presented by looking at both individuals presented below:

Individual 001: 0100110010

Individual 002: 1001001101

If individuals 001 and 002 represent the population at its initial generation stage, a simple crossover operator would be setup to produce two children which would be individuals 003 and 004 shown below:

Individual 003: 0100101101

Individual 004: 0100110010

The highlighted section in Individuals 003 and 004 represent the genotypes which were crossover from the parents. The example above was a simple crossover which took half the genetic markers of each individuals and exchanged it with the one from the other parent. If this was a constructive crossover operator, then individuals 003 and 004 would have higher fitness than individuals 001 and 002. Afterwards, the children individuals would replace the parent individuals within the population. This process is repeated multiple times, creating new generations of individuals with their respective fitness getting higher with each iteration.

Crossover is an operator which allows the genetic algorithm to create individuals based on the fitness of the previous generations. As such, with old generations being removed from the population, the overall population would ideally increase in its average fitness. This would also mean that the best individual within the population with each generation would increase in fitness. Ideally, this would indicate that the genetic algorithm has found the solution to the problem presented.

Single point crossover describes an operation which switches the parents' genotypes past a certain point. The example presented earlier with how Individuals 003

and 004 were formed with their respective parents, individuals 001 and 002 is a single point crossover example.

3.3.1.2 Two-Point Crossover

Another type of crossover is the two-point crossover. For this particular crossover, the genotypes are crossover between both parents between two particular locations. An example of this is given with individuals 1 and 2, which are reproduced below:

Individual 001: 0100110010

Individual 002: 1001001101

For the case of a two-point crossover, only genotypes present within a certain range would be crossed over. For example, if the two-point crossover was designed to crossover only the genotypes present between genotypes 2 and 6, then the crossover would yield individuals 005 and 006 shown below:

Individual 005: 0001000010

Individual 006: 1100111101

The crossover genotypes are highlighted on the offspring individuals. Two-point crossover could be setup to explore only certain genotypes which could lead to a more controlled evolution of the population. However, it could also leave certain genotypes unexplored because of its specific targeting capabilities. This could potentially lead to a delayed evolution.

3.3.1.3 Scattered Crossover

The third type of crossover to be considered is scattered crossover. For this particular kind of crossover, a random binary vector is created in which the

corresponding genotypes are selected according to the vector's 1 values. For example, if individuals 001 and 002 were to be crossed over using this method, a random binary vector needs to be assigned, as shown below:

Individual 001: 0100110010

Individual 002: 1001001101

Vector: 1000110011

The vector has the same size as both parents' genotypes, indicating which genotypes are to be crossover with the location of its 1's. For this specific example, the resulting crossover would be as shown below with individuals 007 and 008.

Individual 007: 1100000001

Individual 008: 0001111110

The highlighted genotypes in individuals 007 and 008 are a result of the scattered crossover. The example presented above shows that scattered crossover is even more targeted than two-point crossover because of its dependency on the selection vector. Once again, this could allow for the specific targeting of genes, but could prove a hindrance if the right genes are not selected by the vector. Within genetic algorithms, the concept of mutation is also one to be considered alongside crossover.

3.3.2 Genetic Operator: Mutation

Mutation is another genetic operator used in GA. While it is less common to witness than crossover in the real world, there are still a number of examples. An example of that would be heterochromia. While both parents had two distinct eye color, the offspring had an entirely new genetic trait of different eye colors in each eye. While mutations are rarer to witness, they can still occur.

As the name suggests, mutation implies that the operator would divert from what is expected. Mutation can be described from the point of view of a genetic algorithm by referring back to the individual presented earlier (shown below):

Individual 001: 0100110010

The mutation operator could vary in many ways. The simplest one would be an operator setup to flip the first genetic marker of the individual [14]. As a result, the individual would be:

Individual 001 (After mutation): 1100110010

. The highlighted genotype shows the mutated genotype in Individual 001 after mutation. Mutation could operate to introduce new traits which would be otherwise unattainable through crossover alone. Potentially, this could cause a bigger jump in fitness values, if the mutated individuals are better than their previous predecessor. However, it is also true that mutation could introduce new faults within an individual. This would be shown through the mutated individual having a lower fitness than its previous predecessor.

With the combination of mutation and crossover, the GA is able to generate individuals which possess higher fitness than the ones present within the previous generations. This process is repeated with each iteration, causing the average fitness as a result. As the fitness increases, the GA presents a case of convergence within its total fitness values. This is because of the fact that with each generation, the top percentile is selected to occupy the new generational pool according to their respective fitness. The fitness convergence could be used to evaluate how well the GA is behaving.

3.4 PATH PLANNING GA

There are several advantages to using GA to solve a path planning problem. Firstly, GA supports multi-objective tasks. If the path solution is required to have several properties to it, a GA could be constructed to meet those needs. An example of this would be a path requirement of obstacle avoidance and for the path length to be within a certain limit, due to fuel resources of the robot. A genetic algorithm could be custom made in order to fit those particular needs, and bias each requirement to come up the ideal path needed.

3.4.1 Advantages of Using GA for Path Planning

The second advantage is that the GA is improved upon with time. The main feature of the GA is that it depends on generational iterations. As such, with more generations, a better result is expected. However, there is a limit for how much improvement the solution will have. Eventually, no significant improvement to the solution will be made with increased generations. With more generations, the GA is able to introduce traits discovered and eliminate weaknesses it identifies. Any traits which increase the fitness of the individuals is more likely to be passed along to the next generation of paths. At the same time, any weaknesses which lower the fitness of paths will not be passed along and will be eliminated as a result.

3.4.2 Disadvantages of Using GA for Path Planning

While there are advantages to using genetic algorithm, there are a number of disadvantages to also consider. These are important to have in consideration in order to attempt to effectively solve the global path planning problem.

The first disadvantage is that the genetic algorithm does not guarantee that the global maxima will be attained. The genetic algorithm is set up in a black box manner for it to run for a specific number of generations or until a certain stop condition is met. However, the possibility exists that the genetic algorithm could loop around itself in effectively or even produce a solution which does not solve the problem.

The second disadvantage would be the ideal parameters for the GA could vary depending on the problem. These parameters include, but are not limited to: population size, maximum number of generations, number of genetic markers and genetic operators used. These are all parameters which control how the genetic algorithm operates. However, as to how the parameters could be setup best for the algorithm, that would be achieved through trial and error.

Thirdly, the black box fashion of the algorithm poses another disadvantage as the user is not certain of what the algorithm is learning. Since the genetic algorithm is run in a black box fashion, this means that once all the parameters are set up, then the algorithm runs and the fitness results from each generation are shown. The algorithm then produces a solution which it believes to satisfy the conditions of the problem set up by the user. The user has to make sure that the algorithm is indeed solving the correct problem intended. The genetic algorithm could be connecting dots and making connections where there isn't any reason to.

Another disadvantage is the time the genetic algorithm takes to converge with its results. This refers to when the algorithm is running, producing the fitness values for each generation. The time that the algorithm takes to converge with.

Another feature of GA’s to consider is the aspect of exploration versus

exploitation [14]. If the GA is setup mainly for exploration, that means that the search space which the GA is working with is larger than normal. The GA would then be tasked with simply finding all possible features which could be found within the population search space. This would usually include exploring all possible features to be explored in order to increase the fitness of the individuals. On the other hand, exploitation is the opposite of that concept. Exploitation involves limiting the search space to a defined limit. This would cause the GA to focus on specific features and improve on them through each generational iteration. This would be best shown in a GA focusing only on a single genotype to improve within its given population. The balance between exploration and exploitation is needed for a well-functioning GA, as both are needed. Exploration is needed, as it gives to GA the advantage of discovering new features which would simplify the solution, if one is indeed discovered. On the other hand, exploitation is also needed, because the iterations are needed for the GA to improve on the solution attained by the GA. Including the previous work done in GA, Sugihara and Smith (1977) explored the application of a fixed path length evolutionary scheme [8]. It was found that the respective algorithm performed well in simple environments, however, required hours to complete a solution for the complex environments. Tu and Wang (2003) suggested a variable length evolutionary scheme, which proved to be more flexible, however yielded a 'no solution' for some test scenarios [8].

Since the idea of a GA was explained in the previous section, it is known that the global path planning problem will be solved by first selecting an initial population of a possible solution. These candidate solutions are then evaluated for their fitness. Afterwards they are run through the GA and undergo crossover and mutation through

generations of data. After the final generation is born, the best individual of that generation is expected to be the ideal solution for the algorithm.

The initial population would be locational nodes present within the environment. Each mode would represent a trait for the individual within the generation. The individual would be a path.

This allows for the paths to crossover with one another, thus partially editing the path as needed. Mutation could introduce new nodes which could potentially be unattainable if crossover was applied purely on its own. The number of nodes would represent the number of traits each individual in the population has. This would be an essential parameter in setting up the GA.

The probabilities at which the crossover and mutation operators take place would also have to be specified. This would allow the GA to know how to move from one generation to another. In addition, a maximum number of generations needs to be specified in order to tell the algorithm when to stop. If this limiting parameter is not implemented, then the algorithm could attempt to run indefinitely, attempting to further improve the best path at hand with each generation. While this could promise a better path result potentially, it certainly would be more computationally expensive, thus reducing the efficiency of the algorithm.

CHAPTER 4 EXPERIMENTAL SETUP AND ALGORITHM DEVELOPMENT

4.1 INTRODUCTION

This section provides a detailed description of the experimental setup used to build, test and troubleshoot the algorithm. The section is structured chronologically in order to show the thought process involved in the algorithm development and in order to provide the justification and reasoning behind each action taken. This section introduces algorithm features and why they were implemented. Their respective testing will be provided in Chapter 5.

4.2 GLOBAL PATH PLANNER: EMPTY MAP

The first step to developing the algorithm was to build it in its most basic form. That is, it was required to design a simple path-planning algorithm which would plan a path from point A, the source to point B the goal. The map used included no obstacles at

all. With the GA tool used in MATLAB, the current setup allowed the algorithm to draw a straight line between both the source and the goal point.

4.3 GLOBAL PATH PLANNER: STATIC MAP

After the algorithm was shown to function as expected, the empty map was populated with more details. These details only included static obstacles. The GA had to be modified in order to find a path which had the condition of not having any points occupy a point coordinate with a static obstacle on it. In order to do that, the fitness function of the GA had to be further specified. Since the GA functioned by populating a pool of paths, with their coordinates as the genotypes of each path, this was utilized in order to bias the GA towards obstacle free paths. This was done by assigning a penalty value to paths which had at least one section occupying a static obstacle location. The penalty value served as an indication that the path in question is of an extremely low fitness value. As a result, the path with the penalty value would have its probability of passing on its features extremely reduced. In addition, should the path manage to get selected to path its traits to the future generation of paths, the offspring paths would also be evaluated with the same condition, and would have the penalty imposed on them as well. This would further reduce the probability of a faulty path section being passed on to the final generation produced by the GA. The fitness function model used to achieve this process is shown below:

$$f = \sum_{i=1}^{n-1} d(A_i, A_{i+1}) + \sum_{i=1}^{m} penalty_i \qquad (1)$$

The penalty is zero for unobstructed paths, but is set to an extremely high value for obstructed paths. The GA implemented in the previous section was used in order to find

the shortest, obstacle-free path from the source point to the goal point. The GA would populate a pool of paths which are randomly generated. All of the generated paths have a pre-defined number of turning points. The turning points represent the genotypes of the paths and how they differ from one another. Following that, the fitness of each path is evaluated. The fitness evaluation is dependent on whether the obstacle is path-free or not, and its total length. If an path was even partially crossing through a static obstacle, its fitness score would be penalized severely, effectively making the path invalid. Afterwards, the paths undergo the crossover operation, discussed in the previous chapter. The new offspring paths are then evaluated for their own respective fitness. Then, the top percentile of paths would be selected to populate the new pool according to their fitness. This process was then repeated multiple times, producing a new generation of paths with each generation. After the limit of generations has been reached, the GA produces the path in its current pool with the highest fitness as the output. Since the GA is being used as a tool to generate the initial paths, single point crossover was used for simplicity purposes.

4.4 GA PATH PLANNER: NUMBER OF TURNING POINTS

The GA had to have the number of turning points defined in order to occupy its pool with the initial population of randomly generated paths. For that reason, some testing had to be done in order to know the optimal number of turning points to be specified. An experiment was designed for that purpose. The purpose of this experiment was to determine whether it was better to have a relatively high number of turning points, or to

set the number of turning points as an incrementing variable. While the pre-set high number case would almost always guarantee a path, forcing the GA to occupy the pool with paths with high number of turning points would increase its processing time significantly. This is due to paths having more genotypes which would be crossed over with each iteration. Another point to consider was that an increased number of turning points means increased path features, some of which could be unneeded. This would require a higher generation limit in order to filter the unneeded features out. This would also cause an increase in the algorithm's computational cost. On the other hand, having the number of turning points set at a lower value, only incrementing if a path could not be found with the current settings could be considered. It is expected that an incrementing value would provide a simpler path as it would guarantee the minimum number of turning points. However, if a path was not attained, and the number of paths had to be incremented, that would mean that the GA would have to run again. This repeated operation could add to the computational cost of the algorithm. It is for that reason that a comparison experiment needs to be put in place in order to determine the best course of action.

4.5 ROBOT CONTROLLER IMPLEMENTATION

Afterwards, the next step was to add in a robot which would follow the path planned by the GA planner. In order to do this, the Pure-Pursuit tool MATLAB tool was used. The tool allowed the creation of a driver robot as an object, with a specific speed, orientation and starting position. The robot is then instructed to follow the path planned out for it, while still adhering to the driver settings that have been set on it, such as its driving speed.

4.6 MULTIPLE ROBOT IMPLEMENTATION

Following that, the code attained up to this point was used in order to create the multiple robot experiment. The GA was used in order to create paths for 3 robots in total given the same map, but different source and goal points for each respective robot. Each robot had its own controller which follows the respective path planned for it by the GA.

With the implementation of multiple robots, the effect on the GA path planner had to be considered. With more robots in play, it was decided to run the GA for each robot in parallel. This is due to a number of reasons. The first reason is that if that were not the case, the GA would have had to been reconfigured in order to find all paths for each robot which would mean three times the pool and three times the computational cost. That computational cost would scale even higher with the addition of more robots. The second reason was that in the case the algorithm was unable to find a solution for even one of the robots, the other robots would be left without action, waiting for all paths to be found, causing increased delayed time in the algorithm's output. Finally, if 1 GA was managing the paths for all robots, with the addition of more paths, the paths for the robots would become more complicated. With more paths occupying the map, there are less open spaces for the algorithm to map out new paths for the remaining robots. Having the GA run once for each robot, fixes the scalability and also allows for each robot to independently operate. The multiple robot flowchart setup for the series and parallel cases is presented in Appendix A.

4.7 POTENTIAL FIELD PATH ADJUSTOR

From there, a problem was spotted within the algorithm. That problem was potential robot-robot collisions. While the GA accounted for the static environment, the paths generated did not account for robots occupying the same point in space at the same time. While it was possible to redesign the GA to have multi-objectives, the complexity of the algorithm would increase drastically with the addition of more robots, the variability of the maps and if other changes were to be applied. With consideration to that, the solution attained to this complication was to design an online path adjuster which would run when needed case-by-case in the event of an upcoming collision.

The concept of potential field is defined in Khatib's 1986 Paper entitled: "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots" [5]. The robot moves affected by different "invisible" potential fields. It is the net force generated by these fields on the respective robot which decide on its directed path. To design this, the goal point described to each robot had an attractive potential fields towards it. In addition, several repulsive fields had to be implemented.

To start off, each robot had a repulsive field emitting from it. This allowed the robots to move away from one another, in the case that they get too close within the influential radius of one another. Furthermore, a repulsive field was added on the environment itself. More specifically, a repulsive field was created on the static obstacles present within the environment. This was possible since the map data was provided to the algorithm as part of the global path planner used earlier. This specific repulsive field was not created so as to prevent the robot from colliding into static obstacles. Rather, it was created to act as a biasing agent. Should the online path adjustment get triggered to change the path of one robot about to collide with another, the static obstacle repulsive field will encourage the

dodging to take place away from areas with heavier densities of static obstacles. This would be shown visually by observing that the robots dodge one another while moving towards areas which are more “obstacle-free”. The purpose of this was to avoid any post robot-robot complications. One possible complication was that the robot could end up in a worse position than its earlier state due to it performing a dodge with another robot.

4.8 LEADER ASSIGNMENT FEATURE

After the concept of potential field was explored in order to prevent any robot-robot collisions, another complication was noticed. That complication is that due to the robots operating with the same settings, their respective repulsive fields were also equal in magnitude. This caused any 2 robots heading for collision to perform similar dodging motion, in opposite directions from one another. However, that action would trigger another dodge order once the robots were within each other’s range again. This was observed most clearly in robots which have their overall paths close to one another, with their respective source and goal points also fairly close. As a result of this complication, the robot-robot dodging caused multiple path adjustments which significantly added to the total path length travelled by both robots. A fix had to be implemented in order to solve this complication, as that would have been a significant limitation to the algorithm if left unfixed.

The fix implemented for the complication above was to assign a ranking system. Every time a collision is set to take place, the online path adjuster will utilize a certain ranking system which assigns one robot to be the “leader”, and the other robot to be its “follower”. The follower robot will be the only robot to perform the dodging action, while the leader robot will continue on its path unchanged.

The leader assignment implementation is a feature which is introduced into the algorithm in order to solve the complication caused by robots dodging one another equally. The next step was to decide on how the ranking system was achieved. For troubleshooting purposes, the ranking system was pre-assigned or was randomly selected through a randomly generated number.

However, for the application of the algorithm, there needs to be a logical reason for as to the ranking system was chosen. One ordering was achieved through the robots' creation order. In other words, robot 1 would be the leader should it encounter robots 2 or 3; while robot 2 would be the leader should it encounter robot 3. This method was too simple and did not account for the current positions of the robots. As a result, the respective distanced each robot had to reach its respective goal was taken into consideration. If a robot was closer to its goal than the other robot it was headed to collide with, it would become the leader in that case. This ranking system allowed the algorithm to prioritize robots to complete their paths as soon as possible. Another ranking system was considered, assigning the robot farthest from its respective goal point as the leader. This biased the algorithm towards all robots finishing their paths within the same time period.

4.9 DYNAMIC OBSTACLE IMPLEMENTATION

The next step was to include dynamic obstacles within the map. In order to do this, the Pure-Pursuit tool was called upon. Basically, a dynamic obstacle would be created by creating another robot which follows a set path given to it. After the dynamic obstacle had completed its path, the robot would make a turn and follow its path in reverse

(switching its assigned source and goal points respectively). This would give the effect of a dynamic obstacle which patrols a certain path between two given points.

Unlike a fellow robot driving towards its respective goal point, the dynamic obstacle is different in the sense that it does not dodge any incoming obstacle. A dynamic obstacle will remain on its planned path regardless of any incoming obstacles. Its entire purpose is to be an obstacle in order to test the flexibility of the algorithm and its ability to handle dynamic obstacles placed within the map.

In order to implement dynamic obstacles within the leader assignment script described earlier within this section, the dynamic obstacles were automatically assigned the top ranking with the leader-follower rank. That would mean that a dynamic obstacle would never have to dodge, and that the robot had to perform the dodging task in order to avoid a collision.

In order to avoid dynamic obstacles, the potential field avoidance strategy was used once again. However, unlike a robot-robot interaction, different avoidance strategies had to be placed in order to deal with dynamic obstacles headed for collision. This is due to the fact that a dynamic obstacle will not dodge or avoid the robot no matter what. In addition, the path planned for the dynamic obstacle is not known to the robot in any way. Hence, different avoidance strategies had to be thought of and tested in order to deal with this complication.

Following this, the avoidance strategies used in order to deal with dynamic obstacles were explored. These included:

- *Stop and Wait*

This strategy included that the robot, upon sensing the dynamic obstacle within a certain influential radius, would stop its motors and wait. The robot would start again once the dynamic obstacle was no longer being detected.

- *Stop and Dodge*

This avoidance strategy included the stop part described in the earlier avoidance strategy. However, following the stop action, the robot would proceed to dodge the dynamic obstacle in a manner similar to how robots dodge one another if they get within each other's influential range. This strategy was also attempted without the stop aspect, which would make it identical to how the robots were designed to dodge one another within that version.

- *Back-Up/Reversing*

This avoidance strategy dictated that upon sensing the dynamic obstacle, the robot would plan to reverse its motion to an earlier position and then attempt its planned path again. The logic behind this strategy is to allow the obstacle to move out of the robot's way, without stopping the robot's motion.

Each strategy would be evaluated on its own through testing, in order to explore its strengths and limitations.

CHAPTER 5 EXPERIMENTAL DATA AND RESULTS

5.1 GA TEST: EMPTY MAP

This section is aimed to present the data and results performed by the experimental setup described in chapter 4. The first part of this section is to present the data attained from the single robot experiment. Figure 1 shows the GA path planner's output with regards to a map not containing any static obstacles. Note that for all the mapped paths figures, the circled points indicate the starting point for the robot, while the star marked point indicates the goal point. Each robot has a unique color assigned to it.

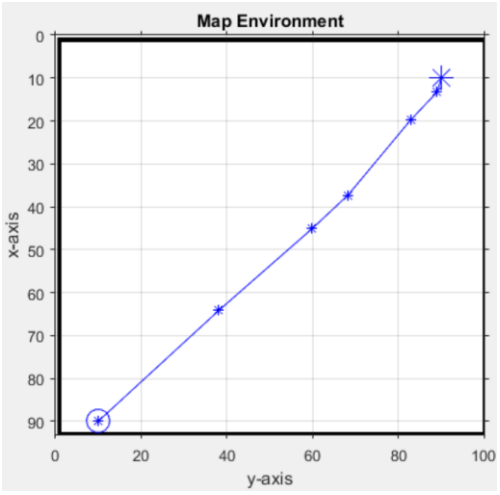


Figure 1: *Outputted Mapped Path for Test Scenario: Empty Map, Single Robot*

Corresponding path data is presented in table 1.

Table 1: *Outputted Path for Test Scenario: Empty Map, Single Robot..*

Path Length (cm)	111.50	
Path Coordinates	Y (cm)	X (cm)
	90.00	10.00
	64.16	38.11
	45.06	59.91
	37.27	68.23
	19.75	82.85
	13.25	89.02
	10.00	90.00

5.2 GA TEST: STATIC MAP 1

Figure 2 shows examples of how the static robot had its path planned from its source to its goal through the static environment.

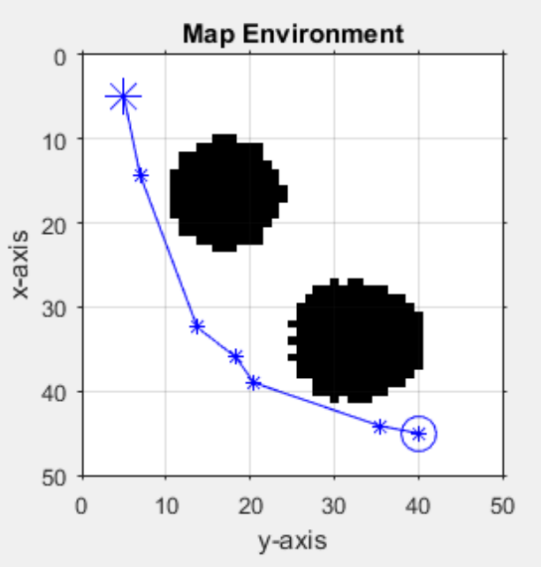


Figure 2 *Outputted Mapped Path for Test Scenario: Static Map 1, Single Robot*

Table 2: *Outputted Path for Test Scenario: Static Map 1, Single Robot..*

Path Length (cm)	61.00	
Path Coordinates	Y (cm)	X (cm)
	45.00	40.00
	44.16	35.51
	39.00	20.46
	35.96	18.29
	32.42	13.73
	14.30	6.93
	5.00	5.00

Table 2 shows the path coordinates visually presented in Figure 2. It can be seen that the GA's biasing against paths crossing through static obstacles is effective by comparing the paths from Figure 1 and Figure 2. The GA is shown to be effective in mapping out valid paths in accordance to the static obstacles provided. No path sections are crossing through a static obstacle.

5.3 GA CROSSOVER TESTING

Next, the genetic algorithm was to be evaluated. The GA was tested in each cases with 5 generations and 50 generations. Three different crossover types were used. The

crossovers used were: single-point crossover (SPCO), two-point crossover (TPCO) and Scattered crossover (SCO). Only the path length was used as the main criteria of evaluations. This is because of the fact that all of the trials were run with the same generation numbers. As a result, the processing time will be the same for all of them. Table 3 below show the total path lengths attained from the experiment. The data trials are shown in appendix C.

Table 3: *Total path Lengths attained through each crossover type*

Total Path Length (cm)		
# of Generations	5	50
Single Point C/O	67	55
Two-Point C/O	93.5	61
Scattered C/O	73	61

From the results shown in Table 3 previously, it can be seen that the trials with the 50 generations yielded paths with less length than those yielded by the 5 generation trials. This makes sense because with every generation, more crossover is applied on the paths which are then selected for their fitness and the process is repeated. The result from that process is the increased fitness of the paths within the population. From the results above, it can be seen that the single point crossover (SPCO) yielded the lowest of all path lengths in both the 5-generation case and the 50-generation case. As a result of that, the algorithm was continued with single-point crossover. Testing showed that varying the mutation probability did not have a direct correlation with the efficiency of the path generated. As a result, mutation was dropped as a genetic operator used by the algorithm

5.4 ROBOT CONTROLLER IMPLEMENTATION

Figure 3 shows how the robot controller handles the mapped path given. It can be seen that with the implementation of a robot controller, there is a base path deviation present between the path planned and the path followed by the robot controller. This is due to the robot settings set up by the controller.

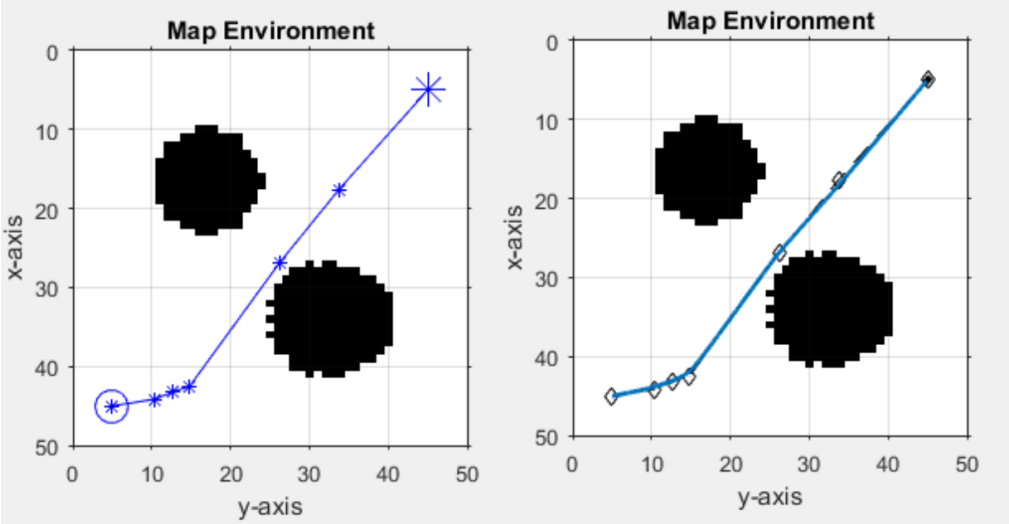


Figure 3: *Outputted Mapped Path for Test Scenario: Static Map 1, Single Robot. Robot Controller path shown on the right figure*

In Table 4, the % path deviation was shown to be 3.34% which is within the acceptable range.

Table 4: *Outputted Path for Test Scenario: Static Map 1, Single Robot. Path Mapped by GA and Path Followed by Robot Controller.*

	Y (cm)	X (cm)
Path Coordinates	45.00	5.00
	44.18	10.26
	43.30	12.58
	42.49	14.83
	26.81	26.26

	17.59	33.75
	5.00	45.00
Mapped Path Length (cm)	56.00	
Followed Path Length (cm)	57.87	
% Path Deviation	3.34%	

It can be visually observed that with minimum number of turns specified, the algorithm relatively finds the shortest distance between the source and the goal point. This is due to the fact that with minimum number of turning points specified, the more the robot is more likely to travel along a straight line to its destination. This minimizes the total distance travelled. Maps which are too simple benefit from this feature. However, with maps that are too complex, the algorithm will not be able to plan a path from the specified source and goal point with the current specified number of turning points. It is for that reason, that the variable number of turning points is considered. The results from that experiment will be discusses later within this section.

5.5 COMPARING GA TO ACO

The next section compared the Genetic Algorithm designed within this thesis with the ACO algorithm [7]. The second algorithm had to be similar in the task that it was solving, in order to have an unbiased comparison of the results. The second algorithm chosen was a set of two algorithms both based on Ant Colony Optimization [7]. Figure 4 and 5 show the mapped path outputs using [7]'s algorithms. Table 6 shows the evaluation results for these algorithms.

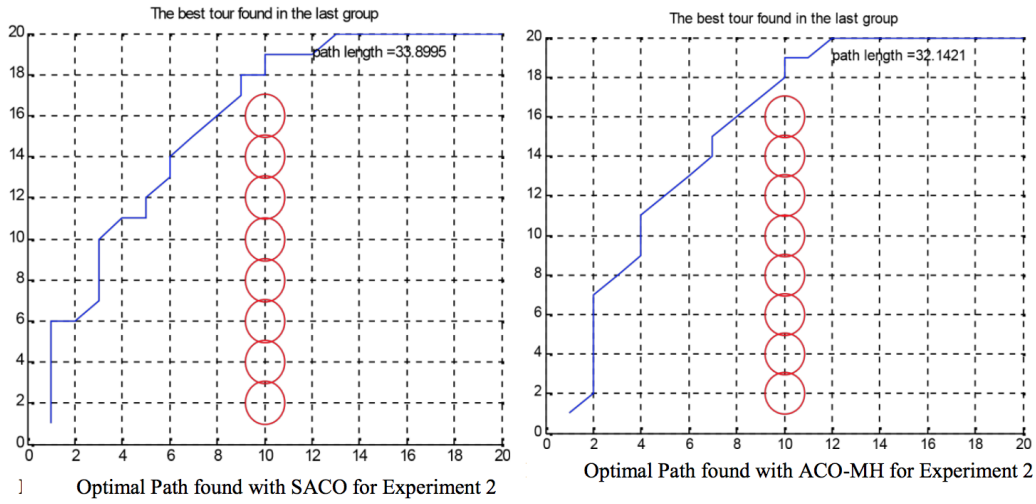


Figure 4: *A screenshot from [7] taken corresponding to Experiment 2 shown in Table 5. Screenshot shows planned paths using SACO (ACS) and ACO-MH Algorithms*

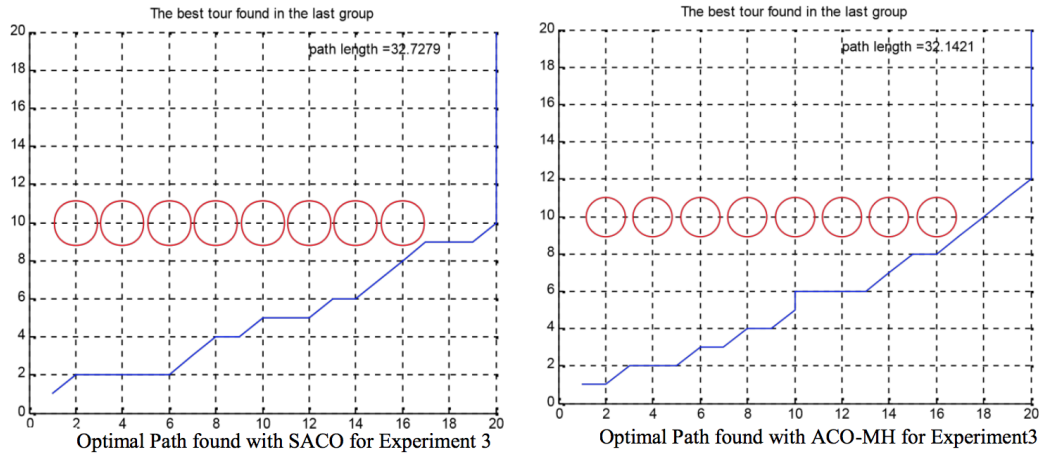
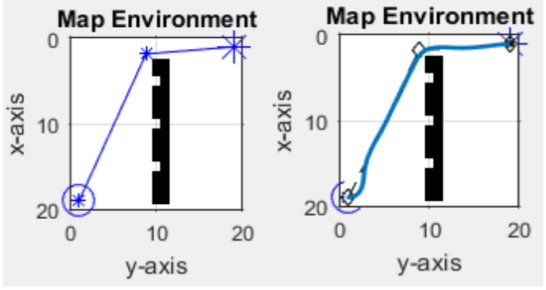


Figure 5: *A screenshot from [7] taken corresponding to Experiment 3 shown in Table 5. Screenshot shows planned paths using SACO (ACS) and ACO-MH Algorithms*

Figures 6, 7, 8 and 9 shows the bench testing done for the GA. The maps used were similar, with the same source and goal points. Bench testing was done on the GA with incremental turning points (GAITP) as well the GA with the pre-set turning points (GAPTP). This was done to further test the effectiveness of the incremental turning point feature.

Generations	TP	Time Cost (sec)	Path Length (cm)
5	1	1.41	28.5



Generations	TP	Time Cost (sec)	Path Length (cm)
5	5	4.55	40.5

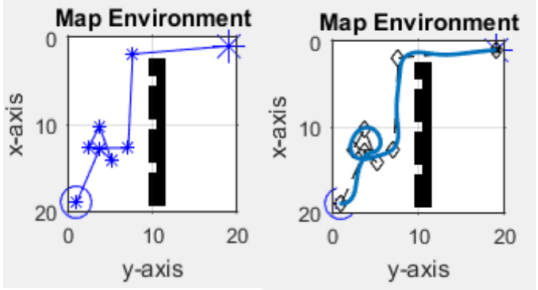
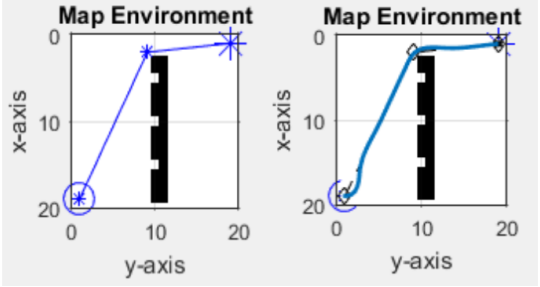


Figure 6: *Experimental Data showing the optimal path in a similar map to Figure 4. 5 GEN with 1 TP was used on the left, while 5 GEN with 1 TP was used on the right.*

Generations	TP	Time Cost (sec)	Path Length (cm)
50	1	10.01	28



Generations	TP	Time Cost (sec)	Path Length (cm)
50	5	11.35	27

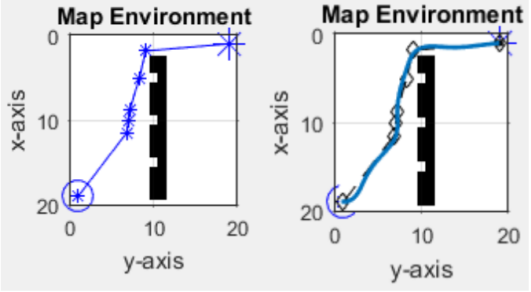
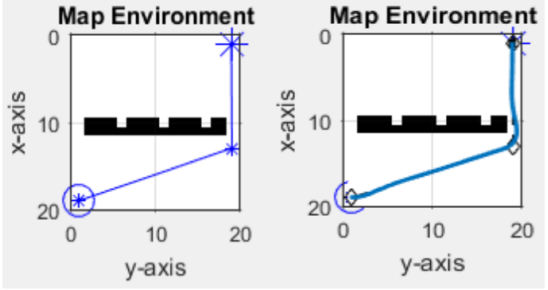


Figure 7: *Experimental Data showing the optimal path in a similar map to Figure 4. 50 GEN with TP was used on the left, while 50 GEN with 1 TP was used on the right.*

Generations	TP	Time Cost (sec)	Path Length (cm)
5	1	1.36	30.5



Generations	TP	Time Cost (sec)	Path Length (cm)
5	5	2.41	37.5

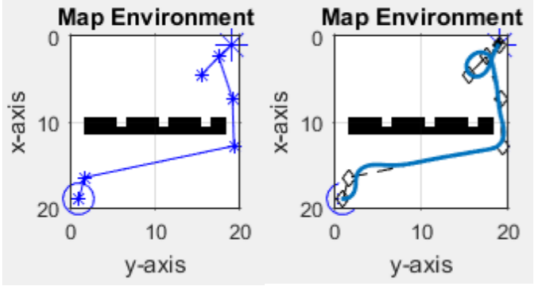


Figure 8: *Experimental Data showing the optimal path in a similar map to Figure 5. 5 GEN with 1 TP was used on the left, while 5 GEN with 1 TP was used on the right.*

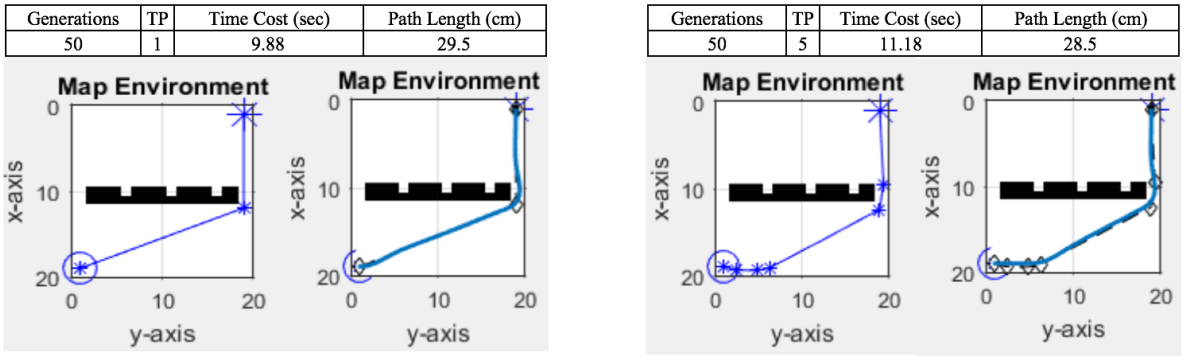


***Figure 9**: Experimental Data showing the optimal path in a similar map to Figure 5. 50 GEN with 1 TP was used on the left, while 50 GEN with 1 TP was used on the right.*

The location of the static obstacles was also replicated. Tables 6 and 7 shows the comparison results of the algorithms. It can be seen that within all GA's with a high generation limit, the time cost of the algorithm was relatively significant. However, these versions of the GA did produce paths which had relatively low path lengths. When a low generation setting was combined with a high turning point value, the result was undesirable. The result attained from this combination was a path which was extremely high compared to the other test cases. In addition, the time cost for the algorithm was not low either. Figure 10 visually represents the data attained from comparing the GA algorithms and the ACO algorithms in terms of their respective time cost and path length.

***Table 5:** Shows the results [7] attained from using the ACO algorithms.*

	SACO(ACS)		ACO-MH	
Ex No	Distance (cm)	Time(sec)	Distance (cm)	Time(sec)
1	31.80	3.67	28.63	3.27
2	33.90	6.57	32.14	6.90
3	32.73	2.40	32.14	2.37

Table 6: *Algorithm's Path Details with experiments 1 and 2*

Algorithm	GAITP				GAPTP			
Turning Points	1		1		5		5	
# of Generation	5		50		5		50	
	Distance (cm)	Time (sec)	Distance (cm)	Time (sec)	Distance (cm)	Time (sec)	Distance (cm)	Time (sec)
Experiment 1	28.50	1.41	28.00	10.01	40.50	4.55	27.00	11.35
Experiment 2	30.50	1.36	29.50	9.88	37.50	2.41	28.50	11.18

Table 7: *ACO [7] Algorithms Path Details*

Algorithm	SACO			SACO-MH		
	Distance (cm)	Time (sec)	TP	Distance (cm)	Time (sec)	TP
Experiment 1	33.90	6.57	15	32.14	6.90	10
Experiment 2	32.73	2.40	11	32.14	2.37	13

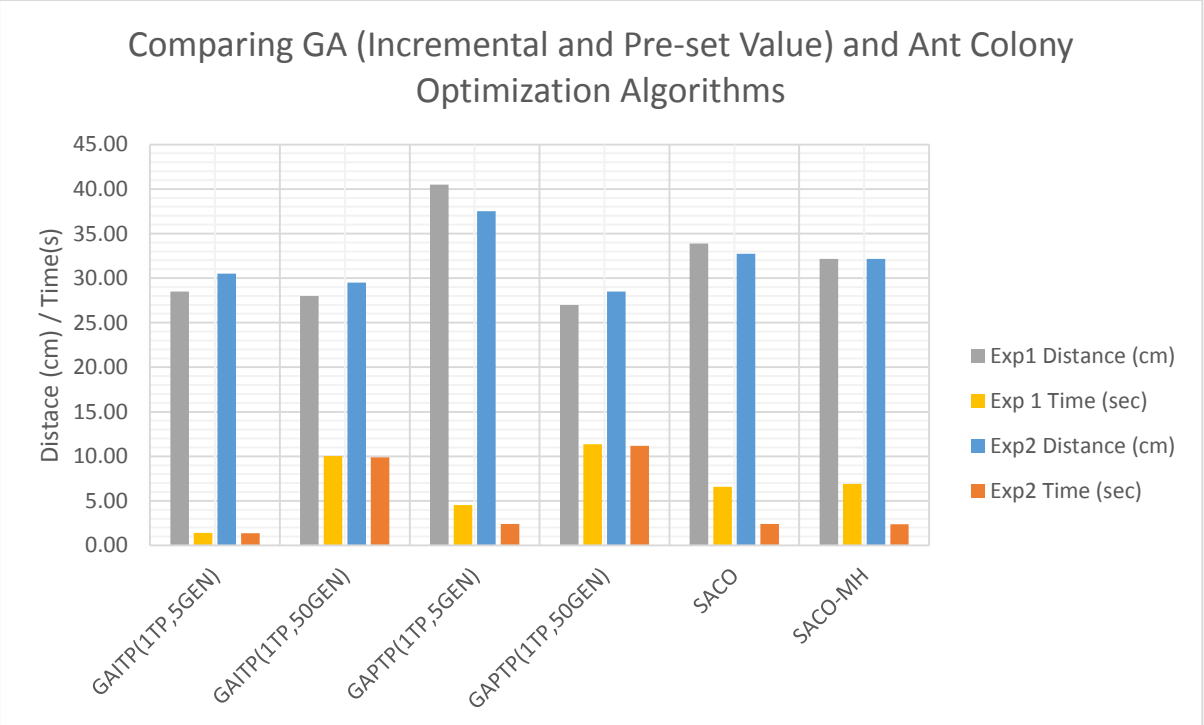


Figure 10: *A bar graph showing the visual representation of the compared data from both own algorithms and the algorithms from the ACO [7].*

The most successful combination was having a low generation setting with a low turning point. That combination yielded paths with relatively low length. In addition, the time cost for that combination was exceptionally lower than the other settings, even when compared to the Ant Colony Optimization (ACO) Algorithms. The reason why this particular algorithm performed exceptionally better was because the map is simple enough to allow the GA to outperform the ACO. In the case that the map is more complicated, the ACO might excel over the GA. The simple map was used because the ACO [7] used it.

5.6 MULTIPLE ROBOT IMPLEMENTATION

As for the multiple robot experiment, the GA proved as effective with multiple robots in producing paths which are static obstacle free. Figure 30 and Table 8 show the mapped paths and the mapped path coordinate data respectively. The GA is shown to remain effective when handling 3 robots in a parallel structure. Corresponding data is shown in table 8.

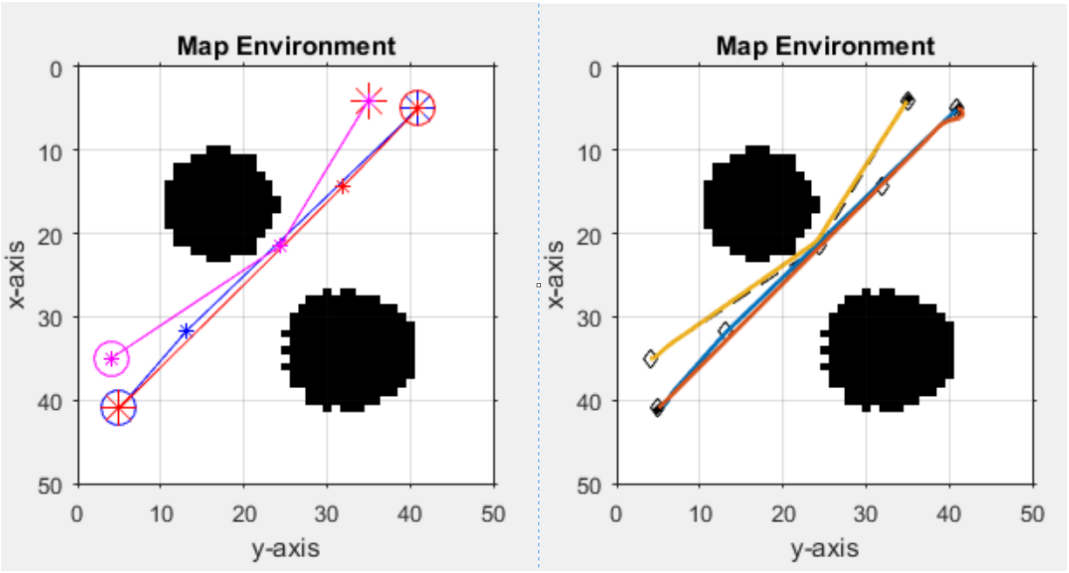


Figure 11: *Outputted Mapped Path for Test Scenario: Static Map 1, 3 Robots. Robot Controller paths shown on the right figure*

Table 8: *Outputted Path Details for Test Scenario: Static Map 1, 3 Robots.*

	Robot 1		Robot 2		Robot 3	
Path Coordinates	Y (cm)	X (cm)	Y (cm)	X (cm)	Y (cm)	X (cm)
	41.00	5.00	5.00	41.00	35.00	4.00
	31.61	13.18	14.30	31.98	21.50	24.36
	5.00	41.00	41.00	5.00	4.00	35.00
Mapped Path Length (cm)	50.00		50.00		46.00	
Followed Path Length (cm)	50.90		52.00		44.60	
% Path Deviation	1.80%		4.00%		3.04%	

5.7 POTENTIAL FIELD IMPLEMENTATION

With regards to robot-robot interaction, the potential field feature added was shown to be operating as intended with robots performing avoidance actions when approaching the pre-determined influential radius of one another. The Figure below presents an example of how a robot would adjust its planned path when within the influential range of another robot. The example shows both robots performing the path adjustment.

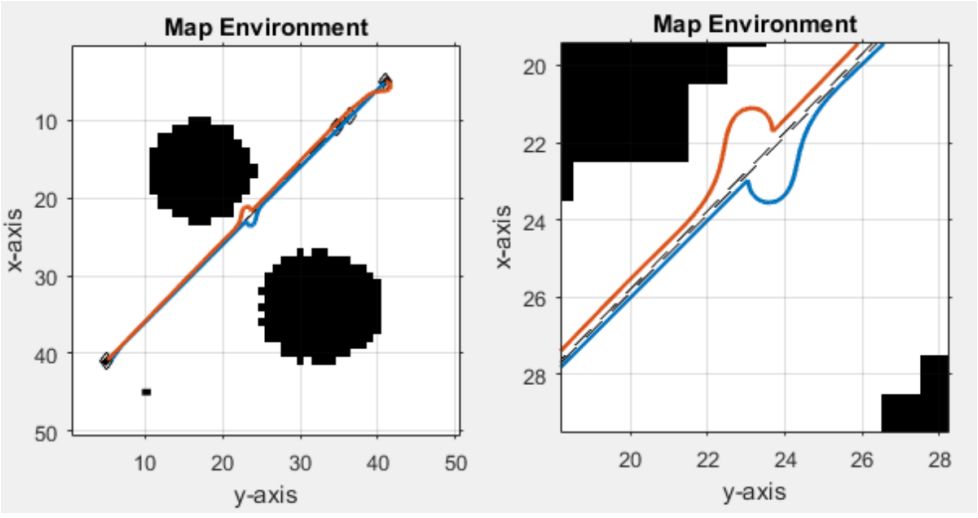


Figure 12: *Shows an example of how a robot would respond to another robot when on course for collision*

Figures 13 and 14 show two separate examples of the robot-robot interaction within two separate test maps. The leader assignment feature is discussed in detail later on within this section.

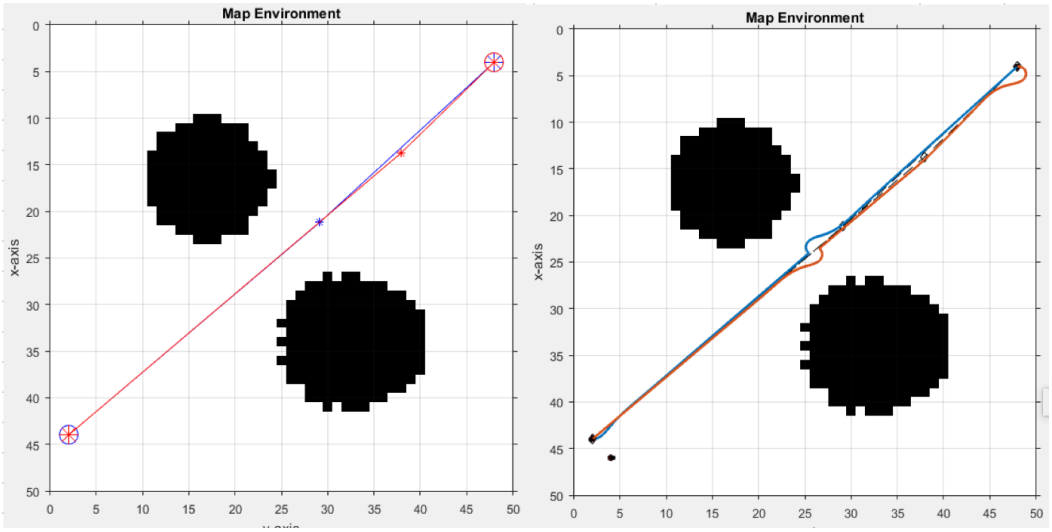


Figure 13: *Outputted Mapped Path for Potential Path Adjustments Experiment: Static Map 1, 2 Robots*

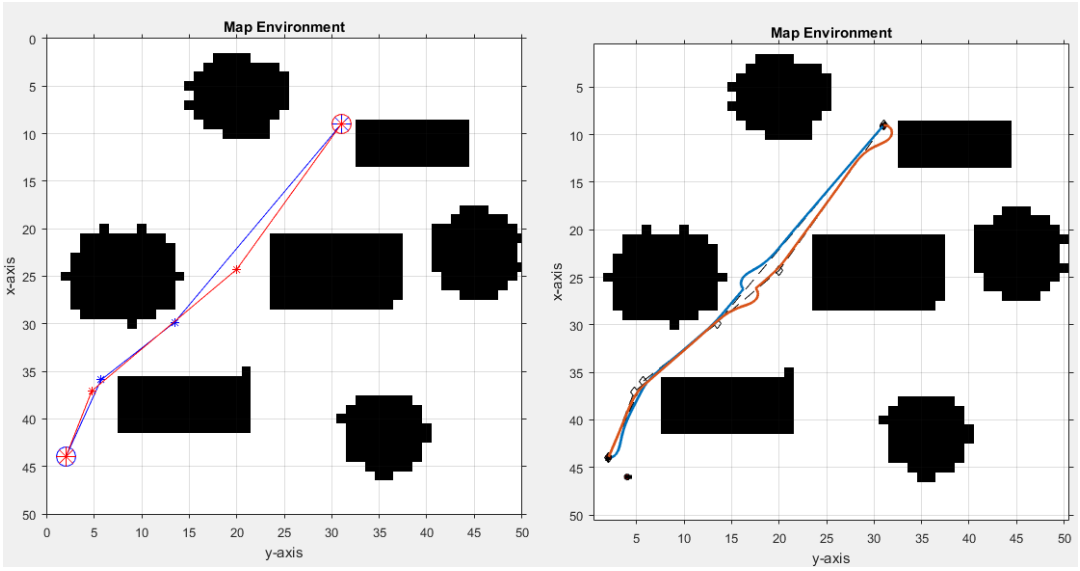


Figure 14: *Outputted Mapped Path for Potential Path Adjustments Experiment: Static Map 2, 2 Robots*

5.8 DYNAMIC OBSTACLE IMPLEMENTATION

After dynamic obstacles were added, the online path adjustment feature was modified in order to account for dynamic obstacles. The Figure below presents an

example of how a robot would deal with a dynamic obstacle, while maintaining its planned path to its goal point.

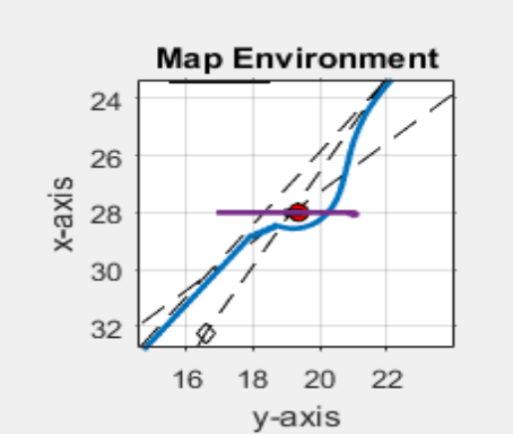


Figure 15: *Shows an example of how robot 1 could be set up to adjust its planned path in response to a dynamic obstacle*

Figure 16 shows the positioning of three dynamic obstacles, along with their mapped paths and robot controller implementation. Table 9 shows the path coordinate data of the dynamic obstacles added.

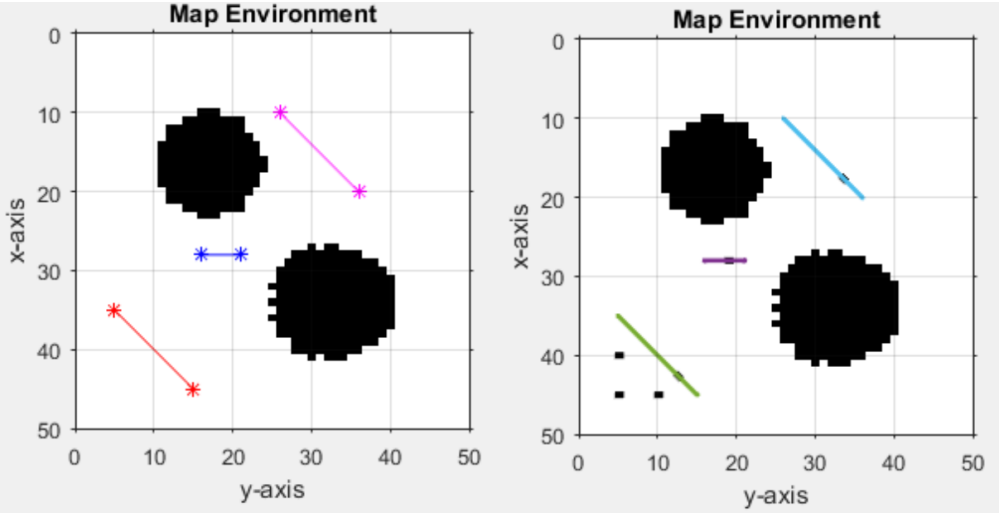


Figure 16: *Path for Dynamic Obstacles. Figure on the left shows the paths planned for the dynamic obstacles. Figure on the right shows the obstacle controller paths.*

Table 9: *Outputted Paths for Each of the Dynamic Obstacles.*

Dynamic Obstacle	1		2		3	
Path Coordinates	Y (cm)	X (cm)	Y (cm)	X (cm)	Y (cm)	X (cm)
	21.00	28.00	5.00	35.00	26.00	10.00
	16.00	28.00	15.00	45.00	36.00	20.00

5.9 DYNAMIC OBSTACLE HANDLING

Next was the experiment designed to test the algorithm's handling of dynamic obstacles. The experiment was set up with 1 robot and 1 dynamic obstacle within its planned path. The situation was run with all the avoiding strategies which included: Stop, Dodge, Reverse. Afterwards, multiple strategies were combined in a chronological order which formed: reverse-dodge and stop-reverse-dodge. Table 10 shows the results from that experiment. The length of each path followed in each strategy was measured. Comparison was made against the path followed with the robot controller if no action was taken. Percentage path deviations were also calculated in each case.

Table 10: *Shows data collected from different dodging strategies*

Dodging Strategy	Path Followed Length (cm)	% Path Deviation	Path Following Time (sec)
No action	21.33	0.00%	26.83
Stop	21.18	0.70%	108.19
Dodge	22.23	4.24%	101.50
Reverse	25.82	21.05%	114.62
Reverse-Dodge	22.68	6.33%	102.67
Stop-Revers-Dodge	22.68	6.33%	118.23

It was found that including the stop function on its own or in a combination of actions added to the time the robot took to complete its course, and did not contribute to minimizing the path followed when combined with other strategies. However, on its own, it had a very low percentage path deviation value of 0.7%. However, the stop strategy on its own was insufficient in dealing with all the testing scenarios attempted, varying robot path and dynamic obstacle path. The strategy most suited for dynamic obstacle handling was the reverse-dodge. This strategy could handle all the testing scenarios attempted. In

addition, it had a medium percentage path deviation value of 6.3% which is still within the acceptable range. It also had a relatively low path following time value.

5.10 EVALUATING NUMBER OF TURNS

The following sections are aimed to evaluate specific features present within the developed algorithm, using the attained results. The first feature to be evaluated is the specification of the number of turns. Due to how the GA was setup, the algorithm managed to find the shortest possible path due to the specification of a low value for the number of turning points. As discussed in chapter 4, the number of turning points represented the number of genotypes which the GA takes into consideration. A lower number of turning points operates well within simple maps because the robot does not need to turn many times. However, within more complicated maps, the GA would be unable to produce a desirable path while being constrained by the maximum number of turning points. As a result, this feature experiment was created. Shown in the following figure are three cases which the robot was given the same map, source and goal points. The only variable was the number of turning points.

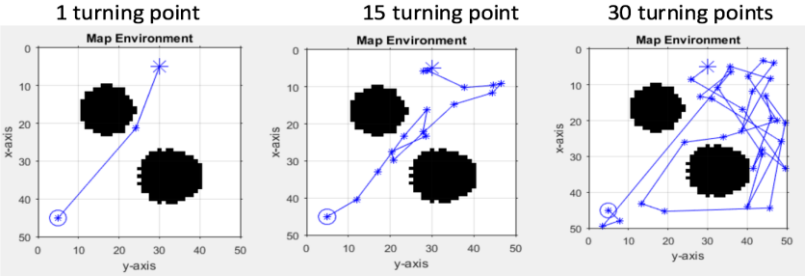


Figure 17: *Three cases shown. Robot 1 is set up a path from point A to B, while varying the number of turning points*

As observed in the previous figure, having a low number of turning points benefitted the algorithm in finding a simple path, rather than the second case, where the algorithm was forced to use the high number of turning points specified. Doing so caused

the GA to produce a path with repeated turns which were unneeded because the shortest path is a desirable criteria of evaluation. If the map was a more complicated one, the low number of turning point will have the algorithm stuck in an infinite loop, as the algorithm fails to find a path with the constraints it has been given. In order to fix this, the algorithm is given a minimum value for the number of turns it initially uses. In the case that it finds a path with the current parameters, it will proceed to pass it on to the robot controller. In the case that that GA was unsuccessful in finding a path, the algorithm will increment the number of turning points and restart the algorithm. Figure 36 in Appendix A shows a flowchart visually representing this process.

Two criteria or evaluation were measured for the performance of the GA in each case. The criteria were computational time and path length. It was shown that having an incrementing number of turning points allowed the algorithm to attain paths which were relatively minimal in their length. Tables 18 and 19 show the data attained from these experiments using two separate test maps.

Table 11: *Outputted Paths for Turning Point Experiment, 3 Robots, Static Map 1*

Map	Static Map 1								
Generations	10				50				
Setup of Turning Points	TP	Time Cost (sec)	Path Length (cm)	Figure	Setup of Turning Points	TP	Time Cost (sec)	Path Length (cm)	Figure
Incremental	1	3.73	47	18	Incremental	1	14.23	47	18
Total	1	3.73	47		Total	1	14.23	47	
Pre-Set Minimum	2	4.41	47	19	Pre-Set Minimum	2	14.57	46.5	19
Total	2	4.41	47		Total	2	14.57	46.5	
Pre-Set Maximum	5	7.37	71.5	20	Pre-Set Maximum	5	20.04	48	20
Total	5	7.37	71.5		Total	5	20.04	48	

Table 12: *Outputted Path for Turning Point Experiment, 3 Robots, Static Map 2*

Map	Static Map 2								
Generations	10				50				
Setup of Turning Points	TP	Time Cost (sec)	Path Length (cm)	Figure	Setup of Turning Points	TP	Time Cost (sec)	Path Length (cm)	Figure
Incremental	1	3.46	0	Figure 21	Incremental	1	14.05	0	Figure 21
	2	4.01	50			2	15.18	47.5	
Total	2	7.48	50		Total	2	29.23	47.5	
Pre-Set Minimum	2	4.56	52	Figure 22	Pre-Set Minimum	2	17.35	56.5	Figure 22
Total	2	4.56	52		Total	2	17.35	56.5	
Pre-Set Maximum	5	6.74	58	Figure 23	Pre-Set Maximum	5	22.79	59	Figure 23
Total	5	6.74	58		Total	5	22.79	59	

The corresponding mapped paths are provided in Figures 18, 19, 20, 21, 22 and 23.

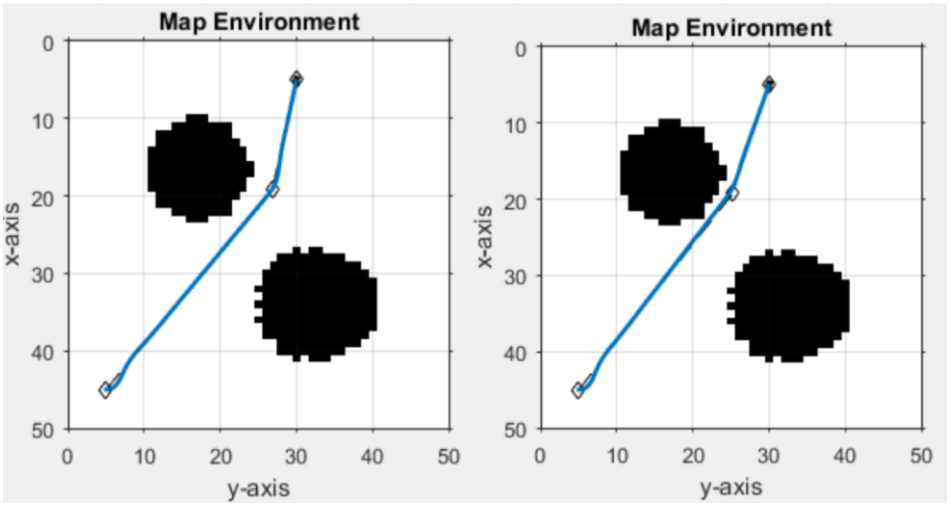


Figure 18: *Outputted Mapped Path for the turning Point Experiment. Incremental Turning Points. Map Reference Corresponds to data in table 11. 10 Generations(left) and 50 Generations (right). Static Map 1.*

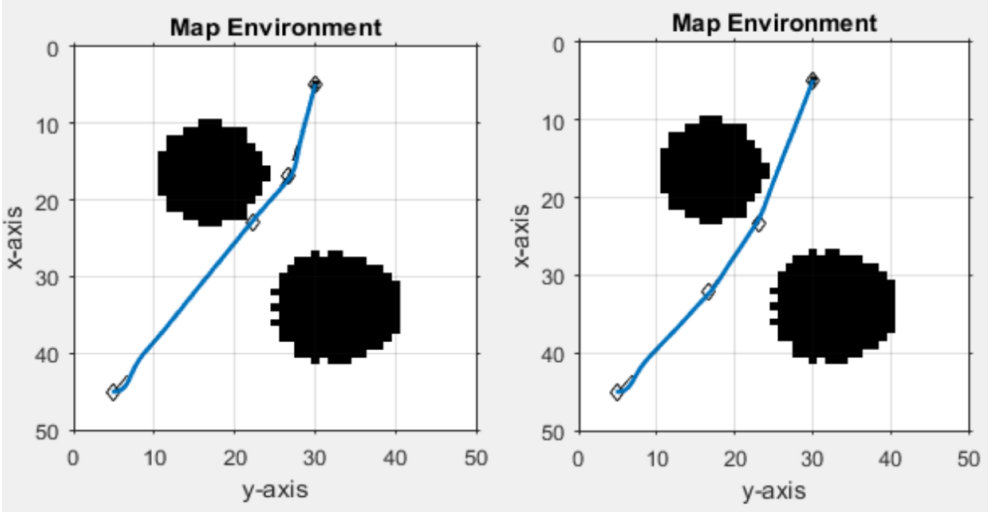


Figure 19: *Outputted Mapped Path for the turning Point Experiment. Pre-set minimum. Map Reference Corresponds to data in table 11. 10 Generations(left) and 50 Generations (right). Static Map 1.*

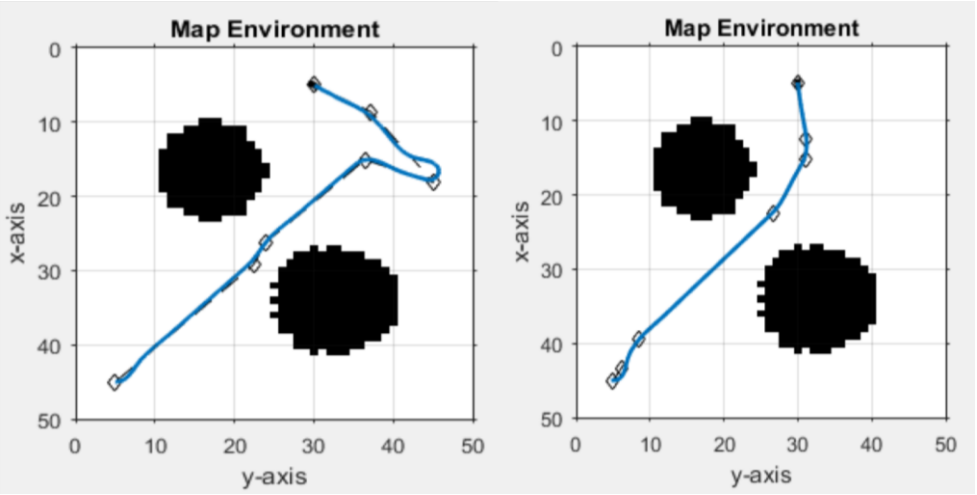


Figure 20: *Outputted Mapped Path for the turning Point Experiment. Pre-set Maximum Turning Points. Map Reference Corresponds to data in table 11. 10 GEN(left) and 50 GEN (right). Static Map 1.*

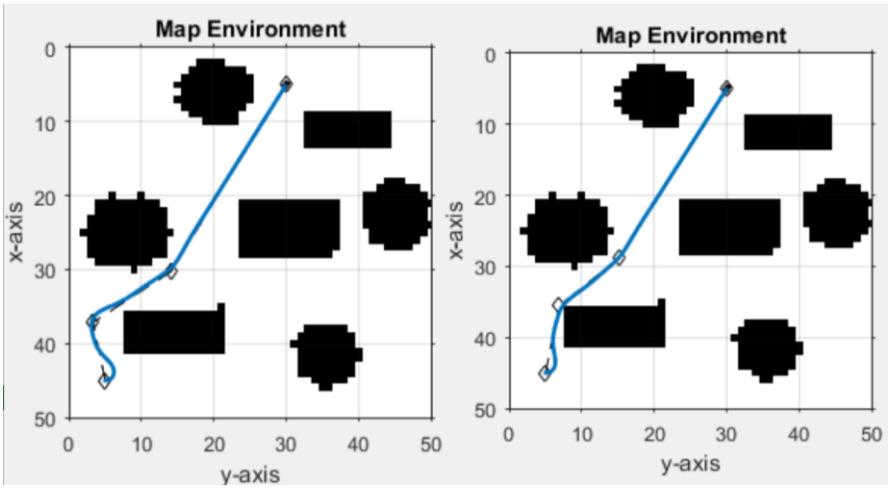


Figure 21: *Outputted Mapped Path for TP Experiment. Incremental Turning Points. Map Reference Corresponds to data in table 12. 10 GEN(left) and 50 GEN (right). Static Map 2.*

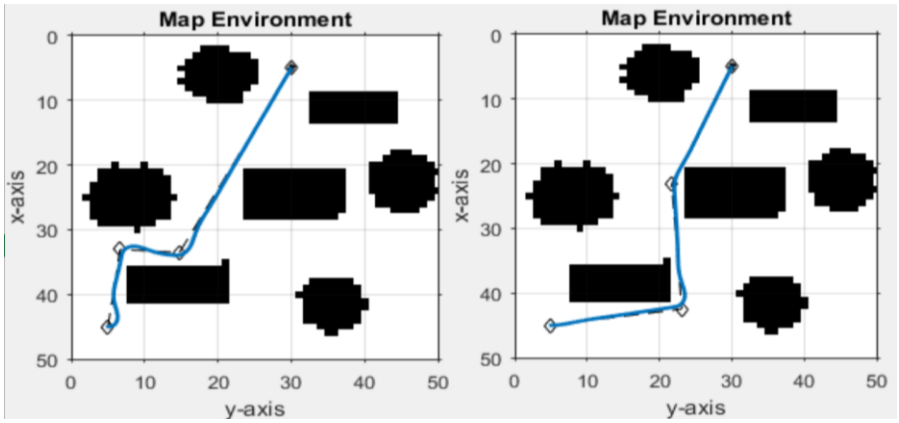


Figure 22: *Outputted Mapped Path for TP Experiment. Pre-set minimum Turning Points. Map Reference Corresponds to data in table 12. 10 GEN(left) and 50 GEN (right). Static Map 2.*

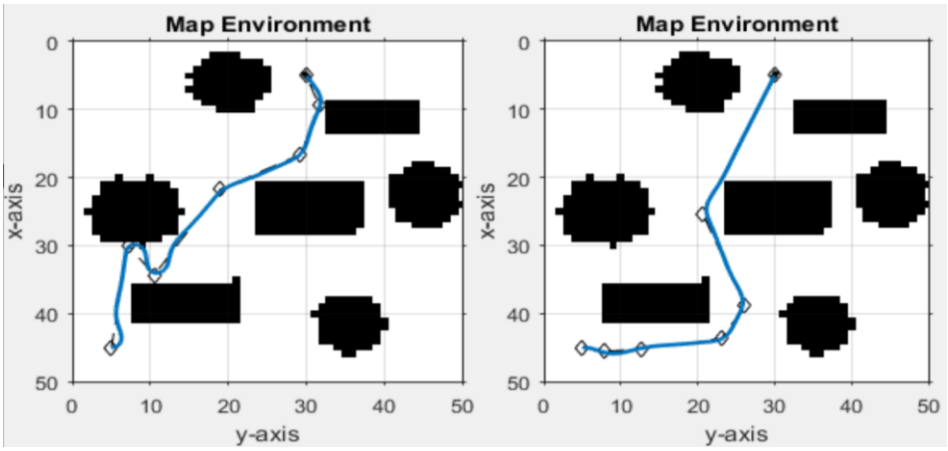


Figure 23: *Outputted Mapped Path for TP Experiment. Pre-set maximum Turning Points. Map Reference Corresponds to data in table 12. 10 GEN(left) and 50 GEN (right). Static Map 2.*

For the higher pre-set maximum number of turns, a short path was attained, but with a relatively high time cost. In the cases of a pre-set minimum number of turning points, an ideal path length was attained, with an acceptable time cost. The problem is that the pre-set minimum has to be set in accordance to each map, source and goal point. From the data, it can be deduced that having an incremental number of turning points, allows for the algorithm to be flexible in multiple maps. Since the incremental setup starts at low value, the time cost for the initial failed runs is not a significant time loss.

5.11 POTENTIAL FIELD ADJUSTOR TEST

The second feature to be evaluated was the potential field algorithm. In order to be most effective, the algorithm needed to work on a case by case basis. If and only if 2 robots come within close proximity of one another, the potential field script would be activated to cause both robots to dodge one another. Once both robots have exceeded each other's influence radius, then the robots would proceed to return on their planned path. In order to test this out, an experiment with 2 robots was setup. In order to bias the robots to be on course to collide with one another, the source of the first robot was the goal for the second robot and vice versa. This causes the algorithm to produce two paths which were very similar to one another, thus causing the robots to be on course for a collision. Figure 13 shows the data from that particular experiment. The data collected included the path planned by the GA as well as the actual path followed by the robot controllers. The robot controllers use the GA planned path as a reference, but in the case of an expected collision, the potential field algorithm would cause certain deviations in order to avoid a collision. The difference between both paths was calculated in order to attain the percentage deviation performed by the potential field script

From Observing the data in figure 13, it can be seen that the potential field script is operating effectively. The robots follow their intended path until one of the potential fields force them otherwise. The dodging is performed to a degree that the robots avoid one another from a safe distance, but return to their planned paths afterwards. The mapped path coordinate data is presented in tables 13 and 14.

Table 13: *Outputted Path for Potential Path Adjustments Experiment: Static Map 1, 2 Robots*

	Robot 1	Robot 2
Path Planned Length (cm)	60.00	60.00
Path Followed Length (cm)	61.70	63.50
Path Length Difference (cm)	1.70	3.50
Path Deviation %Difference	2.83%	5.83%

Table 14: *Outputted Path for Potential Path Adjustments Experiment: Static Map 2, 2 Robots*

	Robot 1	Robot 2
Path Planned Length (cm)	45.00	45.00
Path Followed Length (cm)	46.40	48.00
Path Length Difference (cm)	1.40	3.00
Path Deviation %Difference	3.10%	6.66%

The static potential field exerted by the static environment help the robots' avoidance be biased towards more open spaces, to a certain degree. The paths followed were all shown to have less than 7% path deviation.

It should be noted that some of the deviation is caused by the robot controller's settings. All the robots are set up with an initial angular orientation of 0. Depending on its path, the robot will turn and adjust its orientation accordingly while still having its motors running. This causes some deviation to the path. Another cause of path deviation is that the pure pursuit controller contains a characteristic called look ahead distance. This characteristic, allows the robot to remain deviated from its path, if it can see that

eventually, it will return to the correct path. This characteristic was kept in order to provide further incentive for the robots to return to their original planned path.

5.12 LEADER ASSIGNMENT TEST

Next, the Leader Assignment feature was to be evaluated. The leader assignment feature was designed in order to assign certain robots as leaders to other robots. The leaders would not have to dodge, and would continue on their path normally. The follower robot respective to the leader assigned would proceed to dodge around the leader. This was aimed to decrease the total path deviations occurring when dealing with multiple robots.

In order to do this, an experiment was setup with 2 and 3 robots within a specific map. A collision was encouraged by assigning the robots respective source and goal points which were near one another. Multiple Adjustments were done to the algorithm for testing purposes. The testing scenario was run with no leader assignment (NLA), simple ranked system (SRS), repeated random leader assignment (RRLA). After those scenarios were run, the testing environment was changed to include 3 robots. With 3 robots in place, a testing case was done for Closest to Goal (CTG) and Farthest to Goal (FTG). A simple description is given for each testing setting.

NLA: No Leader is assignment. All robots dodge one another as equals. Data presented in figure 24 and table 15.

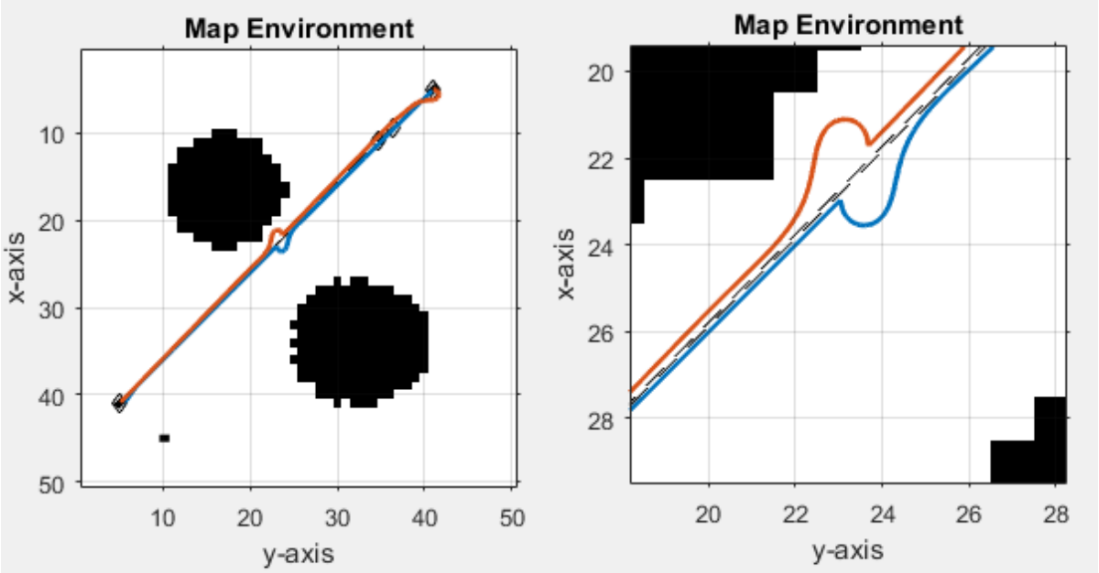


Figure 24: *Mapped Path for Leader Assignment Experiment- NLA Settings. Figure on the right is a magnified version*

Table 15: *Outputted Path for Leader Assignment Experiment- NLA Settings*

	Robot 1	Robot 2	Total
Total Mapped Path Length (cm)	50	50	100
Total Followed Path Length (cm)	52.06	55.32	107.38
% Path Deviation	4.12%	10.65%	7.38%

SRS: A ranking system is provided to the robot's pre-simulation. The ranking system is only assigned once. Ranking System is assigned based on the robot number. Data presented in figure 25 and table 16.

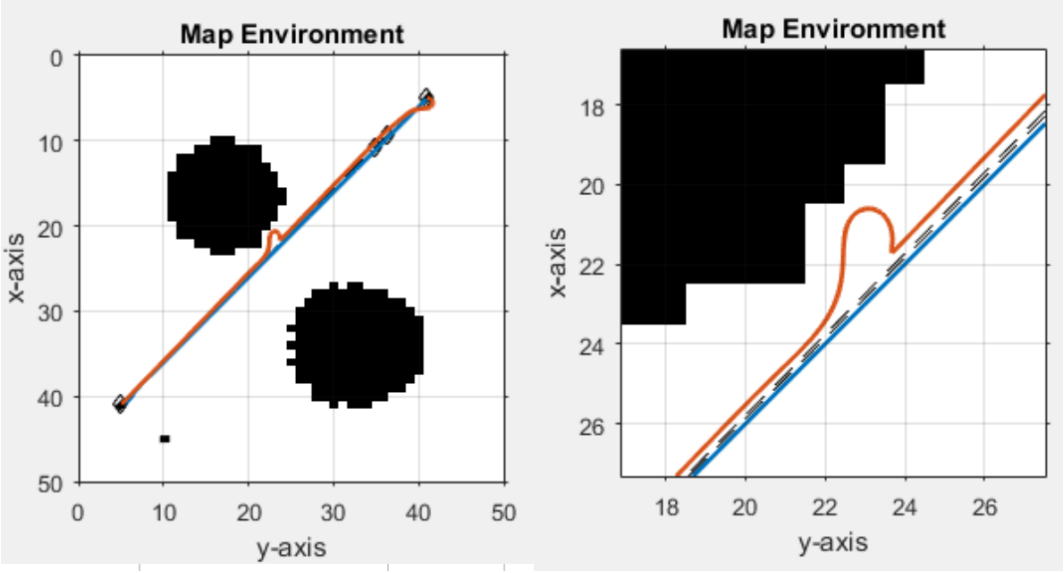


Figure 25: *Mapped Path for Leader Assignment Experiment- SRS Settings. Figure on the right is a magnified version*

Table 16: *Outputted Path for Leader Assignment Experiment- SRS Settings*

	Robot 1	Robot 2	Total
Total Mapped Path Length (cm)	50	50	100
Total Followed Path Length (cm)	50.90	54.50	105.40
% Path Deviation	1.80%	9.00%	5.40%

RRLA: When the robots are on course for collision, a ranking is given to the robots randomly. The ranking is re-assigned every time a collision is about to occur. Data presented in figure 26 and table 17.

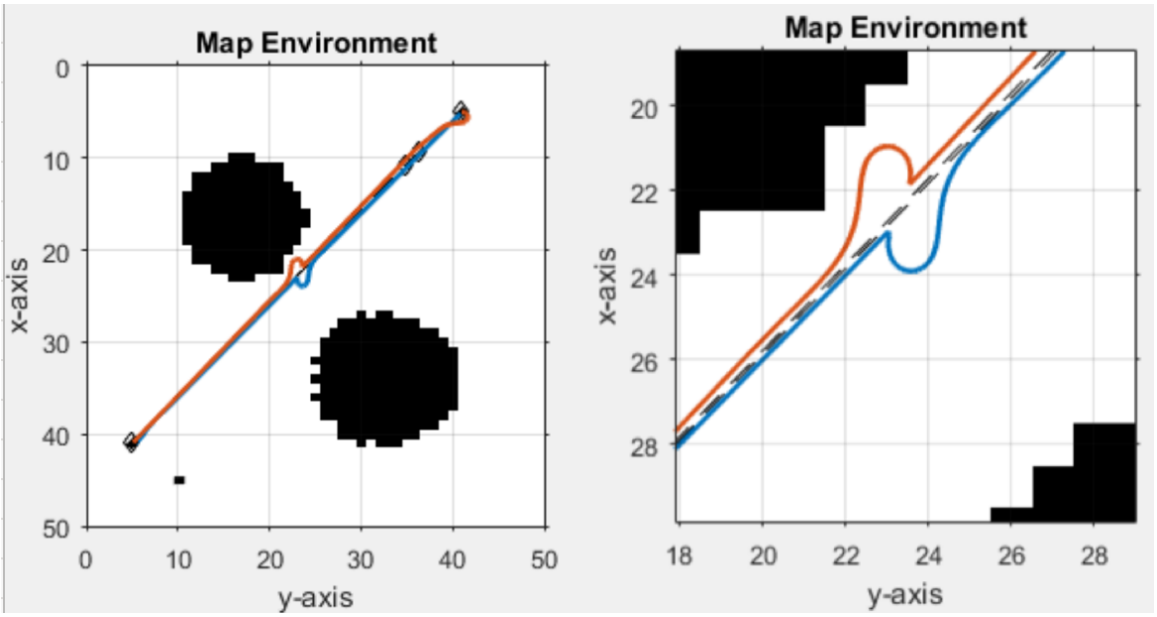


Figure 26: *Mapped Path for Leader Assignment Experiment- RRLA Settings. Figure on the right is a magnified version*

Table 17: *Outputted Path for Leader Assignment Experiment- RRLA Settings*

	Robot 1	Robot 2	Total
Total Mapped Path Length (cm)	50	50	100
Total Followed Path Length (cm)	52.79	54.73	107.52
% Path Deviation	5.58%	9.47%	7.52%

CTG: When the robots are on course for collision, a ranking is given to the robots based on their distance to their goals respectively. The closer the robot is to its goal, the higher its ranking. Data presented in figure 27 and table 18.

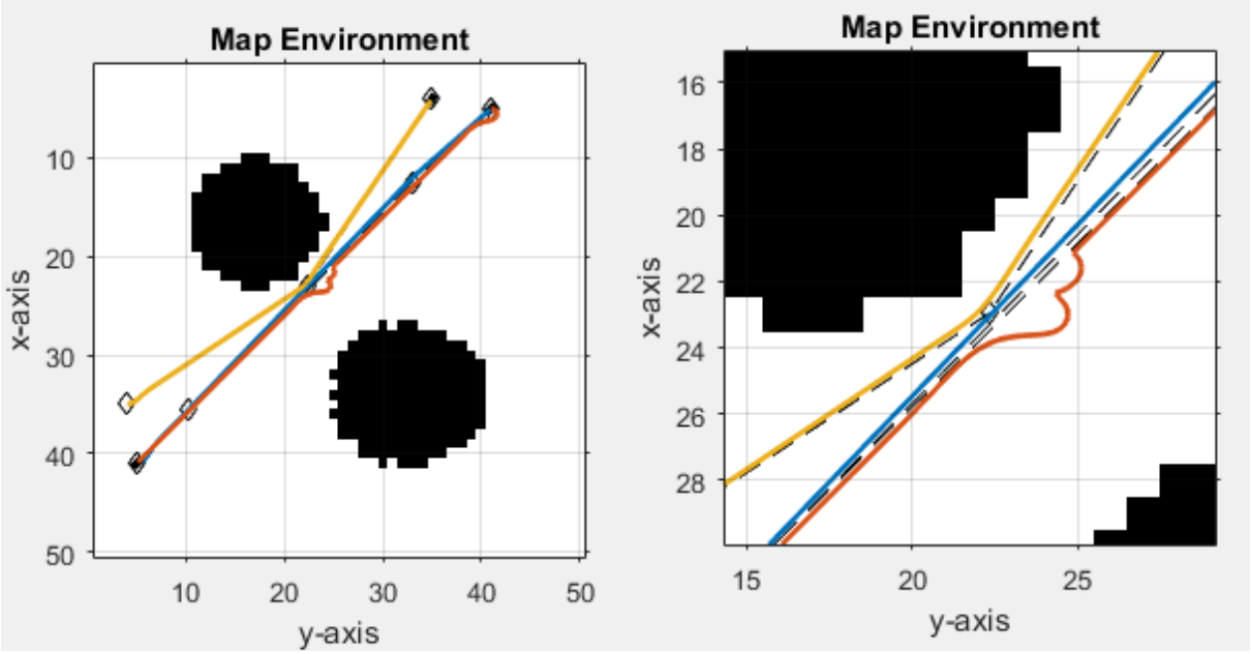


Figure 27: *Mapped Path for Leader Assignment Experiment- CTG Settings. Figure on the right is a magnified version*

Table 18: *Outputted Path for Leader Assignment Experiment- CTG Settings*

	Robot 1	Robot 2	Robot 3	Total
Total Mapped Path Length (cm)	50	50	44	144
Total Followed Path Length (cm)	50.90	53.40	44.50	148.80
% Path Deviation	1.80%	6.80%	1.14%	3.33%

FTG: When the robots are on course for collision, a ranking is given to the robots based on their distance to their goals respectively. The farther the robot is to its goal, the higher its ranking. Data presented in figure 28 and table 19.

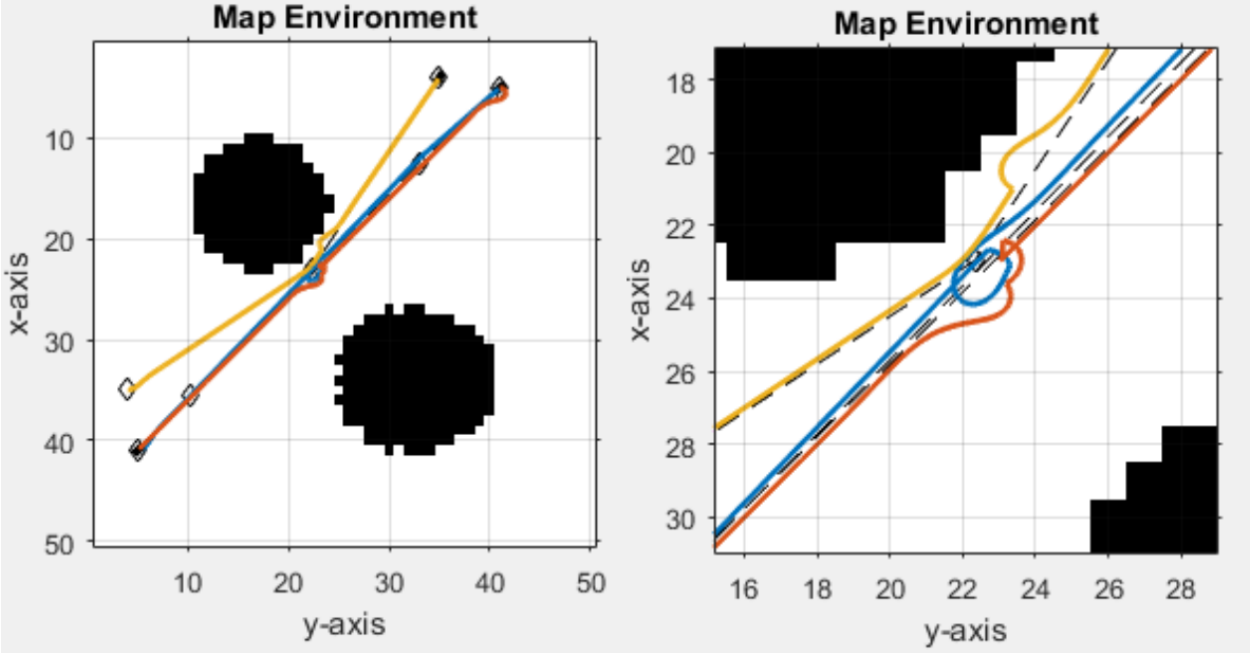


Figure 28: *Mapped Path for Leader Assignment Experiment- FTG Settings. Figure on the right is a magnified version*

Table 19: *Outputted Path for Leader Assignment Experiment- FTG Settings*

	Robot 1	Robot 2	Robot 3	Total
Total Mapped Path Length (cm)	50	50	44	144
Total Followed Path Length (cm)	55.43	54.40	45.10	154.93
% Path Deviation	10.86%	8.80%	2.50%	7.59%

For the first 3 test cases, the leader assignment did show improvement of the overall path deviation. With the NLA, the robots had a percentage path deviation of 4.12% and 10.65% respectively. With SRS in place, only one robot had to perform the dodging action, so the robots had a percentage path deviation of 1.80% and 9.00% respectively. However, for the RRLA case, the robots were randomly selected as the leader and follower. That random selection process was called upon twice and the roles

were switched. As a result, both robots deviated from their paths. For that particular test case, the robots had a percentage path deviation of 5.58% and 9.47% respectively. It was clear that assigning one of the robots as a leader and the other as its follower, minimized the percentage deviation of each robot. The RRLA case had increased deviation because the random leader/follower selector ran multiple times when both robots came within close proximity of one another.

For the last two test cases with the 3 robots in place, a different result was attained. The CTG case had the robots with % path deviations of 1.80%, 6.80%, and 1.14%. This showed that at the time the robots came within close proximity with one another, the robot farthest to its respective goal, was assigned as a follower. As a result, the % path deviation was focused into one path rather than be dispersed amongst all 3 paths. For the FTG case, the opposite logic was used. At the robots came within close proximity with one another, the robot farthest to its respective goal, was assigned as a leader. This was aimed to encourage the algorithm to have all the robots complete their paths at the same time. However, the robots had a percentage path deviation of 10.86%, 8.80% and 2.50% respectively. The FTG leader assignment caused the robots which were in the lead and closer to their goal to move out of the way for the robot farthest to its goal, their leader. This caused needless path deviations and increased the total path deviation significantly for 2 out of the 3 robots. From observing the collective data from the leader assignment experiment, it can be seen that the leader assignment can be a feature which reduces the path deviations. However, that depends on what criteria the leader and follower are chosen. If the wrong leader assignment strategy is chosen, the

path deviations could be increased needlessly as a result. Figure 29 shows the mapped paths for the NLA leader assignment settings for the 3 robot implementation.

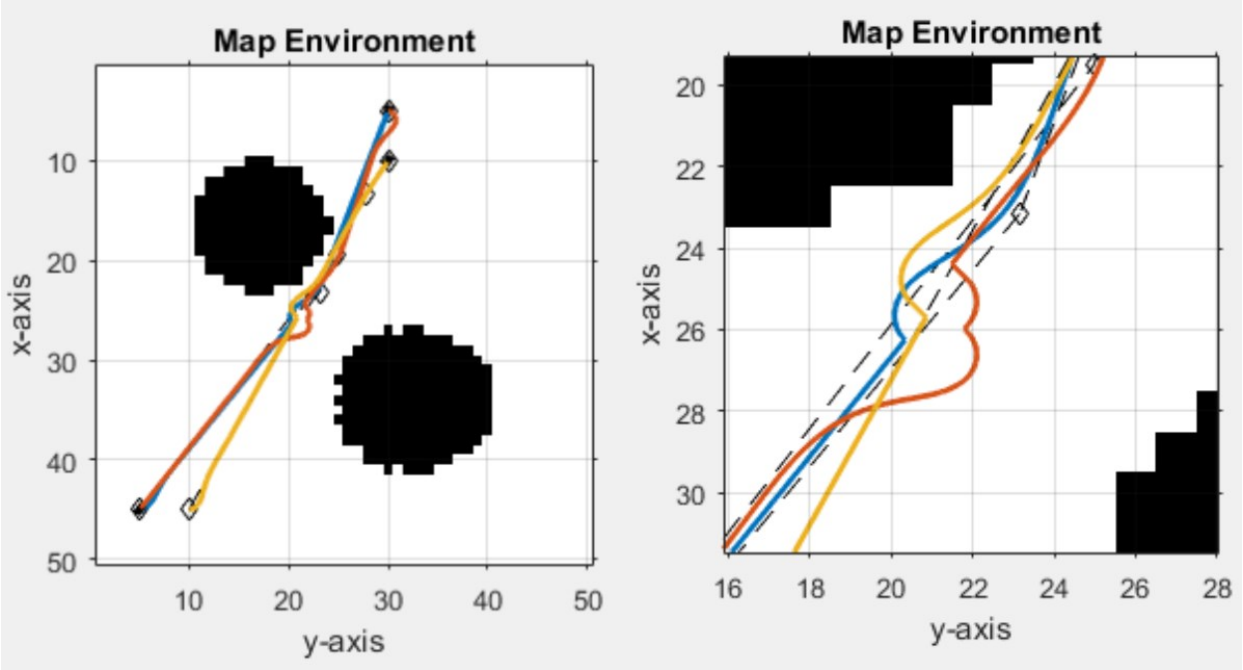


Figure 29: *Mapped Path for NLA settings with 3 Robots in play*

It can be seen that with 3 robots implemented, the path deviations become more and more significant. This would be an important feature to consider if the number of robots is to be scaled even further.

5.13 FULL ALGORITHM IMPLEMENTATION

The final experiment was the implementation of the full algorithm. This experiment was designed with 3 robots within a map which included 2 static obstacles and 3 dynamic obstacles. Figure 30 shows the GA-planned paths for each robot. Figure 31 show the robot controller implementation. Figure 37 is the flowchart showing the flow of the code scripts provided in Appendix D.

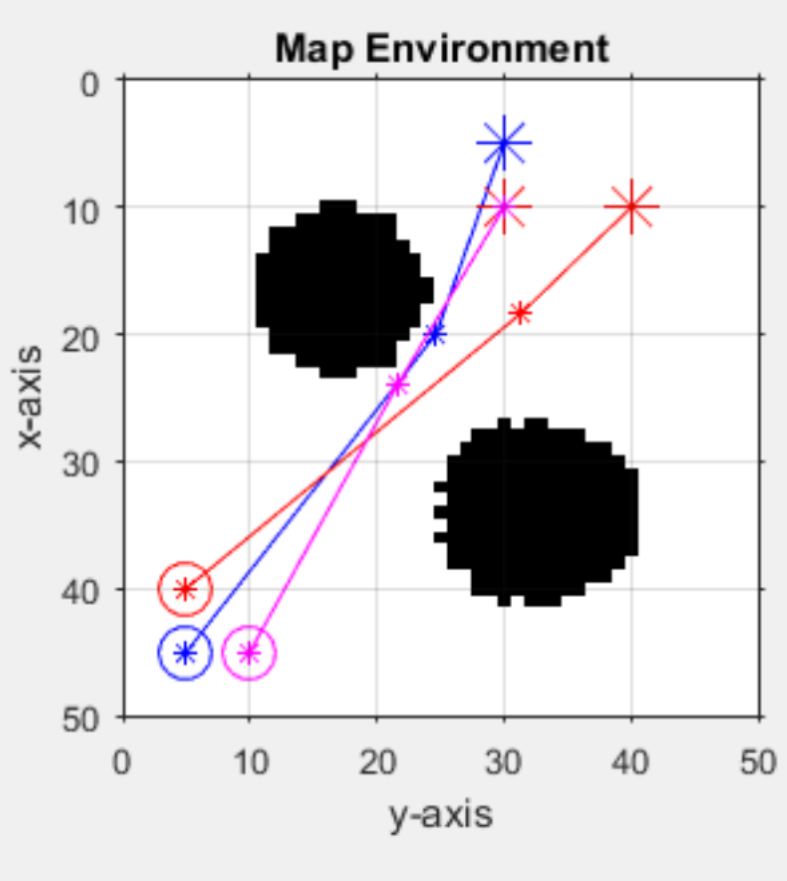


Figure 30: *Figure shows the planned path output for 3 robots. The respective robot controllers are shown in Figure 31.*

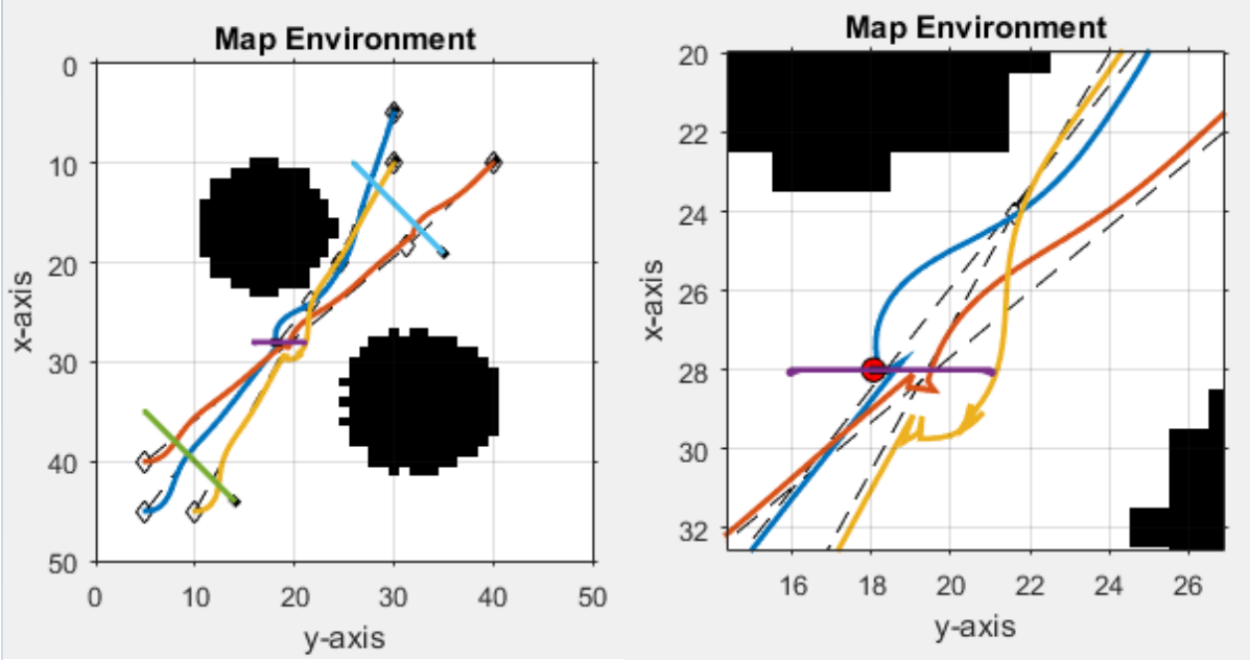


Figure 31: *path data with the full algorithm implementation. 3 Robots were implemented. 3 Dynamic obstacles were present. Figure on the right shows a magnified version of clustered area*

The magnified map shown in figure 31 shows the online path deviations which the robots did in response to one another as well as in response to the dynamic obstacle.

Table 20: *Shows data collected from 3 Robot Experiment shown visually in figure 31.*

Path Type/Robot #	Robot 1	Robot 2	Robot 3
Path Planned Length (cm)	47.00	45.50	39.50
Path Followed Length (No Action)(cm)	47.93	46.13	40.95
Path Followed Length(cm)	50.11	52.65	51.45
% Path Deviation	4.56%	14.15%	25.64%

Corresponding path data is shown in table 20. The full algorithm is effective in handling the test scenarios attempted. The leader assignment portion of the code is also effective. This is shown in the robots having varying % path deviation. Robot 3 had the highest % path deviation. This is due to it being the farthest to its goal when the evaluation was made. The path length followed was compared to the path length attained with no action taken.

CHAPTER 6 CONCLUSION AND FUTURE WORKS

6.1 CONCLUSION

While other algorithms use the GA as the main problem solving tool to generate the obstacle-free paths, the iterative process makes it difficult to apply for dynamic obstacles handling. The algorithm in this thesis presents a two-part algorithm. The first part is an offline GA global path planner which deals with the static environment at hand. The second part is an online path adjustor which is based on the potential field method. The algorithm is shown to be effective with handling both static and dynamic obstacles. The algorithm was applied for both single and multiple robots. Different dodging strategies had to be assigned for the robot-robot potential collisions and the robot-dynamic obstacle potential collision.

This specific algorithm uses the GA in order to generate the general path which the robot controller uses as a base input. The robot controller could adjust the path should a collision is predicted. Possible collisions could include other robots following their own respective paths or even dynamic obstacles which will not dodge any incoming object.

Different features were added in order to fix situational complications. One of those complications was the equal dodging of robots. More dodging resulted in increased path deviations which in some scenarios complicated the situation even further than before the dodge. In order to fix and mitigate this, the leadership assignment feature was implemented. This feature was inspired from the “right of way” rules present within real-life traffic cases. Different methods were tested in order to decide on who will be the leader and who will be the follower. Through testing, it was decided that priority ranking

will be given based on which robot is closer to its respective goal. This made the algorithm prioritize robots reaching their respective goals faster. This feature also handled the dodging of dynamic obstacles. Since dynamic obstacles do not dodge any incoming objects no matter what, they are given the highest of all rankings by default.

Another feature which was added was the incrementing turning points feature. Due to the nature of how the GA was set up, assigning more turning points resulted in more complicated paths and increased processing time. In order to mitigate this, a base minimum for the turning points was assigned for the GA to compute a path with. If the GA is unsuccessful, the number of turning points is incremented and the process is repeated. This allowed the GA to be more flexible with different types of maps; finding simple paths where needed and more complicated ones for the maps which are more complex in their nature.

The GA global path planner was compared to two different algorithms based on the ant colony optimization method. Due to the simple nature of the maps used within [7], the low generation GA proves to be more superior in terms of path length and computational cost. However, that may not be always the case for all maps. Additional testing may be required on more complicated maps to truly understand the advantages and disadvantages of using each algorithm.

Another possible limitation present within the algorithm is the simplicity of the potential field path adjustor. The adjustor takes action once the robot has a foreign object present within a specific radius of influence around the robot. It does not take into consideration the properties of the foreign object. These properties may include but are not limited to: size, speed, acceleration, path and orientation. One possible improvement

would be to have the robots aware of each other's properties as they are following their own respective paths. This could allow for smarter dodging actions to be taken.

Another limitation noticed within the algorithm is related to the potential field forces applied on the robot. If the forces exerted from the dynamic obstacles, static obstacles and other robots cancel out, the robot could potentially have a no solution in terms of how to act. Additional testing may be required in order to make the algorithm more flexible within smaller maps, thus allowing for a more condensed collision space.

One relevant point to note is the time cost evaluation criteria. While the computational time cost of the GA path planner is calculated and used as an evaluation criterion, that is not the time for the full algorithm. Since the algorithm consists of two parts: an offline and an online portion, the time cost needs to include that as well. Additional testing could be done in order to account for both the computational time cost and the time it takes for the robot controller to complete its path. However, that would cause the time cost to scale more significantly with scaling the map sizes.

As for the comparison with the ACO, the GA proved to be more effective. However, it is important to note a few points about that comparison. The map used within [7] was too simple. Because of that, the low generation GA was more effective in finding a path which is optimal in distance and with a relatively low time cost as well. It is important to note that if the comparison was made with more complicated maps, then the GA might not prove to be as effective with its current settings. Another area where additional testing in future works could be done is to compare the GA's performance with multiple path planning algorithm, including different maps and scenarios. Analyzing

what makes an algorithm most effective in a certain situation, could allow for the creation and modification of a hybrid algorithm which is most flexible in all situations.

In conclusion, the algorithm presented within this thesis is designed from the ground up, managed to solve the global path planning problem presented. There exists room for improvement and additional features to be added in order to make the algorithm more adaptable and applicable to different scenarios, maps and complexities.

6.2 FUTURE WORKS

There are areas of future works allow for more improvements to the algorithm. These include: more scenario testing for the algorithm, additional modifications to the potential field path adjustor. More testing could be done to compare the algorithm to other path planning algorithms which may or may not be based on GA, in order to explore additional features to be included. In addition, the implementation of the algorithm in a real life test scenario is another area for future work. Furthermore, more objects could be added to the test environment to test how the algorithm would deal with a higher cluster of objects. A more advanced queuing system could be established where the robots communicate with one another and assign a rank on a criteria different than which is closer to the respective goal point. Larger dynamic obstacles could be tested on the algorithm, adjusting it accordingly. Furthermore, a variable path length could be implemented into the algorithm. This approach would evaluate the map at the global path planner stage, assigning it a score. That score would be used to identify how complex does the GA have to be. This will have to be tested against the turning point data attained in this thesis in order to prove the effectiveness of such a feature. Finally, an experiment could be designed in order to see how the algorithm deals with a “no-solution” situation.

Where there is no possible path to make for the robot, how will the algorithm deal with such a situation. Similar situations could be crafted in order to test the flexibility of the algorithm in dealing with all possible cases. This would further help in developing this algorithm to be one that is realistic and more applicable to the real world.

6.3 CONTRIBUTION

The contribution of this thesis is revisited within this section. Genetic algorithm is a heavily researched field. As a result, adding contributions to the field could prove difficult. Rather, the main contribution lies within the features implemented within the algorithm. These features contribute mainly to the field of path planning. These include the turning point experiment pertaining to the genetic algorithm portion, as well as the leader assignment feature added to the potential field path adjustor. Adding features to the algorithm, not only makes it unique and more differentiable than other algorithms but also allows the experimentation on only certain aspects of the algorithm. Testing on these aspects allows for a more directed testing approach, and allows the addition of additional contributions even to heavily researched fields such as genetic algorithm use in path planning.

BIBLIOGRAPHY

[1] Ahmed, A., Abdalla, T. Y., & Abed, A. A. (2015). Path Planning of Mobile Robot by using Modified Optimized Potential Field Method. International Journal of Computer Applications,113(4), 6-10. doi:10.5120/19812-1614

[2] Distante, C., Indiveri, G., & Reina, G. (2009). An application of mobile robotics for olfactory monitoring of hazardous industrial sites. Industrial Robot: An International Journal, 36(1), 51-59. doi:10.1108/01439910910924675

[3] Jeon, S., Jeong, W., & Park, D. (2014). A Stable Tracking Control of Skid Steered Mobile Platform. Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics. doi:10.5220/0005113305560561

[4] Kettlewell, H. B. (1955). Selection experiments on industrial melanism in the Lepidoptera. Heredity,9(3), 323-342. doi:10.1038/hdy.1955.36

[5] Khatib, O. (1986). Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. Autonomous Robot Vehicles, 396-404. doi:10.1007/978-1-4613-8997-2_29

[6] Large, F., Laugier, C., & Shiller, Z. (2005). Navigation Among Moving Obstacles Using the NLVO: Principles and Applications to Intelligent Vehicles. Autonomous Robots, 19(2), 159-171. doi:10.1007/s10514-005-0610-8

[7] Mohanraj, T. (2014). Mobile Robot Path Planning Using Ant Colony Optimization. International Journal of Research in Engineering and Technology, 03(23), 1 6. doi:10.15623/ijret.2014.0323001

[8] Raja, P. (2012). Optimal path planning of mobile robots: A review. International Journal of the Physical Sciences, 7(9). doi:10.5897/ijps11.1745

[9] Wu, Z., Fu, W., Xue, R., & Wang, W. (2016). A Novel Global Path PlanningMethod for Mobile Robots Based on Teaching-Learning-Based Optimization. Information, 7(3), 39. doi:10.3390/info7030039

[10] Yue, G., Xuelian, S., & Zhanfeng, Z. (2014). Based on Ant Colony Algorithm to Solve the Mobile Robots Intelligent Path Planning for Avoid Obstacles. International Journal of Artificial Intelligence & Applications,5(1), 1-21. doi:10.5121/ijaia.2014.5101

[11] Zeng, C., Zhang, Q., & Wei, X. (2012). GA-based Global Path Planning for Mobile Robot Employing A* Algorithm. Journal of Computers, 7(2). doi:10.4304/jcp.7.2.470-474

[12] Singh, V., & Willcox, K. E. (2016). Methodology for Path Planning with Dynamic Data-Driven Flight Capability Estimation. 17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference. doi:10.2514/6.2016-4124

[13] Air Traffic Control Center Weather Services. (n.d.). Retrieved February 30, 2017, from http://www.nws.noaa.gov/om/aviation/cwsu/CWSUs_1_pager.pdf

[14] Mitchell, M. (1998). An Introduction to Genetic Algorithms. Cambridge, MA: MIT.

APPENDIX A- FLOWCHART FIGURES

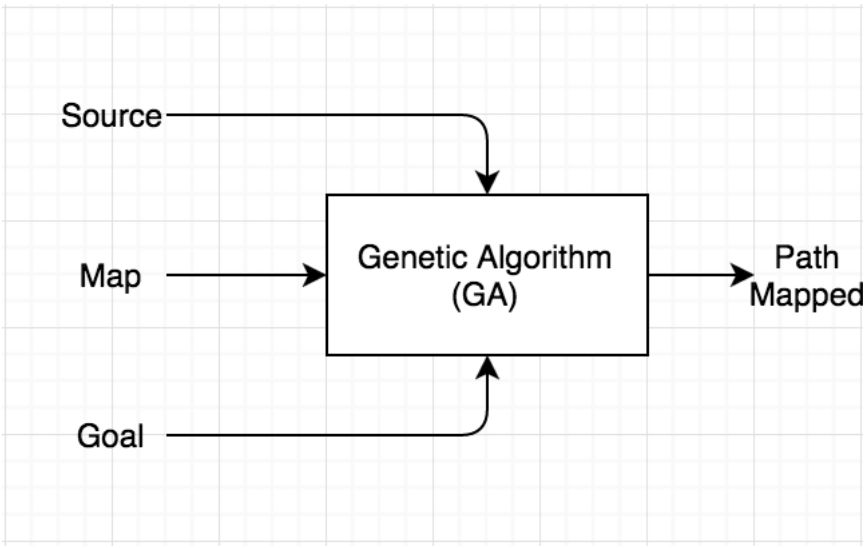


Figure 32: *Flowchart figure for the setup of the GA path planner*

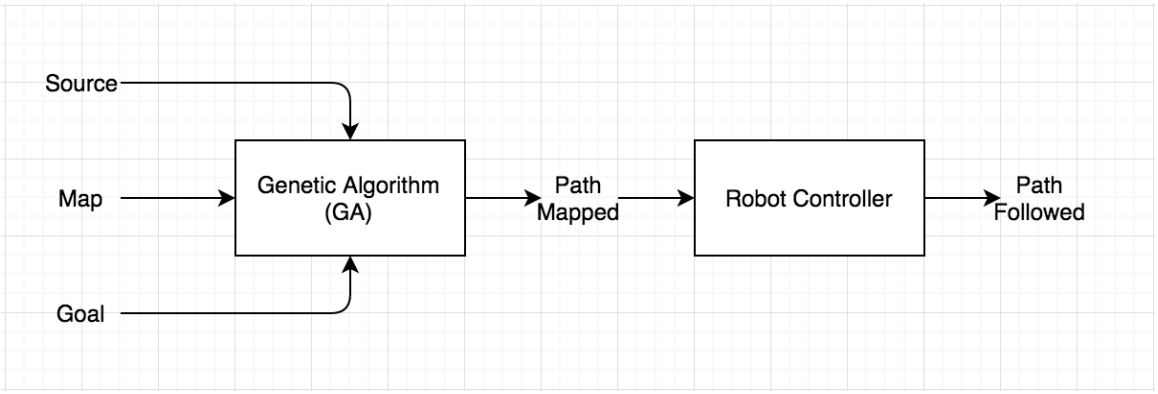


Figure 33: *Flowchart figure for GA path planner and robot controller*

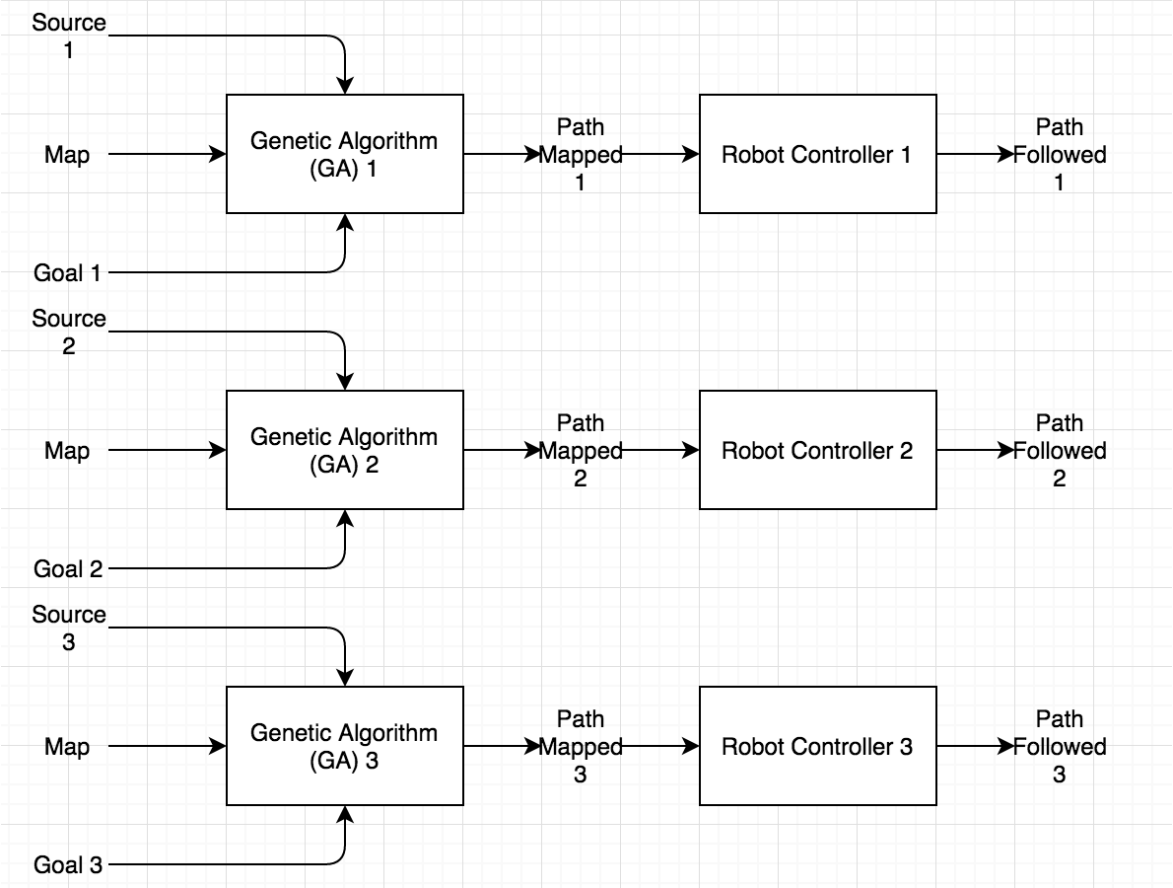


Figure 34: *Flowchart figure for parallel GA setup for multiple robot experimental setup*

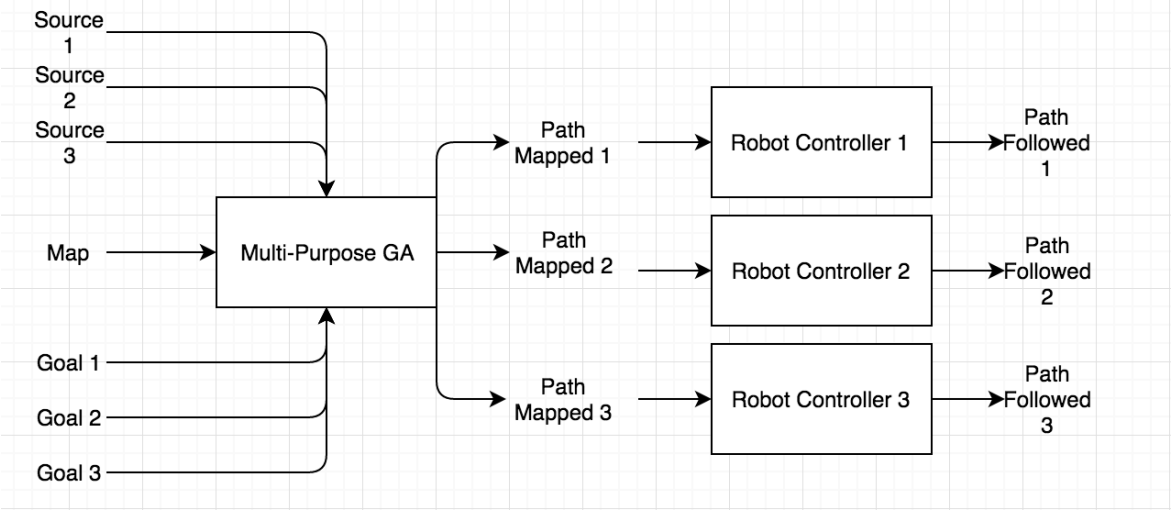


Figure 35: *Flowchart figure for single multipurpose GA for multi robot experimental setup*

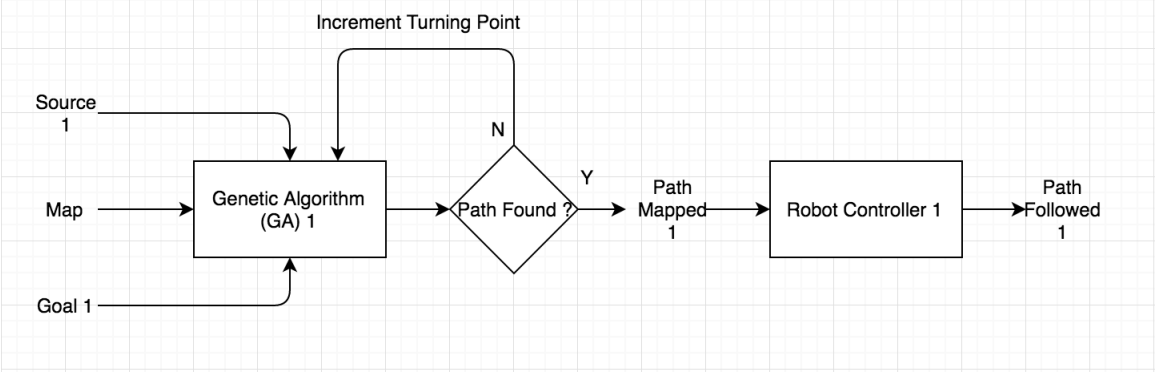


Figure 36- *Flowchart figure used for the incrementing turning points experiment*

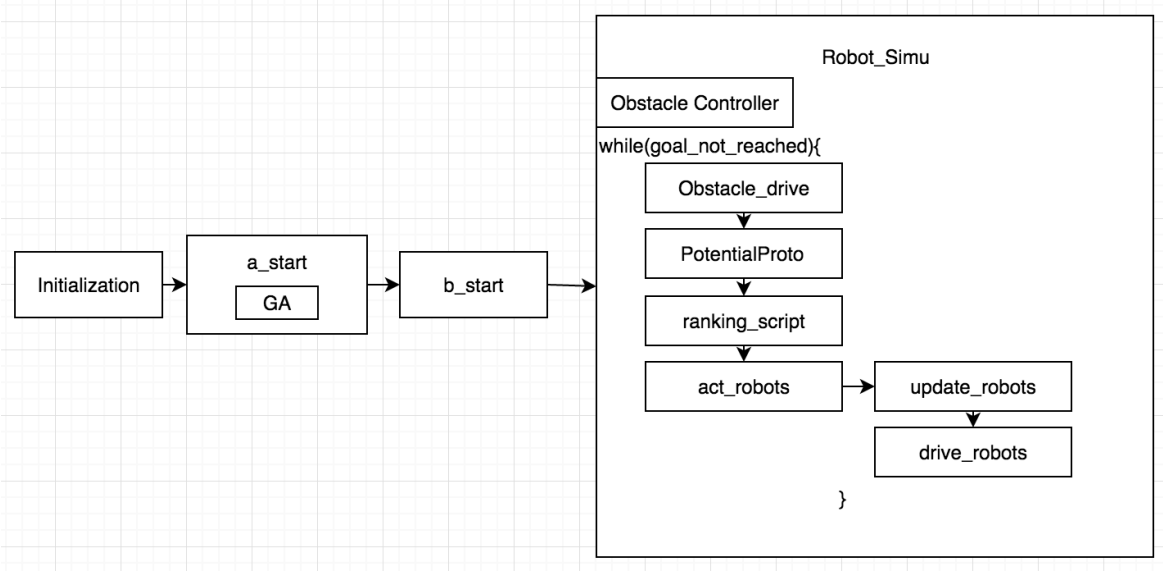


Figure 37: *Flowchart figure for the coded scripts setup*

APPENDIX B – MAP FIGURES

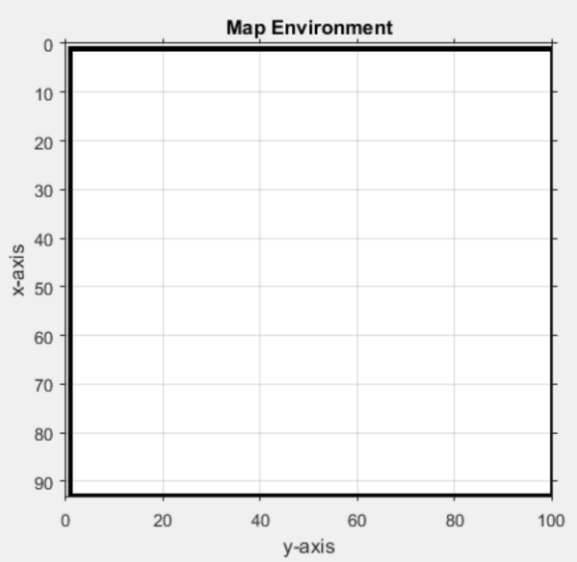


Figure 38: *Empty Map (size: 100x100)*

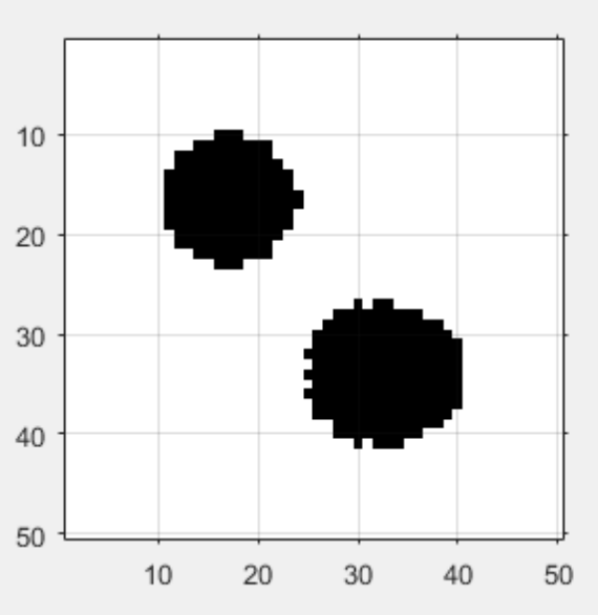


Figure 39: *50x50 static map 1*

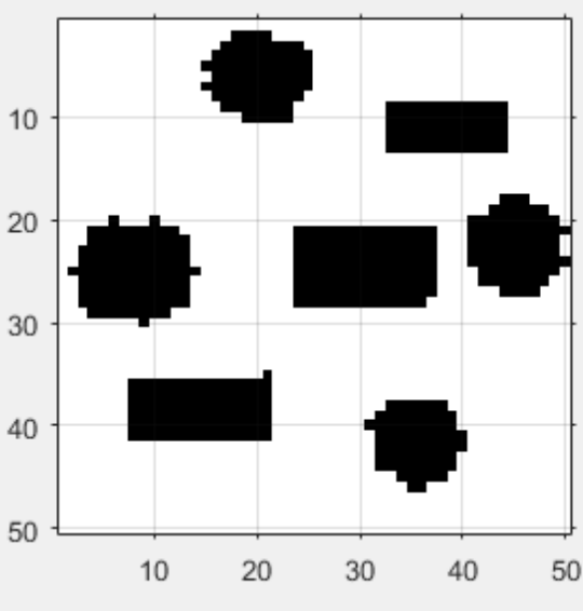


Figure 40: *Static Map 2*

APPENDIX C- GENETIC ALGORITHM DATA

Table 21*- Table below shows the Testing data for the Single-Point Crossover Genetic Algorithm Testing*

# of Generations	5		50	
Path Length (cm)	67		55	
Path Coordinates	Y (cm)	X (cm)	Y (cm)	X (cm)
	45.00	40.00	45.00	40.00
	44.54	33.78	44.11	37.86
	47.97	25.47	42.68	27.51
	38.77	7.72	25.99	12.90
	25.02	5.28	15.14	7.79
	16.32	6.98	9.76	6.13
	5.00	5.00	5.00	5.00

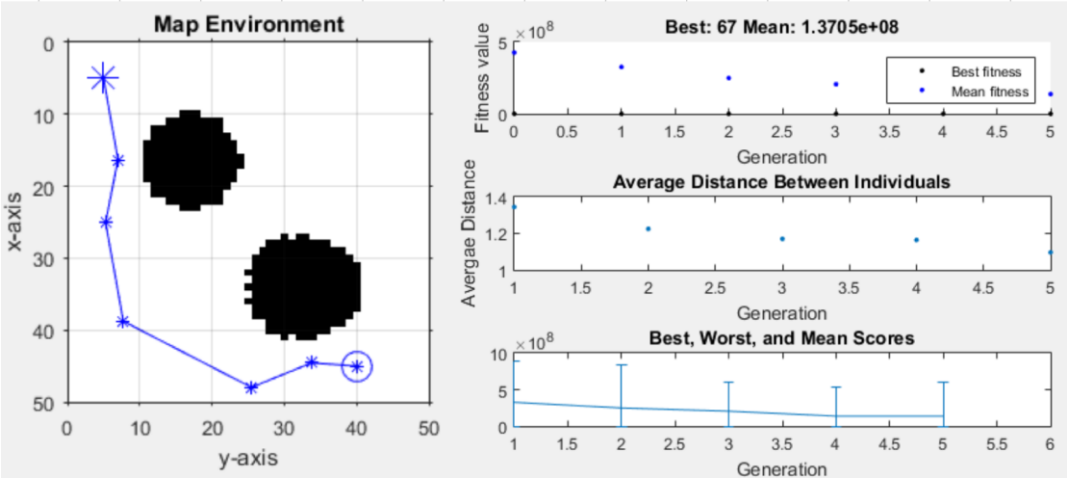


Figure 41 *shows the planned path using the GA (Single-Point Cross Over) for 5 generations*

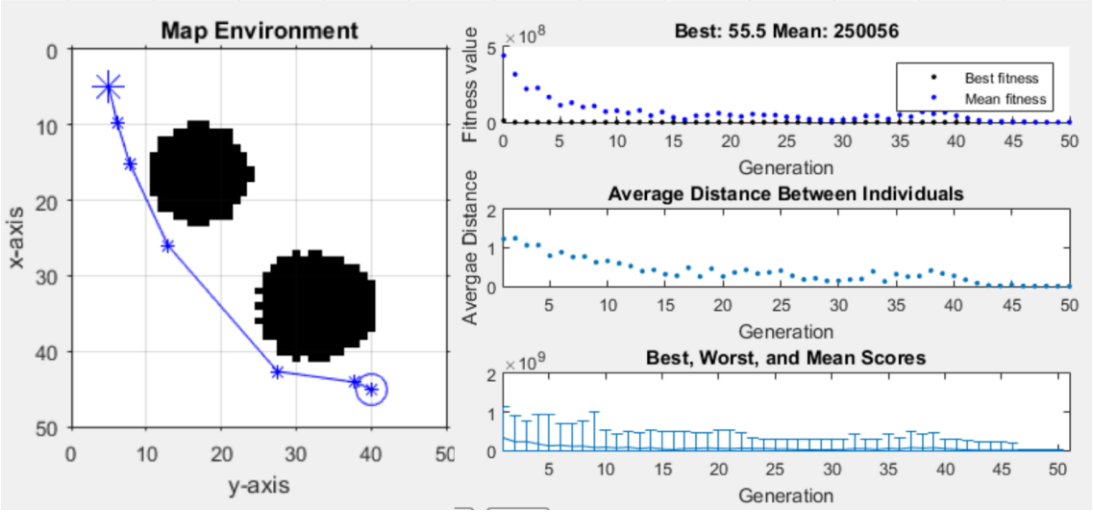


Figure 42 *shows the planned path using the GA (Single-Point Cross Over) for 50 generations*

Table 22*- Table below shows the Testing data for the Two-Point Crossover Genetic Algorithm Testing*

# of Generations	5		50	
Path Length (cm)	93.5		61	
Path Coordinates	Y (cm)	X (cm)	Y (cm)	X (cm)
	45.00	40.00	45.00	40.00
	40.75	23.08	44.14	41.02
	35.69	20.34	35.82	42.51
	26.50	19.20	21.35	38.75
	27.66	24.56	17.26	35.42
	42.44	11.93	6.67	15.86
	5.00	5.00	5.00	5.00

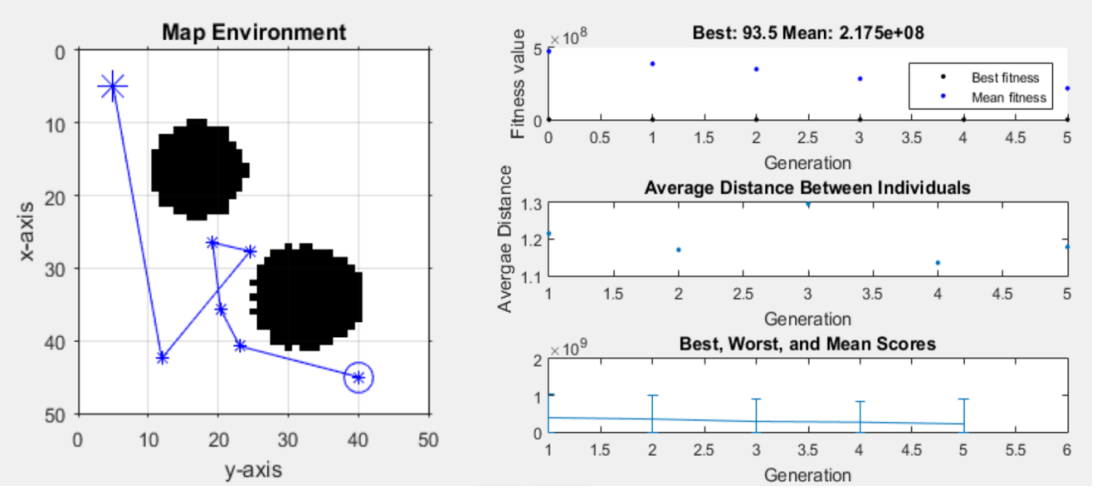


Figure 43 *shows the planned path using the GA (Two-Point Cross Over) for 5 generations*

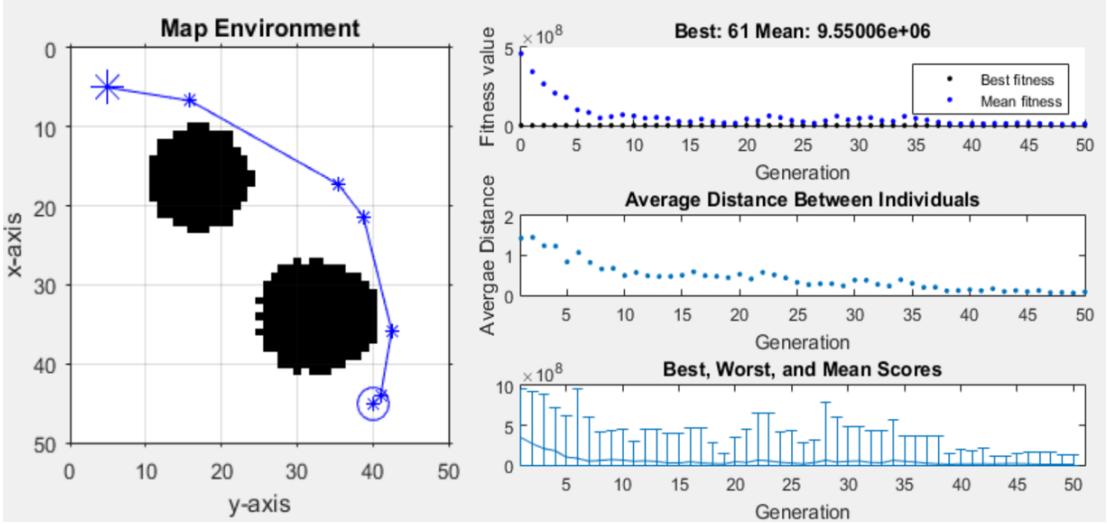


Figure 44 *shows the planned path using the GA (Two-Point Cross Over) for 50 generations*

Table 23- *Table below shows the Testing data for the Scattered Crossover GA Testing*

# of Generations	5		50	
Path Length (cm)	73		61	
Path Coordinates	Y (cm)	X (cm)	Y (cm)	X (cm)
	45.00	40.00	45.00	40.00
	42.74	33.09	44.16	35.51
	47.53	12.96	39.00	20.46

	39.73	13.50	35.96	18.29
	23.24	2.15	32.42	13.73
	9.80	3.04	14.30	6.93
	5.00	5.00	5.00	5.00

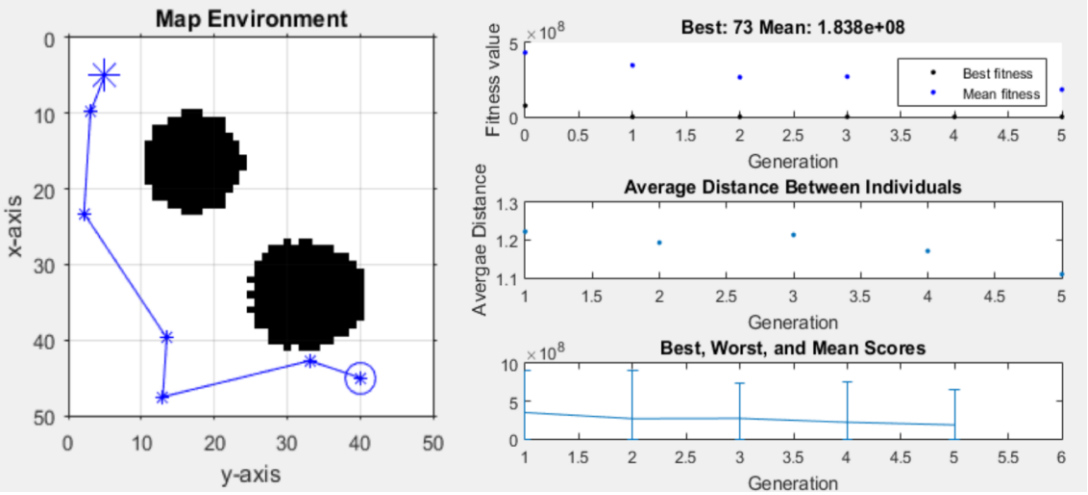


Figure 45 *shows the planned path using the GA (Scattered Cross Over) for 5 generations*

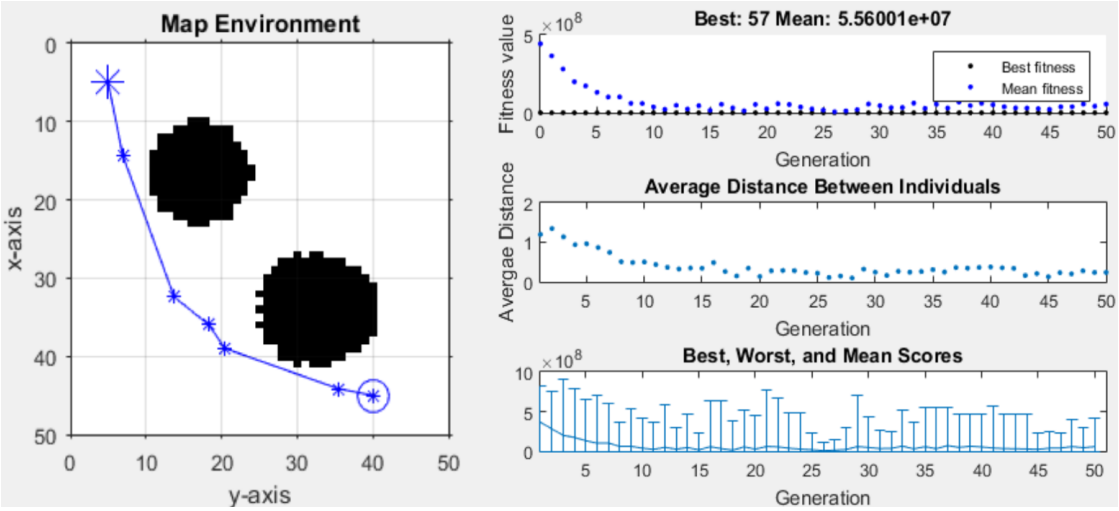


Figure 46 *shows the planned path using the GA (Scattered Cross Over) for 50 generations*

APPENDIX D: CODE SCRIPT

```
%-------------------------------------------------------------------------
--%
%macro.m
Initialization;
a_start;
Robot_Simu;
%-------------------------------------------------------------------------
--%
%Initialization.m
'Initialization start'


%Code for Initialization to be run first time only ( see planogram)
global  map source old_source goal source2 old_source2 goal2 source3
old_source3 goal3 oldmap map2  splineSmoothing
global  obs_source obs_old_source obs_goal obs_source2 obs_old_source2
obs_goal2 obs_source3 obs_old_source3 obs_goal3


% input map read from a bmp file
%map 1 is the map's first state
map=im2bw(imread('map2_50x50.bmp'));

%------------Robots-----------------%
%Robot 1
source=[45 5];
old_source=source;
goal=[5 30];

%Robot 2
source2=[40 5];
old_source2=source2;
goal2=[10 40];

%Robot 3
```

```
source3=[45 10];
old_source3=source3;
goal3=[10 30];
%-------------------------------------%
%------------Obstacles----------------%
%Obstacle 1
obs_source=[28 21];%28 16
obs_old_source=obs_source;
obs_goal=[28 16];%28 22

obs_path(1,:)=obs_source;
obs_path(2,:)=obs_goal;

%Robot 2
obs_source2=[35 5];
obs_old_source2=source2;
obs_goal2=[45 15];

obs_path2(1,:)=obs_source2;
obs_path2(2,:)=obs_goal2;

%Robot 3
obs_source3=[10 26];
obs_old_source3=source3;
obs_goal3=[20 36];

obs_path3(1,:)=obs_source3;
obs_path3(2,:)=obs_goal3;
%-------------------------------------%
i=1;
j=1;
t=1;
x=0;
f=0;
firstrun=0;
o=0;
Q=2;
controlrate=1000000;%10000
speedfactor=1.0;
slowfactor=0.5;
stopfactor=0;

first_attempt_1=0;
first_attempt_2=0;
first_attempt_3=0;

first_time=0;
'Initialization complete'
%-------------------------------------------------------------------------
--%
%a_start.m
'A_Start'

% Num of points that represent a candidate path, excluding the source
and goal
```

```
noOfPointsInSolution1=1;
noOfPointsInSolution2=1;
noOfPointsInSolution3=1;
NoOfGenerations=50;%50
PopulationSize=100;
splineSmoothing=false;


tic;

if ~feasiblePoint(source,map), error('source lies on an obstacle or
outside map');
end
if ~feasiblePoint(goal,map), error('goal lies on an obstacle or outside
map');
end

if ~feasiblePoint(source2,map), error('source2 lies on an obstacle or
outside map');
end
if ~feasiblePoint(goal2,map), error('goal2 lies on an obstacle or
outside map');
end
if ~feasiblePoint(source3,map), error('source3 lies on an obstacle or
outside map');
end
if ~feasiblePoint(goal3,map), error('goal3 lies on an obstacle or
outside map');
end


options=gaoptimset('PlotFcns', {@gaplotbestf, @gaplotdistance,
@gaplotrange},'Generations',NoOfGenerations,'PopulationSize',PopulationS
ize,'CrossoverFcn',@crossoversinglepoint);

[solution cost] = ga(@PathCostGA_R1,
noOfPointsInSolution1*2,[],[],[],[],zeros(noOfPointsInSolution1*2,1),one
s(noOfPointsInSolution1*2,1),[],options);
path=[source; [solution(1:2:end)'*size(map,1)
solution(2:2:end)'*size(map,2)]; goal];

[solution2 cost2] = ga(@PathCostGA_R2,
noOfPointsInSolution2*2,[],[],[],[],zeros(noOfPointsInSolution2*2,1),one
s(noOfPointsInSolution2*2,1),[],options);
path2=[source2; [solution2(1:2:end)'*size(map,1)
solution2(2:2:end)'*size(map,2)]; goal2];

[solution3 cost3] = ga(@PathCostGA_R3,
noOfPointsInSolution3*2,[],[],[],[],zeros(noOfPointsInSolution3*2,1),one
s(noOfPointsInSolution3*2,1),[],options);
path3=[source3; [solution3(1:2:end)'*size(map,1)
solution3(2:2:end)'*size(map,2)]; goal3];
```

```
while(PathCostGA_R1(solution)>size(map,1)*size(map,2) ||
PathCostGA_R2(solution2)>size(map,1)*size(map,2)||
PathCostGA_R3(solution3)>size(map,1)*size(map,2))
    if PathCostGA_R1(solution)>size(map,1)*size(map,2)
    'Path 1 ERROR: NO PATH WAS FOUND for path1'
    noOfPointsInSolution1=noOfPointsInSolution1+1;
    [solution cost] = ga(@PathCostGA_R1,
noOfPointsInSolution1*2,[],[],[],[],zeros(noOfPointsInSolution1*2,1),one
s(noOfPointsInSolution1*2,1),[],options);
    path=[source; [solution(1:2:end)'*size(map,1)
solution(2:2:end)'*size(map,2)]; goal];
    end

    if PathCostGA_R2(solution2)>size(map,1)*size(map,2)
    'Path2 ERROR: NO PATH WAS FOUND for path2'
    noOfPointsInSolution2=noOfPointsInSolution2+1;
    [solution2 cost2] = ga(@PathCostGA_R2,
noOfPointsInSolution2*2,[],[],[],[],zeros(noOfPointsInSolution2*2,1),one
s(noOfPointsInSolution2*2,1),[],options);
    path2=[source2; [solution2(1:2:end)'*size(map,1)
solution2(2:2:end)'*size(map,2)]; goal2];
    end

    if PathCostGA_R3(solution3)>size(map,1)*size(map,2)
    'Path3 ERROR: NO PATH WAS FOUND for path3'
    noOfPointsInSolution3=noOfPointsInSolution3+1;
    [solution3 cost3] = ga(@PathCostGA_R3,
noOfPointsInSolution3*2,[],[],[],[],zeros(noOfPointsInSolution3*2,1),one
s(noOfPointsInSolution3*2,1),[],options);
    path3=[source3; [solution3(1:2:end)'*size(map,1)
solution3(2:2:end)'*size(map,2)]; goal3];
    end
end


if splineSmoothing
   path=bsp(path);
end

'A_Complete'

b_start;
%-------------------------------------------------------------------------
--%
%b_start.m
'B_Start'
H=size(path,1);
H2=size(path2,1);
H3=size(path3,1);

 bsp_=false;
    if bsp_
        path=bsp(path)
```

```
        path2=bsp(path2)
        path3=bsp(path3)
    end



        oldpath=round(path);
        oldpath2=round(path2);
        oldpath3=round(path3);

'B_Complete'
%-------------------------------------------------------------------------
--%
%Robot_simu.m
plotpath;
i=1;%row counter
j=1;%col counter
max_rows=size(map,1);
max_cols=size(map,2);
featurematrix=[];

path_r1=[];
path_r2=[];
path_r3=[];

%forward3=1;
Q1=3;
Q11=3;
Q2=3;
Q22=3;
Q3=3;
Q33=3;
mem=0;
mem1=Q1-1;
mem11=Q11-1;
mem2=Q2-1;
mem22=Q22-1;
mem3=Q3-1;
mem33=Q33-1;

    r1action=0;
    r2action=0;
    r3action=0;

    r12rank=0;
    r13rank=0;
    r23rank=0;



for j=1:max_cols
    for i=1:max_rows
        if map(i,j)==0;%check if current cell is obstacle-filled
            Loc=[i;j];
            featurematrix=horzcat(featurematrix,Loc);
```

```
            %featurematrix([1 2],:) = featurematrix([2 1],:);
            %featurematrix(
        end
    end
end

MapFeatures=size(featurematrix,2);



    imshow(map);
    axis on;
    grid on;
    hold on;

    robotRadius = 1.0;



   %opes map window
   xlabel('y-axis') % x-axis label
   ylabel('x-axis') % y-axis label
   axis([0 size(map,2) 0 size(map,1)])

map_backup=map;

if firstrun==0
        map_backupxor = xor(map_backup,1);
        map_backup = robotics.BinaryOccupancyGrid(map_backupxor);

        oldpath=fliplr(oldpath);
        oldpath2=fliplr(oldpath2);
        oldpath3=fliplr(oldpath3);

        obs_path=fliplr(obs_path);
        obs_path2=fliplr(obs_path2);
        obs_path3=fliplr(obs_path3);







        firstrun=1;
end







plot(path(:,2), path(:,1),'k--d')
plot(path2(:,2), path2(:,1),'k--d')
plot(path3(:,2), path3(:,1),'k--d')
```

```
robot1CurrentLocation = oldpath(1,:);
robot1Goal = oldpath(end,:);

robot2CurrentLocation = oldpath2(1,:);
robot2Goal = oldpath2(end,:);

robot3CurrentLocation = oldpath3(1,:);
robot3Goal = oldpath3(end,:);




initialOrientation1 = 0;
initialOrientation2 = 0;
initialOrientation3 = 0;



robot1CurrentPose = [robot1CurrentLocation initialOrientation1];
robot2CurrentPose = [robot2CurrentLocation initialOrientation2];
robot3CurrentPose = [robot3CurrentLocation initialOrientation3];



robot1 = ExampleHelperDifferentialDriveRobot(robot1CurrentPose);
robot2 = ExampleHelperDifferentialDriveRobot(robot2CurrentPose);
robot3 = ExampleHelperDifferentialDriveRobot(robot3CurrentPose);




controller1 = robotics.PurePursuit;
controller2 = robotics.PurePursuit;
controller3 = robotics.PurePursuit;




controller1.Waypoints = oldpath;
controller2.Waypoints = oldpath2;
controller3.Waypoints = oldpath3;




controller1.DesiredLinearVelocity = 1.5;
```

```
controller2.DesiredLinearVelocity = 1.5;
controller3.DesiredLinearVelocity = 1.5;



controller1.MaxAngularVelocity = 10;
controller2.MaxAngularVelocity = 10;
controller3.MaxAngularVelocity = 10;



controller1.LookaheadDistance = 1.5;
controller2.LookaheadDistance = 1.5;
controller3.LookaheadDistance = 1.5;



goalRadius1 = 0.2;
distanceToGoal = norm(robot1CurrentLocation - robot1Goal);

goalRadius2 = 0.2;
distanceToGoal2 = norm(robot2CurrentLocation - robot2Goal);

goalRadius3 = 0.2;
distanceToGoal3 = norm(robot3CurrentLocation - robot3Goal);

obstaclecontroller
O=0;



while( distanceToGoal > goalRadius1 || distanceToGoal2 > goalRadius2 ||
distanceToGoal3 > goalRadius3)


   [v1, omega1] = step(controller1, robot1.CurrentPose);
   [v2, omega2] = step(controller2, robot2.CurrentPose);
   [v3, omega3] = step(controller3, robot3.CurrentPose);


    obstacledrive


    PotentialProto;

    obs_robot1CurrentLocation;

              locationlist;


    if(distanceToGoal>=goalRadius1)
```

```
            path_r1(Q1,:)=robot1CurrentLocation;


            Theta1_current=robot1.CurrentPose(3);%Current Orientation
Saved

            drive(robot1, v1, omega1)
            PotentialProto;


            robot1CurrentLocation = robot1.CurrentPose(1:2);
            distanceToGoal = norm(robot1CurrentLocation - robot1Goal);


            robot1CurrentLocation;
            if Q1-mem1>1
                xRobot1_previous=transpose(path_r1(Q1-mem1,:));

            else
                mem1=1;
                xRobot1_previous=transpose(path_r1(Q1,:));
            end
            %xRobot1_previous=transpose(path_r1(Q1-mem,:))
            xRobot1_current=transpose(path_r1(Q1,:));
    else
        v1=0;%stop robot1 when reached destination
    end



    if(distanceToGoal2>=goalRadius2)
            path_r2(Q2,2)=robot2CurrentLocation(2);

            drive(robot2, v2, omega2)
            PotentialProto;

            robot2CurrentLocation = robot2.CurrentPose(1:2);
            distanceToGoal2 = norm(robot2CurrentLocation - robot2Goal);

            path_r2(Q2,:)=robot2CurrentLocation;
            Q2;
            mem2;
            if Q2-mem2>1
                xRobot2_previous=transpose(path_r2(Q2-mem2,:));

            else
                mem2=1;
                xRobot2_previous=transpose(path_r2(Q2,:));
            end

               xRobot2_current=transpose(path_r2(Q2,:));
```

```
    else
        v2=0;
    end


    if(distanceToGoal3>=goalRadius3)
        path_r3(Q3,:)=robot3CurrentLocation;

        Theta3_current=robot3.CurrentPose(3); %Current Orientation Saved
         drive(robot3,v3,omega3)

            robot3CurrentLocation = robot3.CurrentPose(1:2);
            distanceToGoal3 = norm(robot3CurrentLocation - robot3Goal);


            if Q3-mem3>1
                xRobot3_previous=transpose(path_r3(Q3-mem3,:));

            else
                mem3=1;
                xRobot3_previous=transpose(path_r3(Q3,:));
            end

               xRobot3_current=transpose(path_r3(Q3,:));

    else
        v3=0;%stop robot when reached destination

    end




end
%-------------------------------------------------------------------------
--%
%obstaclecontroller.m

obs_robot1CurrentLocation = obs_path(1,:);
obs_robotGoal = obs_path(end,:);

obs_robot2CurrentLocation = obs_path2(1,:);
obs_robot2Goal = obs_path2(end,:);

obs_robot3CurrentLocation = obs_path3(1,:);
obs_robot3Goal = obs_path3(end,:);

obs_initialOrientation = 0;
obs_initialOrientation2 = 0;
obs_initialOrientation3 = 0;

obs_robotCurrentPose = [obs_robot1CurrentLocation
```

```
obs_initialOrientation];
obs_robot2CurrentPose = [obs_robot2CurrentLocation
obs_initialOrientation2];
obs_robot3CurrentPose = [obs_robot3CurrentLocation
obs_initialOrientation3];

obs_robot = ExampleHelperDifferentialDriveRobot(obs_robotCurrentPose);
obs_robot2 = ExampleHelperDifferentialDriveRobot(obs_robot2CurrentPose);
obs_robot3 = ExampleHelperDifferentialDriveRobot(obs_robot3CurrentPose);

obs_controller = robotics.PurePursuit
obs_controller2 = robotics.PurePursuit
obs_controller3 = robotics.PurePursuit




obs_controller.Waypoints = obs_path;
obs_controller2.Waypoints = obs_path2;
obs_controller3.Waypoints = obs_path3;

obs_controller.DesiredLinearVelocity = 0.5;
obs_controller2.DesiredLinearVelocity = 0.5;
obs_controller3.DesiredLinearVelocity = 0.5;

obs_controller.MaxAngularVelocity = 12;
obs_controller2.MaxAngularVelocity = 12;
obs_controller3.MaxAngularVelocity = 12;

obs_controller.LookaheadDistance = 0.1;
obs_controller2.LookaheadDistance = 0.1;
obs_controller3.LookaheadDistance = 0.1;

obs_goalRadius = 0.1;
obs_distanceToGoal = norm(obs_robot1CurrentLocation - obs_robotGoal);

obs_goalRadius2 = 0.1;
obs_distanceToGoal2 = norm(obs_robot2CurrentLocation - obs_robot2Goal);

obs_goalRadius3 = 0.1;
obs_distanceToGoal3 = norm(obs_robot3CurrentLocation - obs_robot3Goal);
%-------------------------------------------------------------------------
--%
   %obstacledrive.m
   [obs_v, obs_omega] = step(obs_controller, obs_robot.CurrentPose);
   [obs_v2, obs_omega2] = step(obs_controller2, obs_robot2.CurrentPose);
   [obs_v3, obs_omega3] = step(obs_controller3, obs_robot3.CurrentPose);

   if(obs_distanceToGoal>=obs_goalRadius)
        drive(obs_robot, obs_v, obs_omega)
        obs_robot1CurrentLocation = obs_robot.CurrentPose(1:2);
        obs_distanceToGoal = norm(obs_robot1CurrentLocation -
obs_robotGoal);
    else
```

```
        %Switch path go backwards
        obs_path=flipud(obs_path);
        temp_source=obs_source;
        temp_goal=obs_goal;
        obs_source=temp_goal;
        obs_goal=temp_source;
        obstaclecontroller1;



   end

   if(obs_distanceToGoal2>=obs_goalRadius2)
        drive(obs_robot2, obs_v2, obs_omega2)
        obs_robot2CurrentLocation = obs_robot2.CurrentPose(1:2);
        obs_distanceToGoal2 = norm(obs_robot2CurrentLocation -
obs_robot2Goal);
    else
        %Switch path go backwards
        obs_path2=flipud(obs_path2);
        temp_source2=obs_source2;
        temp_goal2=obs_goal2;
        obs_source2=temp_goal2;
        obs_goal2=temp_source2;
        obstaclecontroller2;
   end

    if(obs_distanceToGoal3>=obs_goalRadius3)
        drive(obs_robot3, obs_v3, obs_omega3)
        obs_robot3CurrentLocation = obs_robot3.CurrentPose(1:2);
        obs_distanceToGoal3 = norm(obs_robot3CurrentLocation -
obs_robot3Goal);
    else
        %Switch path go backwards
        obs_path3=flipud(obs_path3);

        temp_source3=obs_source3;
        temp_goal3=obs_goal3;
        obs_source3=temp_goal3;
        obs_goal3=temp_source3;
        obstaclecontroller3;
    end
%-------------------------------------------------------------------------
--%
%locationlist.m
Q1=Q1+1;
Q11=Q11+1;
Q2=Q2+1;
Q22=Q22+1;
Q3=Q3+1;
Q33=Q33+1;

%mem=1;

path_r1(Q1,1)=robot1CurrentLocation(1);
```

```
path_r1(Q1,2)=robot1CurrentLocation(2);

path_r2(Q2,1)=robot2CurrentLocation(1);
path_r2(Q2,2)=robot2CurrentLocation(2);

path_r3(Q3,1)=robot3CurrentLocation(1);
path_r3(Q3,2)=robot3CurrentLocation(2);

%Q is current location
%Q-1 is previous Location
%-------------------------------------------------------------------------
--%
%PotentialProto.m
Theta1=omega1;
Theta2=omega2;
Theta3=omega3;

%VPR=Variable Per Robot
xSource1=transpose(source);%VPR
xSource2=transpose(source2);%VPR
xSource3=transpose(source3);%VPR

xGoal1=transpose(goal);%VPR
xGoal2=transpose(goal2);%VPR
xGoal3=transpose(goal3);%VPR

xRobot1=transpose(robot1CurrentLocation);%VPR
xRobot2=transpose(robot2CurrentLocation);%VPR
xRobot3=transpose(robot3CurrentLocation);%VPR

RadiusOfInfluence=1.2;
RadiusOfInfluenceEnv=0.5;%4
RadiusOfInfluence_dynamic=1.2;

KEnvironment=3;%1
KGoal=0.05;%0.6
KObj=0.5;
Kdynamic=0.3;
KTotal=0.01;%0.001

Goal_Error1 =  transpose(xGoal1-xRobot1);%VPR
Goal_Error2 =  transpose(xGoal2-xRobot2);%VPR
Goal_Error3 =  transpose(xGoal3-xRobot3);%VPR

FGoal_1=transpose(1*(Goal_Error1)/norm(Goal_Error1));
FGoal_2=transpose(1*(Goal_Error2)/norm(Goal_Error2));
FGoal_3=transpose(1*(Goal_Error3)/norm(Goal_Error3));


%Compute Distance between Robots
%Robot1-Robot2
XDistance12=xRobot1(1)-xRobot2(1);%VPR
YDistance12=xRobot1(2)-xRobot2(2);%VPR
Dp12=[XDistance12;YDistance12];%VPR
```

```
Distance_12 = sqrt(sum(Dp12.^2));%VPR
iInfluencial_12 = find(Distance_12<RadiusOfInfluence);%VPR


%Robot1-Robot3
XDistance13=xRobot1(1)-xRobot3(1);%VPR
YDistance13=xRobot1(2)-xRobot3(2);%VPR
Dp13=[XDistance13;YDistance13];%VPR
Distance_13 = sqrt(sum(Dp13.^2));%VPR
iInfluencial_13 = find(Distance_13<RadiusOfInfluence);%VPR

%Robot2-Robot3
XDistance23=xRobot2(1)-xRobot3(1);%VPR
YDistance23=xRobot2(2)-xRobot3(2);%VPR
Dp23=[XDistance23;YDistance23];%VPR
Distance_23 = sqrt(sum(Dp23.^2));%VPR
iInfluencial_23 = find(Distance_23<RadiusOfInfluence);%VPR


%Compute Distance between Robot1 and Environment
%R1-Env
Dp1Env = featurematrix-repmat(xRobot1,1,MapFeatures);
Distance1Env = sqrt(sum(Dp1Env.^2));
iInfluencial1Env = find(Distance1Env<RadiusOfInfluenceEnv);

%R2-Env
Dp2Env = featurematrix-repmat(xRobot2,1,MapFeatures);
Distance2Env = sqrt(sum(Dp2Env.^2));
iInfluencial2Env = find(Distance2Env<RadiusOfInfluenceEnv);

%R3-Env
Dp3Env = featurematrix-repmat(xRobot3,1,MapFeatures);
Distance3Env = sqrt(sum(Dp3Env.^2));
iInfluencial3Env = find(Distance3Env<RadiusOfInfluenceEnv);

dynamicobstacles




%while(~isempty(iInfluencial_12) || ~isempty(iInfluencial_13) ||
~isempty(iInfluencial_23)  || ~isempty(iInfluencial_R3O2) ||
~isempty(iInfluencial_R3O3) )


    (iInfluencial_12);
    (iInfluencial_13);
    (iInfluencial_23);

%R1
%R1-R2
    if(~isempty(iInfluencial_12))%R1-R2
        %vector sum of repulsions:
        rho12 = repmat(Distance_12(iInfluencial_12),2,1);
```

```
        V12 = Dp12(:,iInfluencial_12);
        DrhoDx12 = -V12./rho12;
        F12 = (1./rho12-1./RadiusOfInfluence)*1./(rho12.^2).*DrhoDx12;
        FObjects_12 = KObj*sum(F12,2);
    else
        %nothing close
        FObjects_12 = [0;0];
    end
%R1-R3
    if(~isempty(iInfluencial_13))%R1-R3
        %vector sum of repulsions:
        rho13 = repmat(Distance_13(iInfluencial_13),2,1);
        V13 = Dp13(:,iInfluencial_13);
        DrhoDx13 = -V13./rho13;
        F13 = (1./rho13-1./RadiusOfInfluence)*1./(rho13.^2).*DrhoDx13;
        FObjects_13 = KObj*sum(F13,2);

    else
        %nothing close
        FObjects_13 = [0;0];
    end

%R2
%R2-R1
    if(~isempty(iInfluencial_12))%R2-R1
        %vector sum of repulsions:
        rho21 = repmat(Distance_12(iInfluencial_12),2,1);
        V21 = Dp12(:,iInfluencial_12);
        DrhoDx21 = -V21./rho21;
        %F21 = (1./rho21-1./RadiusOfInfluence)*1./(rho21.^2).*DrhoDx21;
        F21=(1./rho21-1./RadiusOfInfluence)*1./(rho21.^2).*DrhoDx21;;
        FObjects_21 = KObj*sum(F21,2);
    else
        %nothing close
        FObjects_21 = [0;0];
    end
%R2-R3
    if(~isempty(iInfluencial_23))%R2-R3
        %vector sum of repulsions:
        rho23 = repmat(Distance_23(iInfluencial_23),2,1);

        V23 = Dp23(:,iInfluencial_23);
        DrhoDx23 = -V23./rho23;
        F23 = (1./rho23-1./RadiusOfInfluence)*1./(rho23.^2).*DrhoDx23;
        FObjects_23 = KObj*sum(F23,2);
    else
        %nothing close
        FObjects_23 = [0;0];
    end

%R3
%R3-R1
    if(~isempty(iInfluencial_13))%R3-R1
        %vector sum of repulsions:
        rho31 = repmat(Distance_13(iInfluencial_13),2,1);
```

```
        V31 = Dp13(:,iInfluencial_13);
        DrhoDx31 = -V31./rho31;
        %F31 = (1./rho31-1./RadiusOfInfluence)*1./(rho31.^2).*DrhoDx31;
        F31=(1./rho31-1./RadiusOfInfluence)*1./(rho31.^2).*DrhoDx31;;
        FObjects_31 = KObj*sum(F31,2);
    else
        %nothing close
        FObjects_31 = [0;0];
    end
%R3-R2
    if(~isempty(iInfluencial_23))%R3-R2
        %vector sum of repulsions:
        rho32 = repmat(Distance_23(iInfluencial_23),2,1);
        V32 = Dp23(:,iInfluencial_23);
        DrhoDx32 = -V32./rho32;
        %F32 = (1./rho32-1./RadiusOfInfluence)*1./(rho32.^2).*DrhoDx32;
        F32=(1./rho32-1./RadiusOfInfluence)*1./(rho32.^2).*DrhoDx32;
        FObjects_32 = KObj*sum(F32,2);
    else
        %nothing close
        FObjects_32 = [0;0];
    end

%R1-Environment
    if(~isempty(iInfluencial1Env))
        %vector sum of repulsions:
        rho1Env = repmat(Distance1Env(iInfluencial1Env),2,1);
        V1Env = Dp1Env(:,iInfluencial1Env);
        DrhoDx1Env = -V1Env./rho1Env;
        F1Env = (1./rho1Env-
1./RadiusOfInfluenceEnv)*1./(rho1Env.^2).*DrhoDx1Env;
        FObjects1Env = KEnvironment*sum(F1Env,2);
    else
        %nothing close
        FObjects1Env = [0;0];
    end;
%R2-Environment
if(~isempty(iInfluencial2Env))
        %vector sum of repulsions:
        rho2Env = repmat(Distance2Env(iInfluencial2Env),2,1);
        V2Env = Dp2Env(:,iInfluencial2Env);
        DrhoDx2Env = -V2Env./rho2Env;
        F2Env = (1./rho2Env-
1./RadiusOfInfluenceEnv)*1./(rho2Env.^2).*DrhoDx2Env;
        FObjects2Env = KEnvironment*sum(F2Env,2);

    else
        %nothing close
        FObjects2Env = [0;0];
    end;
%R3-Environment

if(~isempty(iInfluencial3Env))


        %vector sum of repulsions:
```

```
        rho3Env = repmat(Distance3Env(iInfluencial3Env),2,1);
        V3Env = Dp3Env(:,iInfluencial3Env);
        DrhoDx3Env = -V3Env./rho3Env;
        F3Env = (1./rho3Env-
1./RadiusOfInfluenceEnv)*1./(rho3Env.^2).*DrhoDx3Env;
        FObjects3Env = KEnvironment*sum(F3Env,2);
    else
        %nothing close

        FObjects3Env = [0;0];
    end;

%R1-O1
    if(~isempty(iInfluencial_R1O1))
        %vector sum of repulsions:
        rho_R1O1 = repmat(Distance_R1O1(iInfluencial_R1O1),2,1);
        V_R1O1 = Dp_R1O1(:,iInfluencial_R1O1);
        DrhoDx_R1O1 = -V_R1O1./rho_R1O1;
        F_R1O1 = (1./rho_R1O1-
1./RadiusOfInfluence_dynamic)*1./(rho_R1O1.^2).*DrhoDx_R1O1;
        FObjects_R1O1 = Kdynamic*sum(F_R1O1,2);
    else
        %nothing close
        FObjects_R1O1 = [0;0];
    end

%R1-O2
    if(~isempty(iInfluencial_R1O2))
        %vector sum of repulsions:
        rho_R1O2 = repmat(Distance_R1O2(iInfluencial_R1O2),2,1);
        V_R1O2 = Dp_R1O2(:,iInfluencial_R1O2);
        DrhoDx_R1O2 = -V_R1O2./rho_R1O2;
        F_R1O2 = (1./rho_R1O2-
1./RadiusOfInfluence_dynamic)*1./(rho_R1O2.^2).*DrhoDx_R1O2;
        FObjects_R1O2 = Kdynamic*sum(F_R1O2,2);
    else
        %nothing close
        FObjects_R1O2 = [0;0];
    end
%R1-O3
    if(~isempty(iInfluencial_R1O3))
        %vector sum of repulsions:
        rho_R1O3 = repmat(Distance_R1O3(iInfluencial_R1O3),2,1);
        V_R1O3 = Dp_R1O3(:,iInfluencial_R1O3);
        DrhoDx_R1O3 = -V_R1O3./rho_R1O3;
        F_R1O3 = (1./rho_R1O3-
1./RadiusOfInfluence_dynamic)*1./(rho_R1O3.^2).*DrhoDx_R1O3;
        FObjects_R1O3 = Kdynamic*sum(F_R1O3,2);
    else
        %nothing close
        FObjects_R1O3 = [0;0];
    end
%R2-O1
    if(~isempty(iInfluencial_R2O1))

        %vector sum of repulsions:
```

```
        rho_R2O1 = repmat(Distance_R2O1(iInfluencial_R2O1),2,1);
        V_R2O1 = Dp_R2O1(:,iInfluencial_R2O1);
        DrhoDx_R2O1 = -V_R2O1./rho_R2O1;
        F_R2O1 = (1./rho_R2O1-
1./RadiusOfInfluence_dynamic)*1./(rho_R2O1.^2).*DrhoDx_R2O1;
        FObjects_R2O1 = Kdynamic*sum(F_R2O1,2);
    else
        %nothing close
        FObjects_R2O1 = [0;0];
    end
%R2-O2
    if(~isempty(iInfluencial_R2O2))
        %vector sum of repulsions:
        rho_R2O2 = repmat(Distance_R2O2(iInfluencial_R2O2),2,1);
        V_R2O2 = Dp_R2O2(:,iInfluencial_R2O2);
        DrhoDx_R2O2 = -V_R2O2./rho_R2O2;
        F_R2O2 = (1./rho_R2O2-
1./RadiusOfInfluence_dynamic)*1./(rho_R2O2.^2).*DrhoDx_R2O2;
        FObjects_R2O2 = Kdynamic*sum(F_R2O2,2);
    else
        %nothing close
        FObjects_R2O2 = [0;0];
    end

%R2-O3
    if(~isempty(iInfluencial_R2O3))
        %vector sum of repulsions:
        rho_R2O3 = repmat(Distance_R2O3(iInfluencial_R2O3),2,1);
        V_R2O3 = Dp_R2O3(:,iInfluencial_R2O3);
        DrhoDx_R2O3 = -V_R2O3./rho_R2O3;
        F_R2O3 = (1./rho_R2O3-
1./RadiusOfInfluence_dynamic)*1./(rho_R2O3.^2).*DrhoDx_R2O3;
        FObjects_R2O3 = Kdynamic*sum(F_R2O3,2);
    else
        %nothing close
        FObjects_R2O3 = [0;0];
    end
%R3-O1
    if(~isempty(iInfluencial_R3O1))
        %vector sum of repulsions:
        rho_R3O1 = repmat(Distance_R3O1(iInfluencial_R3O1),2,1);
        V_R3O1 = Dp_R3O1(:,iInfluencial_R3O1);
        DrhoDx_R3O1 = -V_R3O1./rho_R3O1;
        F_R3O1 = (1./rho_R3O1-
1./RadiusOfInfluence_dynamic)*1./(rho_R3O1.^2).*DrhoDx_R3O1;
        FObjects_R3O1 = Kdynamic*sum(F_R3O1,2);
    else
        %nothing close
        FObjects_R3O1 = [0;0];
    end
%R3-O2
    if(~isempty(iInfluencial_R3O2))
        %vector sum of repulsions:
        rho_R3O2 = repmat(Distance_R3O2(iInfluencial_R3O2),2,1);
        V_R3O2 = Dp_R3O2(:,iInfluencial_R3O2);
        DrhoDx_R3O2 = -V_R3O2./rho_R3O2;
        F_R3O2 = (1./rho_R3O2-
```

```
1./RadiusOfInfluence_dynamic)*1./(rho_R3O2.^2).*DrhoDx_R3O2;
        FObjects_R3O2 = Kdynamic*sum(F_R3O2,2);
    else
        %nothing close
        FObjects_R3O2 = [0;0];
    end
%R3-O3
    if(~isempty(iInfluencial_R3O3))
        %vector sum of repulsions:
        rho_R3O3 = repmat(Distance_R3O3(iInfluencial_R3O3),2,1);
        V_R3O3 = Dp_R3O3(:,iInfluencial_R3O3);
        DrhoDx_R3O3 = -V_R3O3./rho_R3O3;
        F_R3O3 = (1./rho_R3O3-
1./RadiusOfInfluence_dynamic)*1./(rho_R3O3.^2).*DrhoDx_R3O3;
        FObjects_R3O3 = Kdynamic*sum(F_R3O3,2);
    else
        %nothing close
        FObjects_R3O3 = [0;0];
    end


    FGoal_1 = transpose(KGoal*(Goal_Error1)/norm(Goal_Error1));
    FGoal_2 = transpose(KGoal*(Goal_Error2)/norm(Goal_Error2));
    FGoal_3 = transpose(KGoal*(Goal_Error3)/norm(Goal_Error3));

    FObjects_1=FObjects_12+FObjects_13;
    FObjects_2=FObjects_21+FObjects_23;
    FObjects_3=FObjects_31+FObjects_32;

    FObjects_dynamic_1=FObjects_R1O1+FObjects_R1O2+FObjects_R1O3;
    FObjects_dynamic_2=FObjects_R2O1+FObjects_R2O2+FObjects_R2O3;
    FObjects_dynamic_3=FObjects_R3O1+FObjects_R3O2+FObjects_R3O3;


    FTotal_1 =
KTotal*(FGoal_1+FObjects_1+FObjects1Env+FObjects_dynamic_1);
    FTotal_2 =
KTotal*(FGoal_2+FObjects_2+FObjects2Env+FObjects_dynamic_2);
    FTotal_3 =
KTotal*(FGoal_3+FObjects_3+FObjects3Env+FObjects_dynamic_3);


    Magnitude_1 = min(1,norm(FTotal_1));
    Magnitude_2 = min(1,norm(FTotal_2));
    Magnitude_3 = min(1,norm(FTotal_3));

    FTotal_1 = FTotal_1/norm(FTotal_1)*Magnitude_1;
    FTotal_2 = FTotal_2/norm(FTotal_2)*Magnitude_2;
    FTotal_3 = FTotal_3/norm(FTotal_3)*Magnitude_3;


    Magnitude_4 = min(1,norm(FObjects_1));
    Magnitude_5 = min(1,norm(FObjects_2));
    Magnitude_6 = min(1,norm(FObjects_3));
```

```
    FObjects_1 = FObjects_1/norm(FObjects_1)*Magnitude_4;
    FObjects_2 = FObjects_2/norm(FObjects_2)*Magnitude_5;
    FObjects_3 = FObjects_3/norm(FObjects_3)*Magnitude_6;



    ranking_script

    act_robots;

    update_robots;
 %-------------------------------------------------------------------------
---


%ranking_script.m

followerscript=6;%6
leaderscript=2;%2
defaultscript=2;%2
dynamicscript=7;
stopscript=0;


%Whichever robot is closer to its goal gets priority
if distanceToGoal>distanceToGoal2
    %r1 is closer to goal, r1 gets priority on r2
    r12rank=0.1;
elseif distanceToGoal<distanceToGoal2
    r12rank=1;
end


if distanceToGoal>distanceToGoal3
    %r1 is closer to goal, r1 gets priority on r3
    r13rank=0.1;
elseif distanceToGoal<distanceToGoal3
    r13rank=1;
end



if distanceToGoal2>distanceToGoal3
    %r2 is closer to goal, r2 gets priority on r3
    r23rank=0.1;
elseif distanceToGoal2<distanceToGoal3
    r23rank=1;
end
```

```
 if(~isempty(iInfluencial_12))
   %R1-R2
    if r12rank<=0.5
        %r1 is leader to r2
        %'Robot 1 is leader to R2'

              r2action=followerscript;
              r1action=leaderscript;
              %r3action=defaultscript;

              actrobot2;
              actrobot1;

              %r2action=testscript;
              %actrobot2;


    elseif r12rank>0.5
     %r2 is leader to r1
    % 'Robot 2 is leader to R1'


              r1action=followerscript;
              r2action=leaderscript;
              %r3action=defaultscript;

              actrobot1;
              actrobot2;

              %r1action=testscript;
              %actrobot1;



        end
    end

%R1-R3

if (~isempty(iInfluencial_13))
    if r13rank<=0.5
    %r1 is leader to r3
    %'Robot 1 is leader to R3'


              r3action=followerscript;
              r1action=leaderscript;
              %r2action=defaultscript;

              actrobot3;
              actrobot1;
```

```
            %actrobot2;


            %r3action=testscript;
            %actrobot3;



     %   'Robot1 Has Priority, Robot3 dodges'
    elseif r13rank>0.5
    %r3 is leader to r1
    %'Robot 3 is leader to R1'



            r1action=followerscript;
            %r2action=defaultscript;
            r3action=leaderscript;

            actrobot1;
            actrobot3;


            %actrobot2;

            %r1action=testscript;
            %actrobot1;


    end
end


if(~isempty(iInfluencial_23) )
    if r23rank<=0.5

   %r2 is leader to r3
   % 'Robot 2 is leader to R3'

           r3action=followerscript;
           %r1action=defaultscript;
            r2action=leaderscript;

            actrobot3;
            actrobot2;


           % actrobot1;

            %r3action=testscript;
            %actrobot3;
```

```
    elseif r23rank>0.5
        %r3 is leader to r2
     %   'Robot 3 is leader to R2'


                %r1action=defaultscript;
                r2action=followerscript;
                r3action=leaderscript;
                %'HELLO'

                actrobot2;
                actrobot3;

                %actrobot1;

                %r2action=testscript;
                %actrobot2;

   %     'Robot3 Has Priority, Robot2 dodges'
        end
end




if(~isempty(iInfluencial_R1O1))||(~isempty(iInfluencial_R1O2))||(~isempt
y(iInfluencial_R1O3))

                    %r1action=followerscript;
                    %actrobot1;
                    %updaterobot1;

                    r1action=stopscript;
                    actrobot1;
                    updaterobot1;


                    r1action=dynamicscript;
                    actrobot1;
                    updaterobot1;









else
```

```
                    r1action=defaultscript;
                    actrobot1;
                    updaterobot1;
end


if(~isempty(iInfluencial_R2O1))||(~isempty(iInfluencial_R2O2))||(~isempt
y(iInfluencial_R2O3))


                    %r2action=followerscript;
                    %actrobot2;
                    %updaterobot2;

                    %r2action=followerscript;
                    %actrobot2;
                    %updaterobot2;

                    r2action=stopscript;
                    actrobot2;
                    updaterobot2;

                    r2action=dynamicscript;
                    actrobot2;
                    updaterobot2;




else
                    r2action=defaultscript;
                    actrobot2;
                    updaterobot2;


end



if(~isempty(iInfluencial_R3O1))||(~isempty(iInfluencial_R3O2))||(~isempt
y(iInfluencial_R3O3))


                    r3action=stopscript;
                    actrobot3;
                    updaterobot3;


                    r3action=dynamicscript;
```

```
                    actrobot3;
                    updaterobot3;




else

                    r3action=defaultscript;
                    actrobot3;
                    updaterobot3;


end

update_robots;
%-------------------------------------------------------------------------
--
%act_robots.m
actrobot1;
actrobot2;
actrobot3;
%-------------------------------------------------------------------------
--
%actrobot1.m
%{
 r#action- if r#action is 0, robot is to return to previous location
if r#action is 1, robot is to go to forward location
%}

if r1action==0
    %r1action 0
   controller1.DesiredLinearVelocity=0.0000000000000000001;
end

if r1action==1
    %r1action 1
   controller1.DesiredLinearVelocity=1.5;
   updaterobot1;

end

if r1action==2
    %r1action 2
       controller1.DesiredLinearVelocity=1.5;

controller1.DesiredLinearVelocity=1.0*controller1.DesiredLinearVelocity;
       xRobot1=xRobot1+FTotal_1;
       Theta1 = -atan2(FTotal_1(1),FTotal_1(2));
       updaterobot1;
end

if r1action==3
```

```
    %r1action 3
    xRobot1=xRobot1+FTotal_1;
    Theta1 = -atan2(FTotal_1(1),FTotal_1(2));
    updaterobot1;

end

if r1action==4
    %r1action 4
    xRobot1=xRobot1+FTotal_1;
    Theta1 = -atan2(FTotal_1(1),FTotal_1(2));
      updaterobot1;
end

if r1action==5
    %r1action 5
    xRobot1=xRobot1_previous+FTotal_1;
    Theta1 = -atan2(FTotal_1(1),FTotal_1(2));
      updaterobot1;
end

if r1action==6
    %r1action 6
   controller1.DesiredLinearVelocity=0.0000000000000000001;
    xRobot1_previous=transpose(path_r1(Q1-mem11,:));
    Theta1 = -atan2(FObjects_1(1),FObjects_2(2));
    xRobot1=xRobot1_previous;
    updaterobot1;

    if Q1>=mem11+5
        mem11=mem11+1

        if mem11>=5
            mem11=1;
        end
    end
end
if r1action==7
    %r1action 7
   controller1.DesiredLinearVelocity=0.0000000000000000001;
    xRobot1_previous=transpose(path_r1(Q1-mem11,:));
    Theta1 = -atan2(FObjects_dynamic_1(1),FObjects_dynamic_1(2));
    xRobot1=xRobot1_previous;
    updaterobot1;

    if Q11>=mem11+5
        mem11=mem11+1;

        if mem11>=5
            mem11=1;
        end
    end

end
```

```
updaterobot1;
%-------------------------------------------------------------------------
--
%actrobot2.m
%{
 r#action- if r#action is 0, robot is to return to previous location
if r#action is 1, robot is to go to forward location
%}

if r2action==0
    %r2action 0
   controller2.DesiredLinearVelocity=0.0000000000000000001;

end

if r2action==1
    %r2action 1
   controller2.DesiredLinearVelocity=1.5;
   updaterobot2;
end

if r2action==2
    %r2action 2
       controller2.DesiredLinearVelocity=1.5;

controller2.DesiredLinearVelocity=1.0*controller2.DesiredLinearVelocity;
       xRobot2=xRobot2+FTotal_2;
       Theta2 = -atan2(FTotal_2(1),FTotal_2(2));
       updaterobot2;
end

if r2action==3
    %r2action 3
    xRobot2=xRobot2+FTotal_2;
    Theta2 = -atan2(FTotal_2(1),FTotal_2(2));
end

if r2action==4
    %r2action 4
    xRobot2=xRobot2+FTotal_2;
    Theta2 = -atan2(FTotal_2(1),FTotal_2(2));
      updaterobot2;
end

if r2action==5
    %r2action 5
    xRobot2=xRobot2_previous+FTotal_2;
    Theta2 = -atan2(FTotal_2(1),FTotal_2(2));
      updaterobot2;
end

if r2action==6
    %r2action 2
   controller2.DesiredLinearVelocity=0.0000000000000000001;
```

```
    xRobot2_previous=transpose(path_r2(Q2-mem22,:));
    Theta2 = -atan2(FObjects_2(1),FObjects_2(2));
    xRobot2=xRobot2_previous;
    updaterobot2;

    if Q2>=mem22+5
        mem22=mem22+1

        if mem22>=5
            mem22=1;
        end
    end

   %updaterobot2;
   %return;

end
if r2action==7                      %r2action 2 is stop, backup and dodge
   controller2.DesiredLinearVelocity=0.0000000000000000001;
    xRobot2_previous=transpose(path_r2(Q2-mem22,:));
    %Theta2 = -atan2(FObjects_dynamic_2(1),FObjects_dynamic_1(2));
    xRobot2=xRobot2_previous;
    updaterobot2;

    if Q22>=mem22+5
        mem22=mem22+1;

       if mem22>=5
            mem22=1;
        end
    end
end


updaterobot2;
%-------------------------------------------------------------------------
--
%actrobot3.m
%{
 r#action- if r#action is 0, robot is to return to previous location
if r#action is 1, robot is to go to forward location
%}

if r3action==0
    %r3action 0
   controller3.DesiredLinearVelocity=0.0000000000000000001;
end

if r3action==1
    %r3action 1
   controller3.DesiredLinearVelocity=1.5;
   updaterobot3;
end

if r3action==2
```

```
    %r3action 2
       controller3.DesiredLinearVelocity=1.5;

controller3.DesiredLinearVelocity=1.0*controller3.DesiredLinearVelocity;
    xRobot3=xRobot3+FTotal_3;
    Theta3 = -atan2(FTotal_3(1),FTotal_3(2));
      updaterobot3;
end

if r3action==3
    %r3action 3
    xRobot3=xRobot3+FTotal_3;
    Theta3 = -atan2(FTotal_3(1),FTotal_3(2));
    updaterobot3;
end

if r3action==4
    %r3action 4
    xRobot3=xRobot3+FTotal_3;
    Theta3 = -atan2(FTotal_3(1),FTotal_3(2));
      updaterobot3;
end

if r3action==5
    %r3action 5
    xRobot3=xRobot3_previous+FTotal_3;
    Theta3 = -atan2(FTotal_3(1),FTotal_3(2));
      updaterobot3;
end

if r3action==6
    %r3action 2
   controller3.DesiredLinearVelocity=0.00000000000000000001;
    xRobot3_previous=transpose(path_r3(Q3-mem33,:));
    Theta3 = -atan2(FObjects_3(1),FObjects_3(2));
    xRobot3=xRobot3_previous;
    updaterobot3;

    if Q3>=mem33+5
        mem33=mem33+1;

        if mem33>=5
            mem33=1;
        end
    end

end
if r3action==7
    %r3action 2
   controller3.DesiredLinearVelocity=0.00000000000000000001;
    xRobot3_previous=transpose(path_r3(Q3-mem33,:));
    %Theta3 = -atan2(FObjects_dynamic_3(1),FObjects_dynamic_3(2));
    xRobot3=xRobot3_previous;
    updaterobot3;
```

```
    if Q33>=mem33+5
        mem33=mem33+1

        if mem33>=5
            mem33=1;
        end
    end
end


updaterobot3;



%-------------------------------------------------------------------------
--
%updaterobots.m
updaterobot1;
updaterobot2;
updaterobot3;
%-------------------------------------------------------------------------
--
%updaterobot1.m

r1action;

  if r1action==0
        controller1.DesiredLinearVelocity=0.0000000000000000001;
  end

if r1action==6 || r1action==0 || r1action==7
        controller1.DesiredLinearVelocity=0.0000000000000000001;

        r1CurrentLocation=transpose(xRobot1);%R1
        robot1.CurrentPose(1)=r1CurrentLocation(1);
        robot1.CurrentPose(2)=r1CurrentLocation(2);
        robot1.CurrentPose(3)=Theta1;
end

if r1action==7
        controller1.DesiredLinearVelocity=0.0000000000000000001;

        r1CurrentLocation=transpose(xRobot1);%R1
        robot1.CurrentPose(1)=r1CurrentLocation(1);
        robot1.CurrentPose(2)=r1CurrentLocation(2);
        %robot1.CurrentPose(3)=Theta1;
end

if r1action~=0 && r1action~=6 && r1action~=4&& r1action~=7
        controller1.DesiredLinearVelocity=1.5;

controller1.DesiredLinearVelocity=1.5*controller1.DesiredLinearVelocity;
end
```

```
        [v1, omega1] = step(controller1, [transpose(xRobot1)
robot1.CurrentPose(3)]);









    %robot1.CurrentPose=[transpose(xRobot1) Theta1]


%-------------------------------------------------------------------------
--
%updaterobot2.m
Distance_R2O1;
r2action;
    if r2action==0
        controller2.DesiredLinearVelocity=0.0000000000000000001;

    end
    if r2action==6 || r2action==3 || r2action==7
        controller2.DesiredLinearVelocity=0.0000000000000000001;
        r2CurrentLocation=transpose(xRobot2);%r2
        robot2.CurrentPose(1)=r2CurrentLocation(1);
        robot2.CurrentPose(2)=r2CurrentLocation(2);
        robot2.CurrentPose(3)=Theta2;

    end

    if r2action==7
        controller2.DesiredLinearVelocity=0.0000000000000000001;
        r2CurrentLocation=transpose(xRobot2);%r2
        robot2.CurrentPose(1)=r2CurrentLocation(1);
        robot2.CurrentPose(2)=r2CurrentLocation(2);
        %robot2.CurrentPose(3)=Theta2;

    end

    if r2action~=0 && r2action~=6 && r2action~=3&& r2action~=7
        controller2.DesiredLinearVelocity=1.5;

controller2.DesiredLinearVelocity=1.5*controller2.DesiredLinearVelocity;
    end
            [v2, omega2] = step(controller2, [transpose(xRobot2)
robot2.CurrentPose(3)]);


%-------------------------------------------------------------------------
--
%updaterobot3.m
```

```
Distance_R3O1;
r3action;
    if r3action==0
        controller3.DesiredLinearVelocity=0.0000000000000000001;


    end
    if r3action==6 || r3action==3 || r3action==7
        controller3.DesiredLinearVelocity=0.0000000000000000001;
        r3CurrentLocation=transpose(xRobot3);%r3
        robot3.CurrentPose(1)=r3CurrentLocation(1);
        robot3.CurrentPose(2)=r3CurrentLocation(2);
        robot3.CurrentPose(3)=Theta3;

    end

     if r3action==7
        controller3.DesiredLinearVelocity=0.0000000000000000001;
        r3CurrentLocation=transpose(xRobot3);%r3
        robot3.CurrentPose(1)=r3CurrentLocation(1);
        robot3.CurrentPose(2)=r3CurrentLocation(2);
        %robot3.CurrentPose(3)=Theta3;

    end

    if r3action~=0 && r3action~=6 && r3action~=3 && r3action~=7
        controller3.DesiredLinearVelocity=1.5;

controller3.DesiredLinearVelocity=1.5*controller3.DesiredLinearVelocity;
    end
            [v3, omega3] = step(controller3, [transpose(xRobot3)
robot3.CurrentPose(3)]);

%-------------------------------------------------------------------------
--
```