MOTION-PLANNING USING RRTS FOR A SWARM OF ROBOTS CONTROLLED BY GLOBAL INPUTS

A Thesis

Presented to

the Faculty of the Department of Electrical and Computer Engineering

University of Houston

in Partial Fulfillment

of the Requirements for the Degree

Master of Science

in Electrical Engineering

by
Parth Joshi
December 2018

MOTION-PLANNING USING RRTS FOR A SWARM OF ROBOTS CONTROLLED BY GLOBAL INPUTS

Parth Joshi

Approved:

Chair of the Committee
Dr. Aaron T. Becker, Assistant Professor
Department of Electrical Engineering

Committee Members:

Dr. Rose Faghih, Assistant Professor
Department of Electrical Engineering

Dr. Nikolaos Tsekos, Professor
Department of Computer Science

Dr. Julien Leclerc, Postdoctoral Fellow
Department of Electrical Engineering

Dr. Suresh K. Khator, Associate Dean,
Cullen College of Engineering

Dr. Badrinath Roysam, Professor and Chair,
Electrical and Computer Engineering

Acknowledgements

I thank my thesis advisor Dr. Aaron T. Becker for his constant support and invaluable guidance. He has set an example of excellence as a mentor and an instructor.

I am grateful to my committee members, Dr. Rose Faghih, Dr. Nikolaos Tsekos and Dr. Julien Leclerc, for agreeing to serve as my master thesis committee and spending their valuable time and feedback on my work.

I thank all my labmates at the Swarm Robotics lab for all the great time that we spent together. Topics presented by my lab mates during weekly meeting helped me develop many ideas for my work. I thank my friends Meet Shah, Jonathan Wang and Mohammad Sultan for their ideas, support, and guidance.

I am grateful to my family for their unconditional love and support during all these years. I would like to thank my family: My parents Jaya Joshi and Mukesh Joshi. I want to especially thank my sister Jinal Joshi who never stopped encouraging me, helped me achieve my goals, and motivated me throughout these years.

MOTION-PLANNING USING RRTS FOR A SWARM OF ROBOTS CONTROLLED BY GLOBAL INPUTS

An Abstract

of a

Thesis

Presented to

the Faculty of the Department of Electrical and Computer Engineering

University of Houston

in Partial Fulfillment

of the Requirements for the Degree

Master of Science

in Electrical Engineering

by
Parth Joshi
December 2018

Abstract

Small-scale robots have great potential to bring transformation in the field of medical applications, defense systems, security, micro-assembly and many other areas. Imagine a group of milli, micro or nano-robots that can navigate inside the body to solve medical problems, or a swarm of robots that can paint a beautiful painting. Robots containing iron can be steered using a magnetic field generated by an MRI scanner, but all the robots will move in the same direction because the magnetic effect is global and not local. This thesis uses a customized version of the motion-planning technique called Rapidly Exploring Random Tree (RRT) to solve this problem for multiple robots by using obstacles. This thesis project solves instances of this motion-planning problem for more than one robot when steered using a magnetic field, and develops a motion planner that searches for sequences of control inputs to simulations steer the robots to desired goal positions.

Table of Contents

List of Figures

Chapter 1

Introduction

Small-scale robots [4] have great potential to bring transformation in the field of medical applications [5], defense systems [6], security, micro-assembly [7, 8] and many other areas. Imagine a group of milli, micro or nanorobots which can navigate inside the body to solve medical problems or a swarm of robots that can paint a beautiful painting. Robots containing iron can be steered using magnetic field generated by MRI scanner [9, 10, 11], but all the robots will move in the same direction because the magnetic effect is global and not local. Many important problems require robots to go to different locations, which is hard because the control input moves them in the same direction. This thesis uses a customized version of the motion-planning technique called Rapidly Exploring Random Tree (RRT) [12] to solve this problem for multiple robots by using obstacles. This thesis project solves instances of this motion-planning problem for more than one milli robot when steered using a magnetic field and develops a motion planner that searches for sequences of control inputs to steer the robots to different goal positions.

1.1 Problem Definition

This thesis investigates particle-like robots which are all moved by the same force. Each robot will move together until its movement is obstructed by any obstacles in the simulation environment created by us. The problem is trivial if there is only one robot, but as the number of the robots increases the problem gets difficult because the combination of goals might require robots to move in different directions. In another

perspective, each robot acts as a constraint for all others robots, so with the increase in the number of robots, constraints increases too. In this thesis, we will perform most of most experiments using two particles but the same process applies to larger numbers of particles. We demonstrate the results with multiple robots in Chapter 4.

Our problem is a motion-planning problem for a swarm of robots. Our robots are dumb, and we simulate as if they are controlled by a strong magnetic field device like an MRI scanner [13]. Under a magnetic field, all the robots will move in the same direction under global inputs [3, 14]. Though the swarm of robots moves under global inputs, the problem is to make all the robots reach their individual goals. My thesis will propose a solution to this problem. In Figure 1.1, the obstacle is shown in white color and the robots are shown as white circles and discs. All robots move under global inputs and both robots cannot reach their goals (green circles) if the starting positions of both the robot has different vector-difference compared to the goals. In this case only one robot can reach its goal at one time. What we seek is to find a tool to solve problems where multiple robots can reach their individual goal positions under global inputs. Along with that, we want to know which parameters affect the coverage of maps [15]. Finally, we want to reuse the RRT exploration to find a solution for multiple sets of goals.

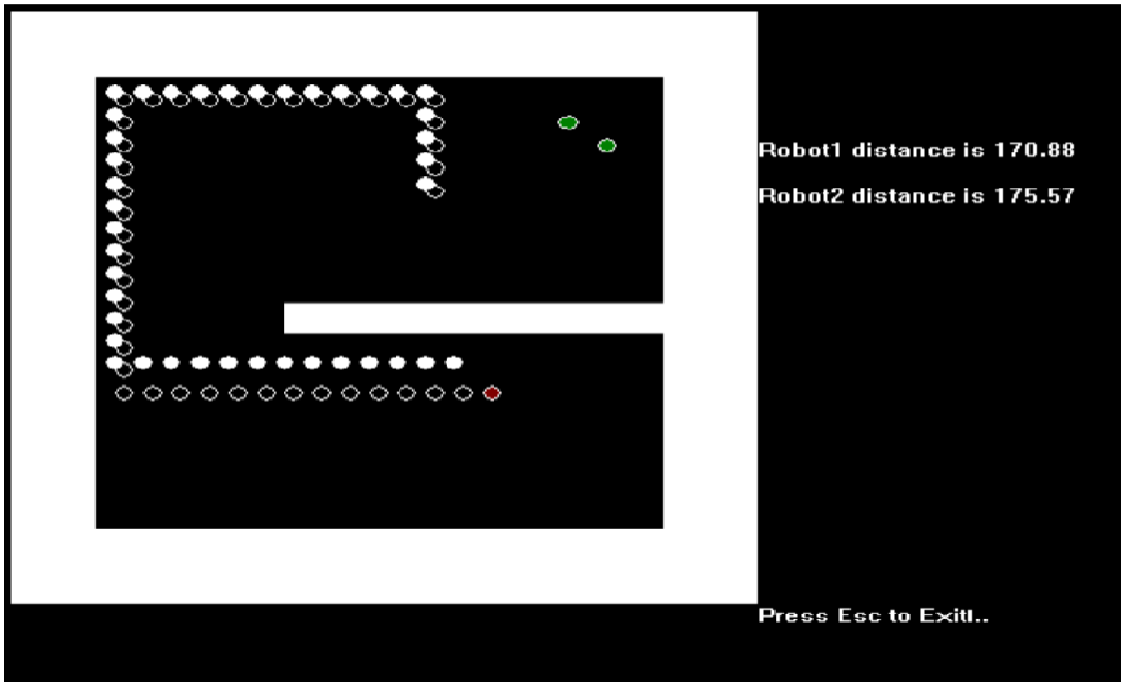


Figure 1.1: Robots moving using global inputs.

To explore the effects of parameter variation, we move robots from start to goal in different environments and calculate how well our robots explored the space. The exploration of the space can be computed by dividing the whole space into smaller grid cells and checking which fractions of the grid cells have been explored simultaneously by both robots. Using this method, we check coverage of the map by robots. Once the space then is explored, we change the goals of the robots, reusing the possible paths we generated. We then apply the same for more numbers of robots to understand the effect of increasing the number of the robots.

1.1.1 Configuration Space

If the robot has n degrees of freedom, the set of transformations is usually a manifold of dimension n. This manifold [16] is called the *configuration space* of the robot, and its name is often shortened to C-space [17]. The C-space can be considered as a special state space. To solve a motion-planning problem, algorithms must conduct

a search in the C-space. The C-space provides a powerful abstraction that converts the complicated models and transformations of robot configurations into the general problem of computing a path that traverses a manifold. By developing algorithms directly for this purpose, they apply to a wide variety of different kinds of robots and transformations. Our problem will be addressed by representing obstacles in the configuration space. These obstacles are called *configuration space obstacles* [3]. An example showing a robots workspace and configuration space is Figure 1.2

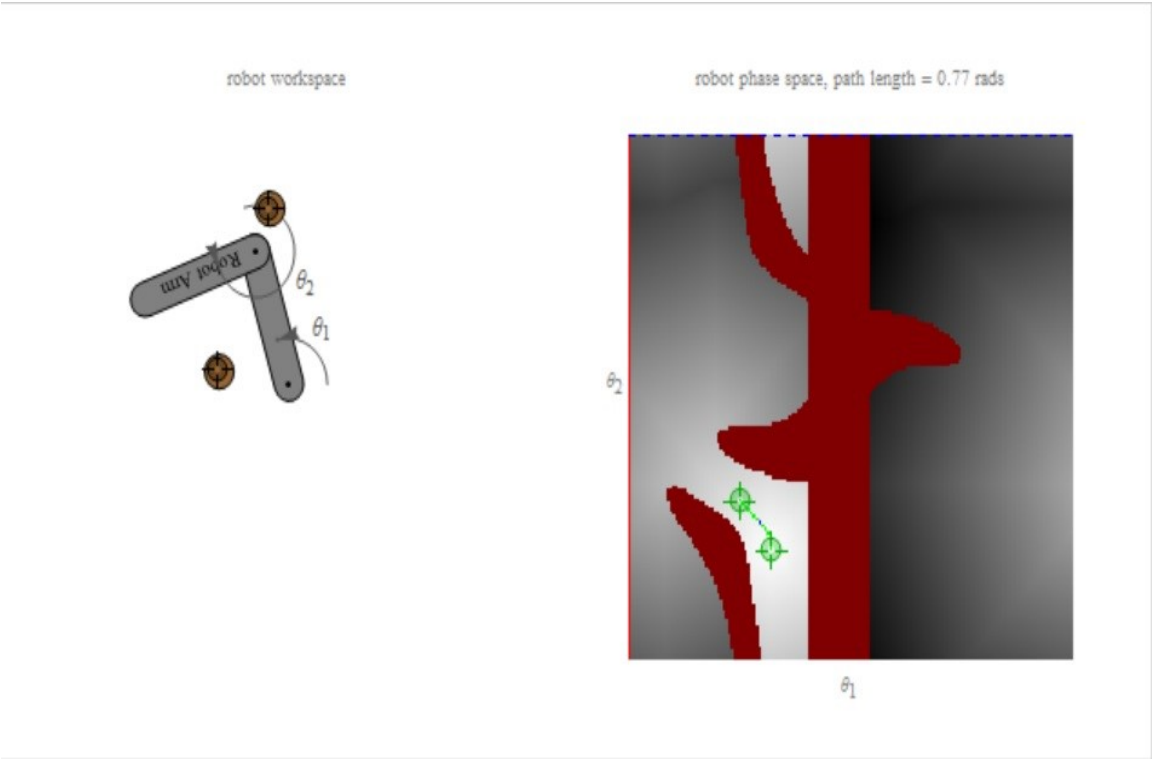


Figure 1.2: A 2-link robot with two circular obstacle (left) and its configuration space (right). Wolfram demonstration created by Dr. Aaron Becker for class *Intro to Robotics* at the University of Houston.

1.1.2 $2D$ Rigid Bodies

$2D$ Rigid bodies: Since any $x_t, y_t \in \mathbb{R}^2$ can be selected for translation, this alone yields a manifold $M1 = \mathbb{R}^2$. Independently, any rotation, $[0, 2\pi)$, can be applied. Since 2π yields the same rotation as 0, they are identical, which makes the set of

$2D$ rotations into a manifold, $M2 = S^1$. To obtain the manifold that corresponds to all rigid-body motions, simply take $C = M1 \times M2 = \mathbb{R}^2 \times S^1$. The C-space is 3 dimensional, but the dimension for orientation wraps around from 2π to 0.

1.1.3 Workspace of a Robot

The *workspace* [16] of a robot can be defined as all the space reachable by the end effector of the robot. The total volume that can be swept out by the end effector will define a *workspace*. The mechanical joints and geometry of the robot and the environment limitations bound the *workspace*. For example, a revolute joint restricted to less than 2π cannot complete a 360° rotation. The *workspace* can be divided further into the the *reachable workspace* and a *dexterous workspace*. The reachable *workspace* [16] is the all the points reachable by manipulators while the dexterous *workspace* is the part of the reachable *workspace* which reachable by the by end effector with any orientation.

1.2 Definitions

1.2.1 Robot

The word *robot* was introduced in 1921 by the Czech playwright Karel Capek in his satirical play R. U. R. (Rossums Universal Robots), where he depicted robots as machines which resembled people but worked tirelessly [12]. The Oxford English Dictionary gives the following definition: [A machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer]. This thesis defines robots as particles that can move in $2D$ directions.

1.2.2 Homeomorphism

If shape 1 can stretch or bend to obtain a shape 2 they are called *homeomorphic* and the property is called *homeomorphism*. A famous example is of a cup and a donut. A cup and a donut are topologically the same as there is a continuous transformation between the two. So, they are called homeomorphic. However, a sphere and a donut are not *homeomorphic* because the donut has a hole.

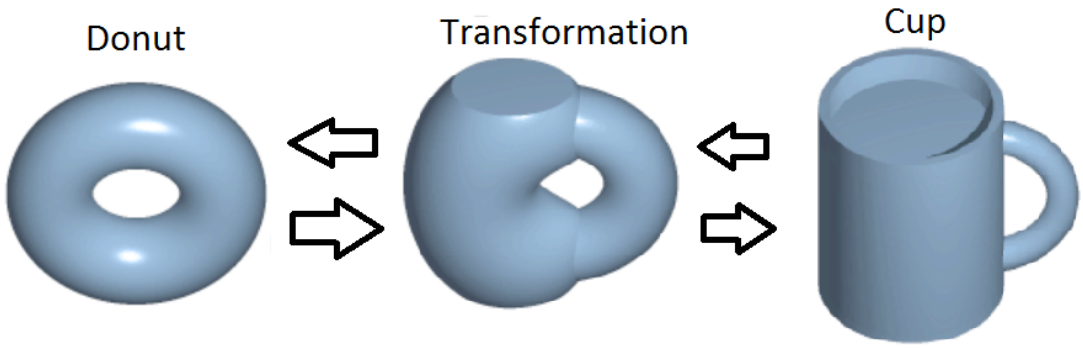


Figure 1.3: Explanation of the homeomorphism.
`https://en.wikipedia.org/wiki/Homeomorphism`

1.2.3 Manifold

To reflect a true structure of the space of transformations it helps to understand manifolds. According to the textbook on robotics by S. M. LaValle, a topological space $M \subset \mathrm{R}^m$ is a manifold if for every $x \in M$, an open set $O \subset M$ exists such that: (1) $x \in O$, (2) O is *homeomorphic* to R^n, and (3) n is fixed for all $x \in M$. The fixed n is referred to as the dimension of the manifold M. The second condition is the most important. It states that in the vicinity of any point, $x \in M$ the space behaves just like it would in the vicinity of any point ($y \in R^n$) intuitively, the set of directions that one can move appears the same in either case [12].

1.2.4 Cartesian Products

Suppose A and B are topological spaces. Then the new topological space is given by the Cartesian product of A and B, i.e. $A \times B$ such that all the point in A and all the points in B generates points in the Cartesian product of A and B.

1.2.5 1-D Manifolds

A *manifold* is a topological space that locally has features similar to Euclidean space near each point. Lines and circles are examples of $1D$ manifolds, but not shapes with crossing points such as figure eights (shape of 8)

1.2.6 2-D Manifolds

$2D$ manifolds are basically outcomes of the Cartesian products of two $1D$ manifolds. They are surfaces. The plane sphere and the torus are the examples of a $2D$ manifolds.

1.2.7 Particle

The configuration space of a particle can be given as R^3, if it can move in X, Y and Z Cartesian coordinates in a Euclidean space.

1.2.8 The Degrees of Freedom (DOF)

If the configuration of any robot or group of robots (in our case) is defined by n parameters, in robotics *the degrees of freedom* are the modes in which the system can move or the independent movements possible at each joint of the robot. There are three translation movements (X, Y and Z-direction) possible in the real world and three rotational motions are possible (Roll, Pitch, and Yaw).

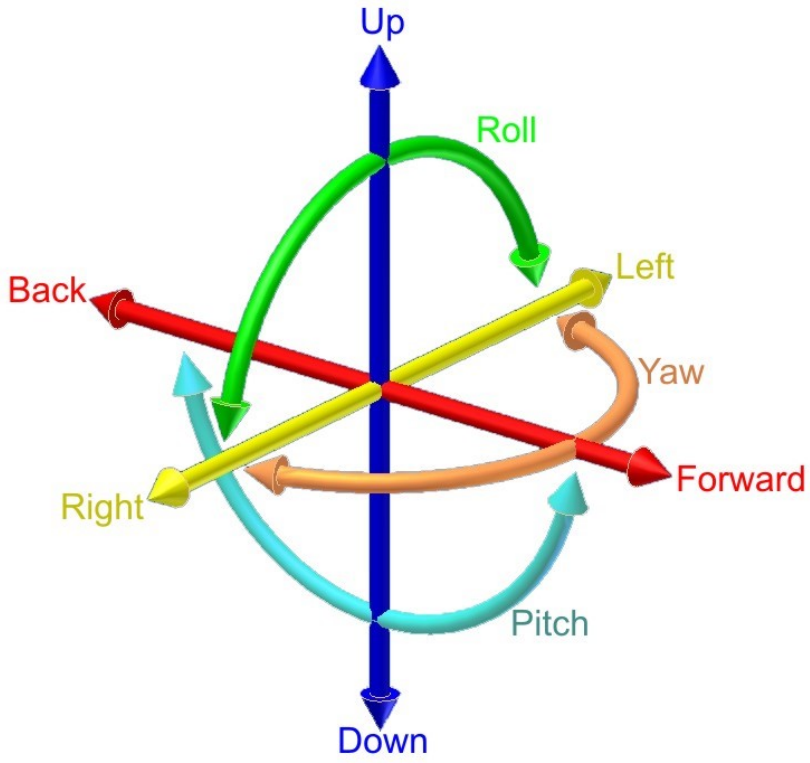


Figure 1.4: 6 DOF for the rigid body: translation (up, down and back) and rotation movement (roll, pitch, and yaw), `https://en.wikipedia.org/wiki/File:6DOF_en.jpg`.

1.2.9 Rigid Body

A rigid body is a solid body in which there is minimal or zero deformation. A rigid body has six degrees of freedom.

1.3 Coupled and Decoupled approach

1.3.1 Centralized Multi-robot Coordination

Steering multiple robots in any environment is a complex problem of motion-planning. One solution is to compute multiple robots as a single robot with each robot as its degree of freedom. Basically, we will plan for all the robots simultaneously by treating all of the robots as a single composite system with n degrees of freedom. This

is called centralized multi-robot planning. If we have n robots then the dimension of the composite robot is n (n degrees of freedom) and the c-space is also n dimensional.

1.3.2 Decentralized Multi-robot Coordination

As the number of robots increases, the dimension of the state space increases (due to increase of the degrees of freedom). Path-planning on multiple robots simultaneously is computationally a challenge. This is often called "the curse of dimensionality". The solution to this problem can be given by a decentralized plan-planning approach. In a decentralized robot, each robot in the swarm is treated as an individual robot rather than considering all the robots as one composite robot. This is not possible when all the robots are steered simultaneously by a shared control input. The problem in this thesis is under-actuated.

1.4 Difficulties in motion-planning with More Than One Robot

1.4.1 The Dimension of a Swarm of Robots

As the number of the robots in the swarm increases the dimensions of the robot increases. The the degrees of freedom for each robot add to create a composite robot. For two robots in the $2D$ space, the dimension of the configuration space is 4D (X and Y coordinate of each robot) while for 8 robots who can move in X and Y there are 16 dimensions in the configuration space.

1.4.2 Curse of Dimensionality

The centralized multi-robot concept [16], considers multiple robots as one composite robot. As the number of robots increases the configuration space dimension increases and the data required to find the solution of such problems increases expo-

nentially. In other words, the volume of the space increases so fast that it becomes difficult and computationally challenging to search the space and find a solution.

1.5 Proposed Method

Using the C++ language and OMPL libraries, we created a simulations of swarms of particles in different environments. We define start and goal for each robot in the swarm in a specific environment. The aim of the robots is to reach their individual goals despite obstacles by using the obstacles to adjust differences in position between swarm members. We use a motion-planning algorithm to find a solution, in which the particles perform a random walk. We simulate different paths and measure coverage of the map to check the performance of the motion-planning algorithms. We created five different environments in C++ to see how our robots will perform in these environments and drove our conclusion.

There are several assumptions in this work. The robots are treated as points (zero radius), are not affected by friction other than collisions with the obstacles, and move with constant velocity.

1.6 Environment Description

All the environments we use have a workspace of 400×400 units, which means the robots are bounded and cannot cross the limit of 400 units. Along with that, the environment has boundary obstacles that are 50 units thick on each side, as shown in Figure 1.5. The available workspace for the robot is 300×300 units less the obstacles inside the workspace. The configuration space has dimension $300 \times 300 \times 300 \times 300$ for 2 robots. The dimension of the configuration space rises to the $2n$ dimension where n is the number of robots.

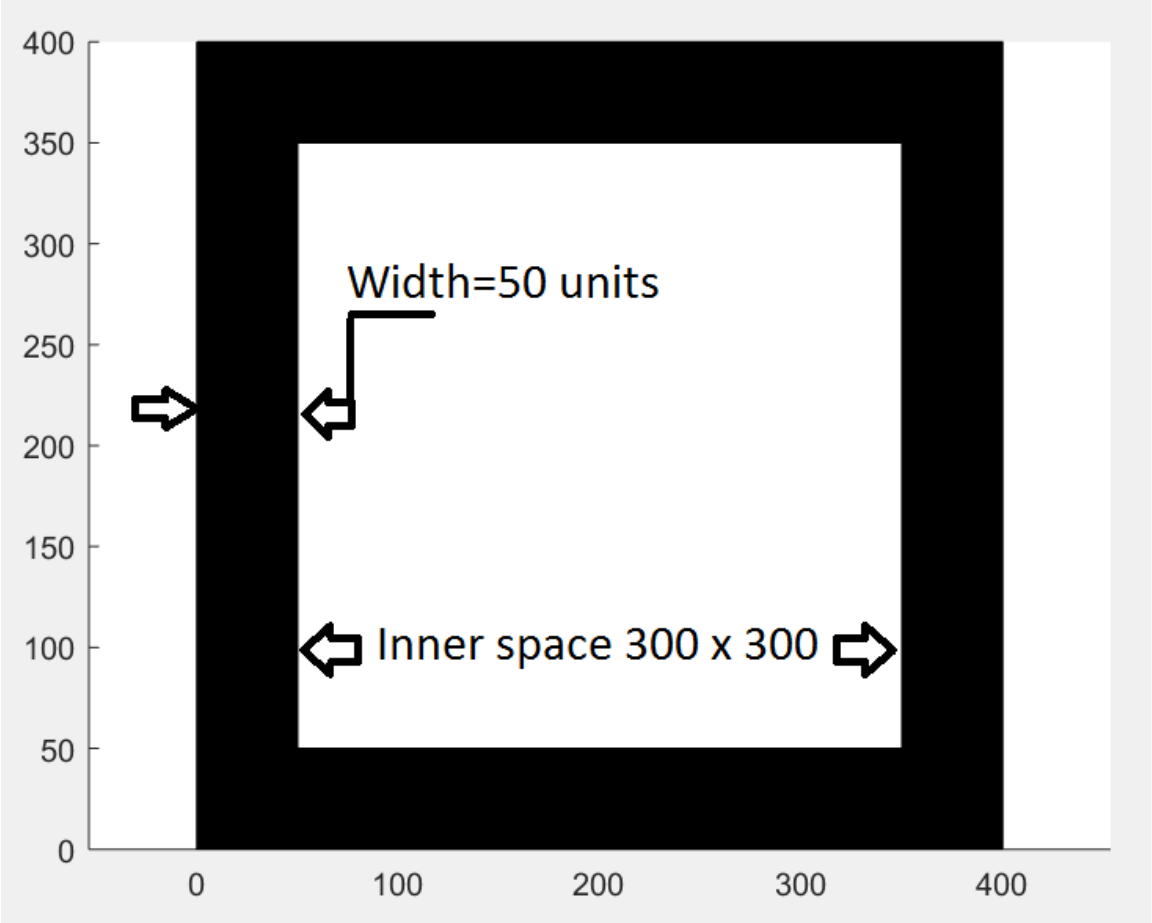


Figure 1.5: Environment prototype and available inner space.

Chapter 2

Theory

This thesis provides a *probabilistically complete* algorithm for the problem of motion-planning for a swarm of robots. This means that if a solution exists, we will find it with probability 1, as the size of our exploration tree grows to infinite size. We simulate the approach using the open source motion-planning package OMPL. This section will discuss different tools for motion-planning and the algorithms used in this thesis. We will also discuss the structure of the OMPL (Open Motion-Planning Library) and how we have used the OMPL package for our thesis. If the step size is fixed then the algorithm is not probabilistically complete but it is *resolution complete*, meaning that a solution will only be found if one exists for this step size. However, if the step size is determined by the collision checking the algorithm is *probabilistically complete*.

2.1 Motion-Planning

Motion-planning [16] of the robot includes planning a collision-free path. We have the initial and the final configuration of the robots and the problem is to plan a path for the robot that avoids the robot passing through any obstacles space. However, our algorithm uses obstacles to achieve its goal. Planning paths and trajectories are considered some of the most difficult problems in computer science. An algorithm which finds a solution whenever one exists is called a *complete algorithm*. However, the computational complexity of complete algorithms increases exponentially with the number of degrees of freedom of the robot. In this thesis, we use an algorithm

which is a so-called *probabilistically complete* algorithm. This algorithm finds the solution if there is one but does not determine if there is no solution.

2.1.1 Sampling-based Planner [1]

The configuration space (C-space) has infinite resolution, since each degree of freedom is a real number. Therefore, a planner which can sample the space in a finitely large number of states can be considered better in terms of space exploration. Rapidly-exploring random trees (RRT) and Probabilistic Road Maps (PRM) are sampling-based planners.

- Rapidly Exploring Random Tree (RRT)

  A Rapidly Exploring Random Tree (RRT) [18] is a tree-based algorithm that can be used to search non-convex high dimensional spaces. The tree begins with one initial node (called the root node) or the start node and randomly samples to find the next node. From every random node, it will search for the nearest neighbor [19]-[20] already in the tree and try to *expand* that node by generating a new node from this neighbor towards the random node. If the connection is valid it will connect it. RRT will keep on expanding until it finds the required goal or reaches its limit of time or memory.

For given start state S, and total k vertices, an RRT tree is constructed as shown in Algorithm 1.

Step 3 in the algorithm: If the environment has obstacles the random sample should also be checked for collision with an obstacle and reject the samples in C_{obs}.

Step 4 in the algorithm: To find the nearest neighbor the node nearest to the random sample must be calculated. This can be achieved by calculating distance function in Algorithm 2.

Algorithm 1 Basic RRT ALGORITHM: GENERATE RRT $(S, k, \Delta t)$.

procedure
Tree.int(S);
  for all i in k **do**
    $X_{rand} \leftarrow SAMPLE\_RANDOM\_STATE$;
    $X_{near} \leftarrow FIND\_NEAREST\_NEIGHBOR(X_{rand})$;
    $X_{new} \leftarrow NEW\_STATE(X_{near}, stepsize)$;
    $Tree.add\_vertex(X_{new})$;
    $Tree.add\_edge(X_{near}, X_{new}, u)$;
  end for
return $Tree$
end procedure

Algorithm 2 FIND NEAREST NEIGHBOR

procedure $FIND\_NEAREST\_NEIGHBOR(q_{rand}, Tree)$
  for all n in N **do**
    if $dist\ (q_{rand}, n) < d$ **then**
      $X_{near} = n$;
      $d \leftarrow dist(q_{rand}, n)$;
      return q_{rand}
    end if
  end for
return X_{near}
end procedure

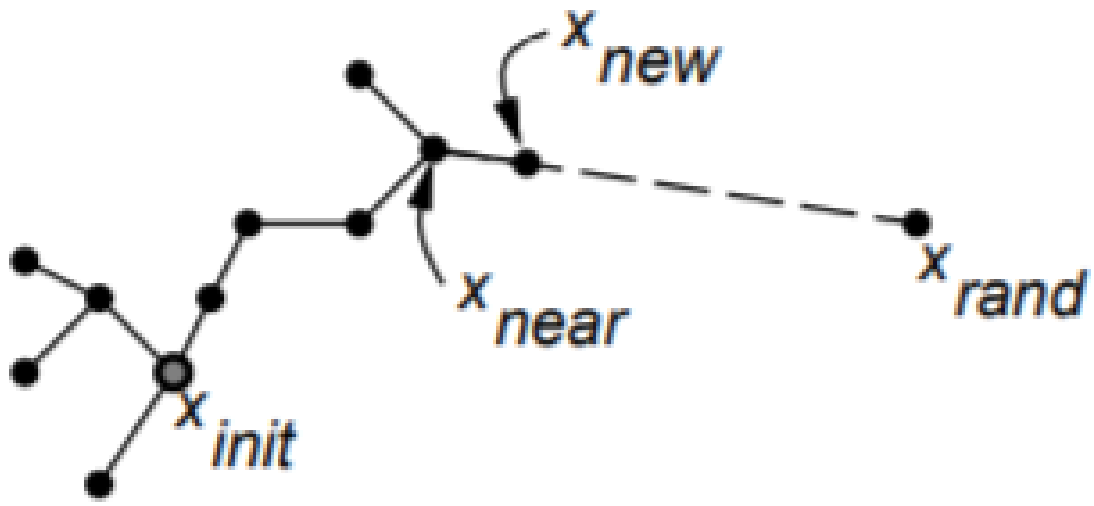


Figure 2.1: A basic RRT is generated incrementally by sampling new X_{rand}, searching for X_{near} in the tree and finding X_{new} to match the step size in the direction of X_{rand}. Image from "fundamental sampling issues in a motion-planning" presentation by Steven M. LaValle, University of Illinois.

- PRM

  Probabilistic Road Maps (PRMS) [3] are a *roadmap* graph in the configuration space, that can be used to generated paths. The goal of this method is to generate a topological graph which can efficiently solve the motion-planning problem. Such a type of graph is called a road map. The samples are generated in the map using uniform or random sampling in the collision-free space. Then we attempt to connect nearby sampled points to create a *roadmap* $G(V, E)$. In the algorithm below G represents a graph, N is the number of samples, C_{free} is the configuration space free of obstacles.

Algorithm 3 Basic PRM ALGORITHM[3]:BUILT_ROADMAP

1: **procedure**
2: $G.init()$; $i \leftarrow 0$;
3: *while* $i < N$
4: **if** $\alpha(i) \in C_{free}$ **then**
5: $G.add\_vertex(\alpha(i))$;
6: $i \leftarrow i + 1$;
7: **end if**
8:
9: **for all** q $\in NEIGHBORHOOD(\alpha(i).G)$ **do**
10: **end if**$((!G.same\_Component(\alpha(i), q))$ *and* $CONNECT(\alpha(i), q))$
12: $G.add\_Edge(\alpha(i)$,q);
13: **end if**
15: **return** G
16: **end procedure**

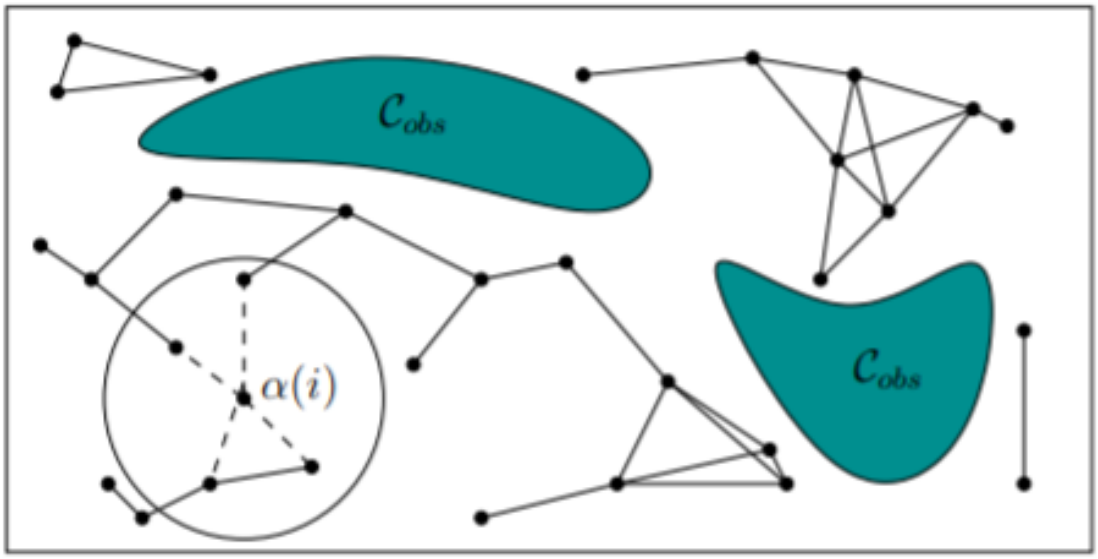


Figure 2.2: The sampling-based roadmap is constructed incrementally by attempting to connect each new sample, $(\alpha(i))$, to nearby vertices in the roadmap. Figure is taken from Planning Algorithms textbook [3]

.

2.2 Model of Movement Using Global Inputs

In this section we discuss more details about the global inputs and some assumptions. In this thesis, we consider the situation where all the robots are moving under the same field effect e.g., iron particles pulled by a magnetic field. When any of the robots collide with an obstacle that robot will stop moving until it has free space to move. However, the robots can slide along edge of obstacles.

Consider a group of the robots $r(r_1, r_2, ..., r_n)$ moving under the command to move from the start $S(s_1, s_2, ..., s_n)$ towards their individual goal $G(g_1, g_2, ..., g_n)$. Each move command is a force vector in 2D that can be written as $M = (M_x + M_y)$. Also, let each robot in 2D have an X and Y coordinate. So R can be given by $R(x, y)$ and the state evolution equation is given by

$$x_{new} = x_{old} + M_x \text{ and } y_{new} = y_{old} + M_y. \tag{2.1}$$

The above equations can be generalized in Algorithm 4.

Algorithm 4 Global inputs

```
1: procedure MOVE ROBOTS FROM S TO G
2:
3:     for all i < N do
4:       x_i(new) = x_i(old) + M_x;
5:       y_i(new) = y_i(old) + M_y;
6:         if ri ∉ C_free then
7:             x_i(new) = x_i(old);
8:             y_i(new) = y_i(old);
9:         end if
10:      ri = (x_i(new), y_i(new));
11:      i ← i + 1;
12:    end for
13:      return r = (r_1, r_2, ..., r_n)
14:
15: end procedure
```

2.3 Calculation of Exploration of the Space

In this section, we discuss the method used to calculate the coverage of the configuration space by the robots. We want to compare the effect of changing a different parameter in the algorithm to find out how we can improve exploration of the space. It is important to find the exploration of the space as we also want to reuse our previously generated RRT tree for different queries. To calculate coverage, we divide the whole space into grids. The size of the grids can be decreased to improve the precision (details in Section 4 Simulation and Results). As shown in Figure 2.3 the frame and the bounds remain the same in all the environments we have used for this thesis.

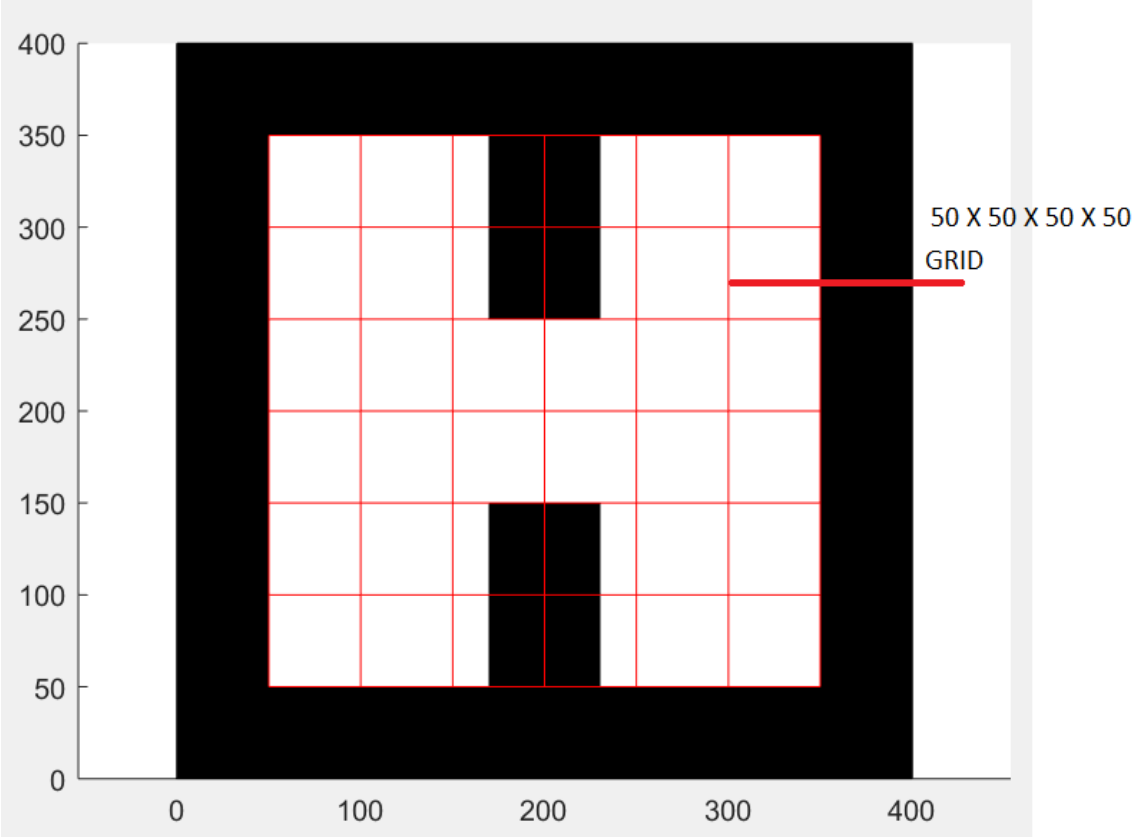


Figure 2.3: Example of a grid on an environment

2.4 Algorithms

2.4.1 Collision Avoidance

Collision avoidance is an important algorithm, used in this thesis as we not only have to avoid passing through obstacles, but also must use the obstacles to reach our goal. There are many kinds of collision checks we must perform. OMPL library does not have a collision detection/avoidance algorithm, so we had to develop our own for the experiments. As our robot is just a particle, we will do the following three collision checks.

IsValidPoint: Checks whether the point is valid as we assume that our robot is just a point, not a circle. If point has the coordinate (x, y) and the plane is given

by $(x_{max},\ y_{max})$ and $(x_{min},\ y_{min})$ then the algorithm can be given by:

Algorithm 5 IsValidPoint(x, y)

1: **procedure** CHECK IF POINT IS VALID
2: **return** $x_{min} \leq x \leq x_{max}$ **AND** $(y_{min} \leq y \leq y_{max})$
3: **end procedure**

DoSegmentIntersect: In algorithm 6, we check whether the line segments are intersecting. Reference (developed during course "Algorithmic Robotics" at Rice University).

IsValidPath: A collision check routine with the path (straight line segment) and rectangle obstacle to check whether the path is valid and the robot is not jumping over the obstacle. This checks line segment collision between the path and all the sides of the rectangle. Let the obstacle sides be given by (o_1, o_2, o_3, o_4) and path of a robot is given by (p_1, p_2).

2.4.2 Distance Function

The Euclidean distance is used in the RRT in the configuration space to find the nearest point to be added to the tree and is given by the equation below. The distance between two points is defined as the square root of the sum of the squares of the differences between the corresponding coordinates of the points. Let $X_s = [q_{s1}, ..., q_{sn}]$ $X_f = [q_{f1}, ..., q_{fn}]$. The distance between two points is given by

$$distance = \sqrt{\sum_{i=1}^{n} (q_{si} - q_{fi})^2}. \tag{2.2}$$

Algorithm 6 *DoSegmentIntersect*$((A, B), (C, D))$

procedure DETAILS
Given two segments with points (A, B) (C, D).
Check if both the line segments are parallel and check if they are vertical, return false.
Check if intersecting on one endpoint, return that point.
Find if the line segments are overlapping.
Using line equations and slope and find intersecting point $y_{coordinate} = slope_{coordinate} + constant$.
Confirm whether the intersecting point is on the line segment.
Algorithm in detail:
bool isValidSegment($A_x, A_y, B_x, B_y, C_x, C_y, D_x, D_y$**)**

    if $(A_x == B_x$ && $C_x == D_x)$ **then** ▷ check if the lines are parallel
      return false;
    else if $A_x == B_x$ **then** ▷ Check if the line is vertical
        $m_{CD} = (D_y - C_y)/(D_x - C_x)$; ▷ slope
        $b_{CD} = C_y - m_{CD} \cdot C_x$; ▷ line equation
        $int_y = m_{CD} \cdot A_x + b_{CD}$;

      if $(A_x \geq \min(C_x, D_x)$ & $A_x \leq \max(C_x, D_x)$ & $int_y \geq \min(C_y, D_y)$ & $int_y \leq max(C_y, D_y)$ & $int_y \geq \min(A_y, B_y)$ & $int_y \leq \max(A_y, B_y))$ **then**
        return true;
      else
        return false;
      end if

    else if $(C_x == D_x)$ **then**
        $m_{AB} = (B_y - A_y)/(B_x - A_x)$;
        $b_{AB} = A_y - m_{AB} \cdot A_x$;
        $int_y = m_{AB} \cdot C_x + b_{AB}$;

      if $(C_x \geq \min(A_x, B_x)$ & $C_x \leq \max(A_x, B_x$ & $int_y \geq \min(A_y, B_y)$ & $int_y \leq \max(A_y, B_y)$ & $int_y \geq \min(C_y, D_y)$ & $int_y \leq \max(C_y, D_y))$ **then**
        return true;
      else
        return false;

      end if
    end if

35: **procedure** CONTINUE
36: $m_{AB} = (B_y - A_y)/(B_x - A_x)$; ▷ compute line equations for line AB and line CD endpoint
37: $b_{AB} = A_y - m_{AB} \cdot A_x$;
38: $m_{CD} = (D_y - C_y)/(D_x - C_x)$;
39: $b_{CD} = C_y - m_{CD} \cdot C_x$;
40: $int_x = (b_{CD} - b_{AB})/(m_{AB} - m_{CD})$; ▷ Find intersection point
$int_y = m_{AB} \cdot int_x + b_{AB}$;
41:
42: **if** $(int_x \geq min(A_x, B_x)$ & $int_x \leq max(A_x, B_x)$ & $int_x \geq min(C_x, D_x)$ & $int_x \leq max(C_x, D_x)$ & $int_y \geq min(A_y, B_y)$ & $int_y \leq max(A_y, B_y)$ & $int_y \geq min(C_y, D_y)$ & $int_y \leq max(C_y, D_y))$ **then** ▷ Check if the intersection point int_x and int_y is on the line segments
43: **return** true;
44: **else**
45: **return** false;
46: **end if**
47: **end procedure**

Algorithm 7 IsValidPath

1: **procedure** INVALIDPATH $(Obstacle, Path)$
2: $IsValidSegment(Obs_{side1}, Path)||IsValidSegment(Obs_{side2}, Path)||$
3: $IsValidSegment(Obs_{side3}, Path)||IsValidSegment(Obs_{side4}, Path)$
4: **end procedure**=0

2.4.3 Nextpoint Function

Suppose a point p is given by $p = (x, y)$. In our RRT we use a list of 16 angles $\phi = 22.5° \cdot k, k \in [0, 15]$ and we compute which of these directions generates a state that is closest to the randomly generated sample point.

Algorithm 8 Find Nextpoint algorithm

1: **procedure** NEXTPOINT $(x, y, distance, angle)$
2: $x_n = x + distance \cdot cos((angle \cdot pi)/180)$
3: $y_n = y + distance \cdot sin((angle \cdot pi)/180)$
4: **return** (x_n, y_n)
5: **end procedure**

2.4.4 Grid Coverage

Suppose the grid is as shown in Figure 2.3 for 2 robots and each robot can move in the x and y-direction. The configuration will be R^4 and for these 2 robots, the grid is created in a 4D configuration space. The grid discritizes the configuration space into voxels of equal size. Our coverage function reads the tree and counts the total number of the voxels covered. The covered voxels will give the percentage coverage of the space.

2.5 OMPL Library [2]

The open motion-planning library (OMPL) is an open source motion-planning library for sampling-based motion-planning [2]. There are many tools inbuilt in this library that are helpful for easily creating a sampling-based planner. It is important to understand some terms used in OMPL library.

- *State space*: The state space in the OMPL is same as the C-space of any configuration discussed in the previous chapters. Translations and rotation can be used to define the configuration for freely moving rigid bodies.

- *Control Space*: This is used to define the space for a dynamic robot. We are not using Controls Space in this thesis.

- *Sampler*: Different states can be generated from the state space using different sampler in OMPL. This is important for a sampling-based planner (geometric).

- *State Validity Checker*: This is an important routine which can be helpful to find whether the sampled state is valid or not. It will check the bounds for the robot and for collisions.

- *Local Planner*: Local planners are routines which within the geometric constraints can be used to solve some special requirements for the planner. In our thesis, it is used to find the next valid point function.

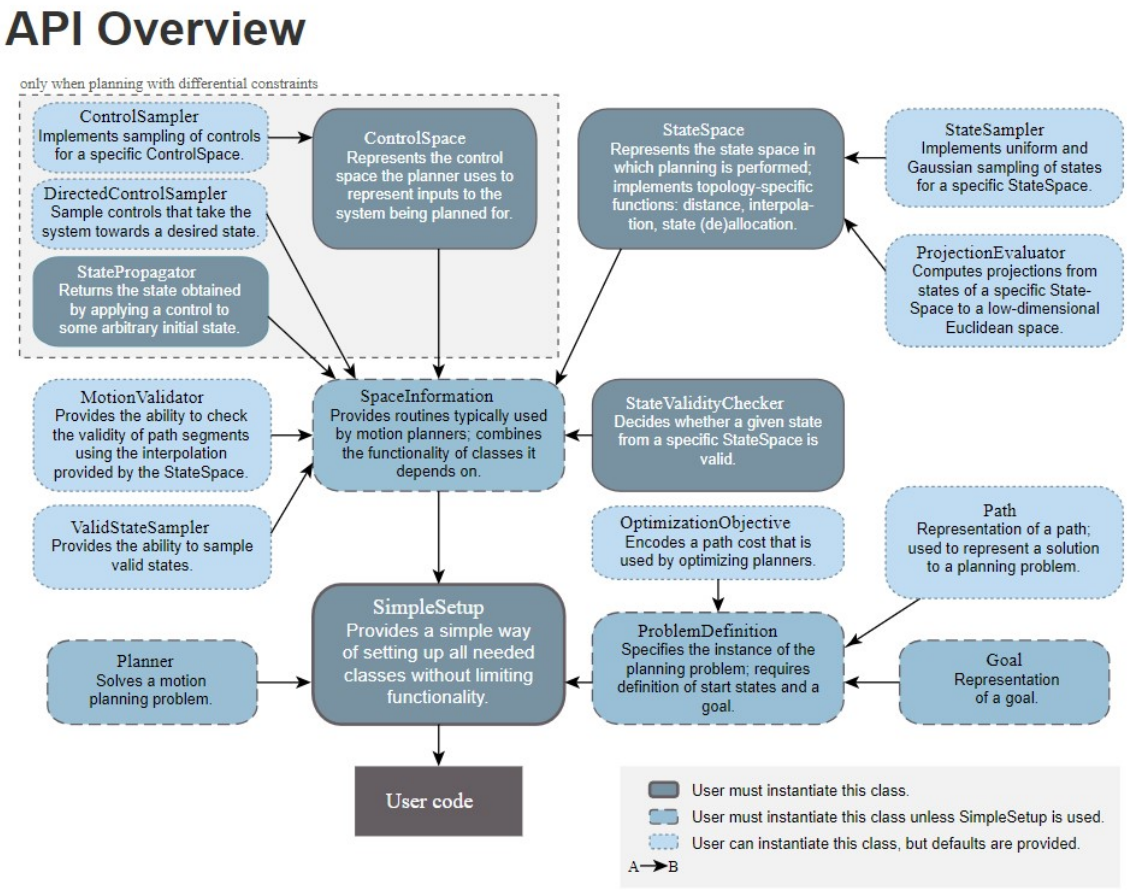


Figure 2.4: OMPL class structure reference
`https://ompl.kavrakilab.org/api_overview.html`.

It is important to define a configuration space correctly in OMPL. If the robot can perform only translation motions in X and Y then the C-Space is given by an R^2 configuration while if it can move in X, Y, Z direction, then it is given by R^3. In OMPL translation motion can be given by $RealVectorStateSpace$. Similarly, Rotations in three dimensions, Roll, Pitch and Yaw can be given by $SO3StateSpace$. Combinations of such spaces can be modelled by using $CompoundedStateSpace$ in

OMPL. Also, a sampling of the space is important in sampling-based planners. OMPL includes four types of samplers: state space samplers (*StateSampler*), valid state samplers (*ValidStateSampler*), control samplers (*ControlSampler*), and directed control samplers (*DirectedControlSampler*).

2.6 OMPL motion-planning Problem with an Example.

In this section we discuss an example of 2 robots that can move in the x and y directions. The aim of the robots is to move from the start to the goal using a planner from the OMPL library. The C++ code for the problem discussed above is as follows:

```
//Check whether the state is valid using collision checking
//or other required criteria

Bool isValidState (state)
    Return true or false

//r4 is the configuration for a 2 point robot which moves in 2D
//set the State space bounds

ompl::base::StateSpacePtr r4
        (new ompl::base::RealVectorStateSpace (4));
ompl::base::RealVectorBounds bounds (4);

// Define the environment to be between 0 to 400

bounds.setLow (0);
bounds.setHigh (400);

r4 ->as <ompl::base::
            RealVectorStateSpace>()->setBounds(bounds);
ompl::geometric::SimpleSetup ss (r4);
 ss.setStateValidityChecker ((isValidState));

// Set start and Goal position for the robots

 ompl::base::ScopedState<> start(r4);
```

```
      start[0] = 240;
      start[1] = 340;
      start[2] = 260;
      start[3] = 260;

ompl::base::ScopedState<> goal(r4);
     goal[0]= 320;
     goal[1]= 95;
     goal[2]= 320;
     goal[3]= 313;

ss.setStartAndGoalStates (start,goal)

//Select the Motion planner from OMPL library or
//User defined. Here it is RRT

ompl::base::PlannerPtr planner
         (new ompl::geometric::RRT
             (ss.getSpaceInformation()));

ss.setPlanner(planner);

// Run solver and select the time for a solver to run

ompl::base::PlannerStatus solved = ss.solve(1.0);

If {solved}{

     ompl::geometric::PathGeometric &
                 path = ss.getSolutionPath ();

     // Once solved print the path

     path.printAsMatrix (std::cout);
}
```

2.7 Updated RRT Algorithm for Multiple Robots

For given multiple start positions and multiple goal positions, an RRT Tree and total k vertices are constructed as shown in the algorithm as follows. Starting postions of the individual robots combines as S_h in a higher dimension and goals as

G_h, k_h is the vertex of the tree in higher dimensions. For finding the next step we have used 16 angles and a fixed step size. Our planner will select the configuration nearest to our random sample

Algorithm 9 Updated RRT ALGORITHM

```
1: procedure GENERATE RRT (S_h, k_h, Δt)
2: Tree.int(S_h);
3:     for all i in k_h do
4:       X_rand ← SAMPLE_RANDOM_STATE;                ▷ Higher dimension
5:       X_near ← FIND_NEAREST_NEIGHBOR(X_rand);
6:       X_new ← NEW_STATE(X_near, stepsize, angles[16]);
7:       Tree.add_vertex(X_new);
8:       Tree.add_edge(X_near, X_new, u);
9:     end for
10: return Tree
11: end procedure
```

Step 3: If the environment has obstacles, the random sample should be checked for collisions with obstacles and reject samples in the collision. Also we must check for collisions during the path.

Step 4: To find the nearest neighbor, the node nearest to the random sample must be calculated. We have predetermined the step size and use 16 angles and must calculate which is nearest to the random sample. This can be achieved by using the distance function as follows:

Algorithm 10 FIND NEAREST Neighbor

```
1: procedure FIND_NEAREST_Neighbor(q_rand, Tree)
2: d = ∞;
3:     for all  n in N do
4:         if dist (q_rand, n) < d then
5:             X_near = n;
6:             d ← dist(q_rand, n);
7:             return q_rand
8:         end if
9:     end for
10: return X_near
11: end procedure
```

Chapter 3

Simulations and Results

This chapter presents the simulation results. With these results, the approach of solving the motion-planning problem for the multiple robots under global inputs can be evaluated. Results are evaluated by changing the different parameters including (1) number of nodes, (2) step size of the robot in RRT, (3) environment, (4) changing grid sizes, (5) changing distances between the start and goal and, (6) changing the initial location of robots. This approach gives us an idea of how each factor listed above affects exploration of the space.

3.1 Environment 1: L-Shaped Obstacle

As shown in Figure 3.1 Environment 1 has a border of obstacles (shown in black) on the sides and an L-shaped obstacle (in black) in the center. The starting positions for the robots are shown in the pictures as black and green discs for robot 1 and 2 respectively. The goals for the robot 1 and robot 2 are shown as a blue circle and red circles respectively. Also, the RRT for robot 1 is shown in cyan color and the robot 2 is in yellow color. The whole space is in a 4D dimension, but the picture shows the projection of the 4D space into 2D. The aim of each robot is to simultaneously reach its individual goal even when moved by global inputs as described earlier.

3.1.1 Changing Step Size

In the RRT algorithm defined above (Section 2.1.1.1), at every random sample, we move the robots with a fixed step and we call it the *robostep*. The aim of this

experiment is to observe how the robostep of the robot affects the coverage of the C-space. To achieve this goal, we show pictures of our RRT Tree for 2 robots in 2D dimensions with an increasing number of steps to compare. The start and goal positions of the robots is the same for each experiment, the grid size is $50 \times 50 \times 50 \times 50$, and the coverage of the space is calculated as discussed in Section 2.3.

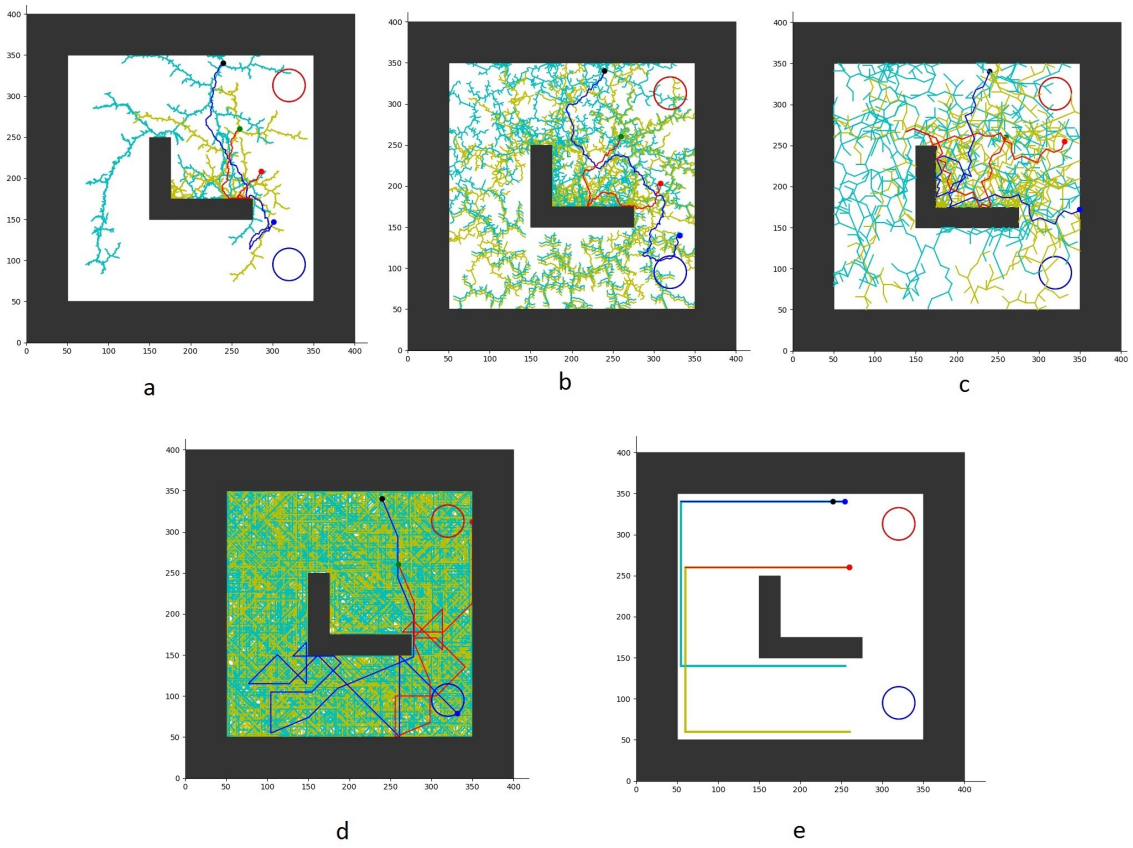


Figure 3.1: The image represents the effect of step size in RRT on coverage of the search space. RRT with time 0.5 sec and step size (a) 1, (b) 3, (c) 5, (d) 50, (e) 200 units.

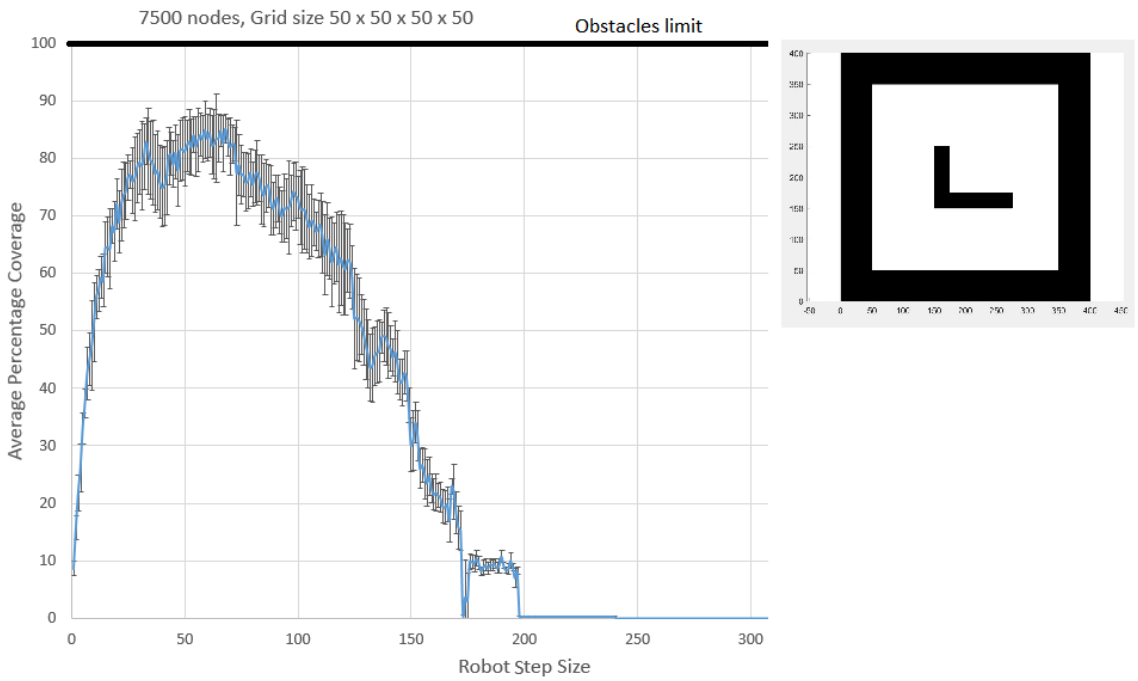


Figure 3.2: This shows the coverage of the space as a function of step size for a fixed number of nodes 7500 nodes and grid size is 50 $\times$50 $\times$ 50 $\times$ 50.

3.1.2 Changing the Number of Nodes

The number of nodes in the RRT algorithm is proportional to the time we allow the RRT to sample the space until it reaches the goal. The exploration of the C-space will also depend on how long we keep sampling the space. In all our experiments we increased the time taken by RRT to solve the problems, instead of increasing the number of nodes. The relationship between the time taken to solve the map is obtained and we measure the number of nodes in the RRT tree. Figure 3.3 clearly shows that the exploration of the space increases with the increase in the number of nodes.

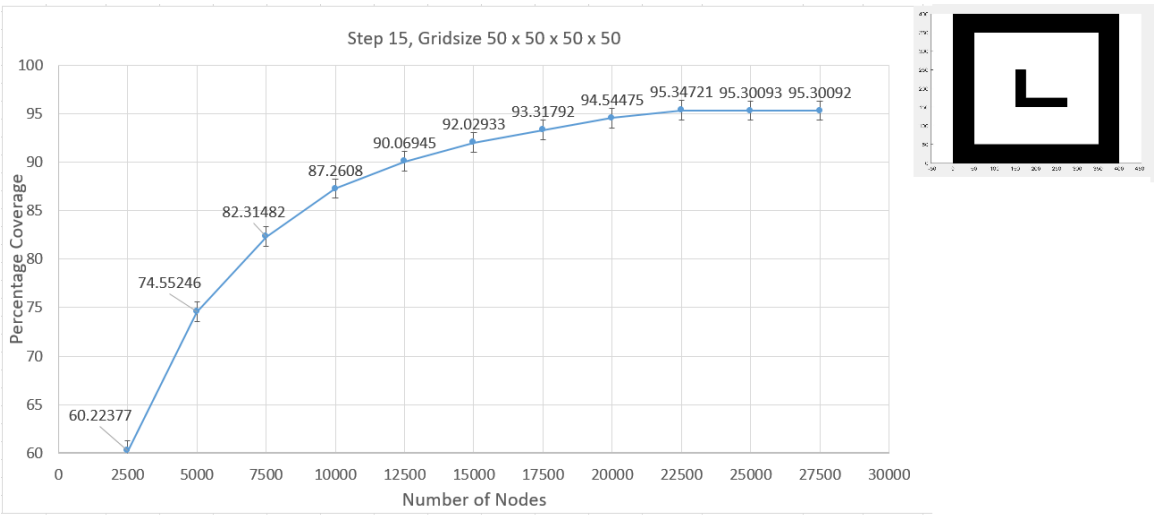


Figure 3.3: A graph of the percentage coverage of the map vs. a number of nodes of the RRT tree. The image in the right-hand corner represents the environment used for the experiment.

To compare the results from changing the number of steps and the number of nodes, we can combine the data. These results are represented in Figure 3.4.

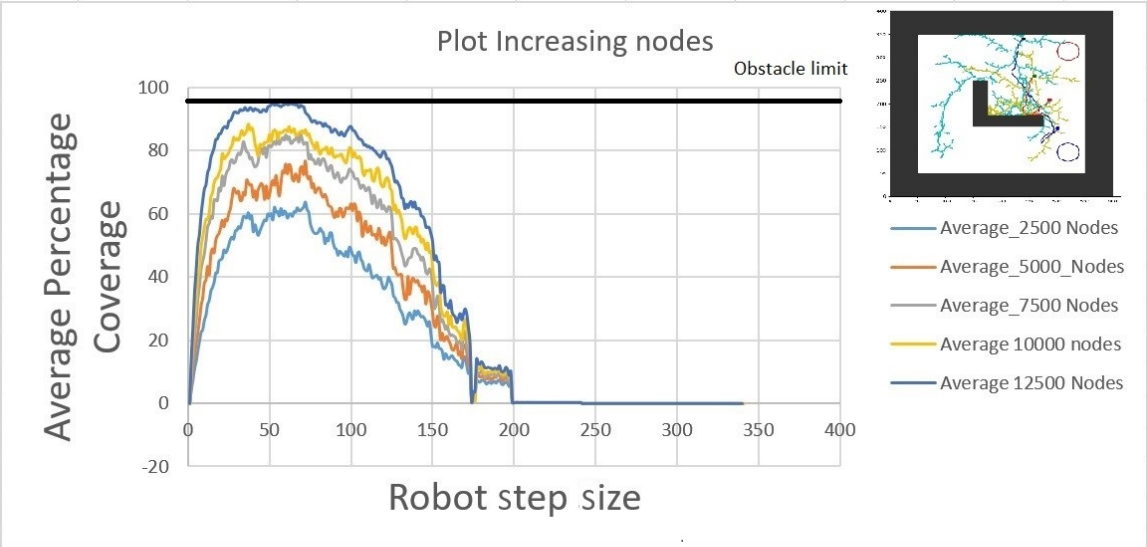


Figure 3.4: Comparison of the average percentage of coverage and robostep using the environment shown in the right-hand corner. Also includes a comparison of the average percentage coverage and number of nodes of the RRT Tree.

3.1.3 Changing Grid Size

In this experiment, we change the grid size to observe the effect on the coverage percentage. The coverage is not actually changing, we make each voxel (the grid is high dimension) smaller. The experiment is performed by changing the grid size to 40. It is the same as the environment before. The dimension of the grid increases exponentially with the increase in the number of the robots in the swarm. As the dimension increase, dividing the map into the grid becomes computationally complex and time-consuming. However, for our work, it is an effective way of comparing the outputs.

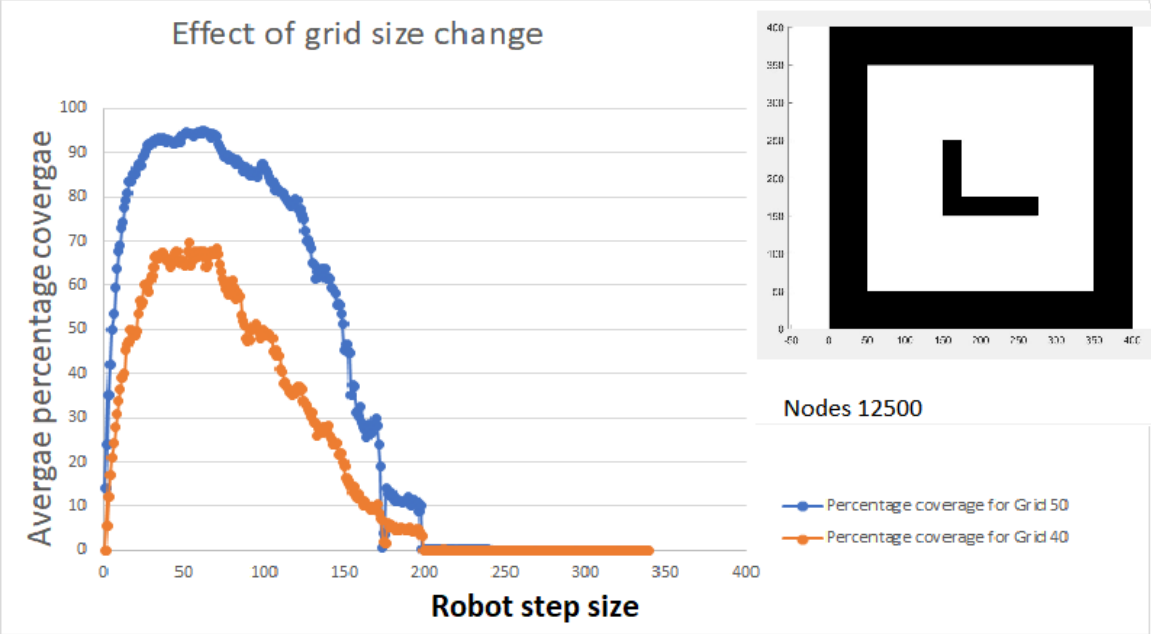


Figure 3.5: Represents the effect of grid size change on the average percentage coverage.

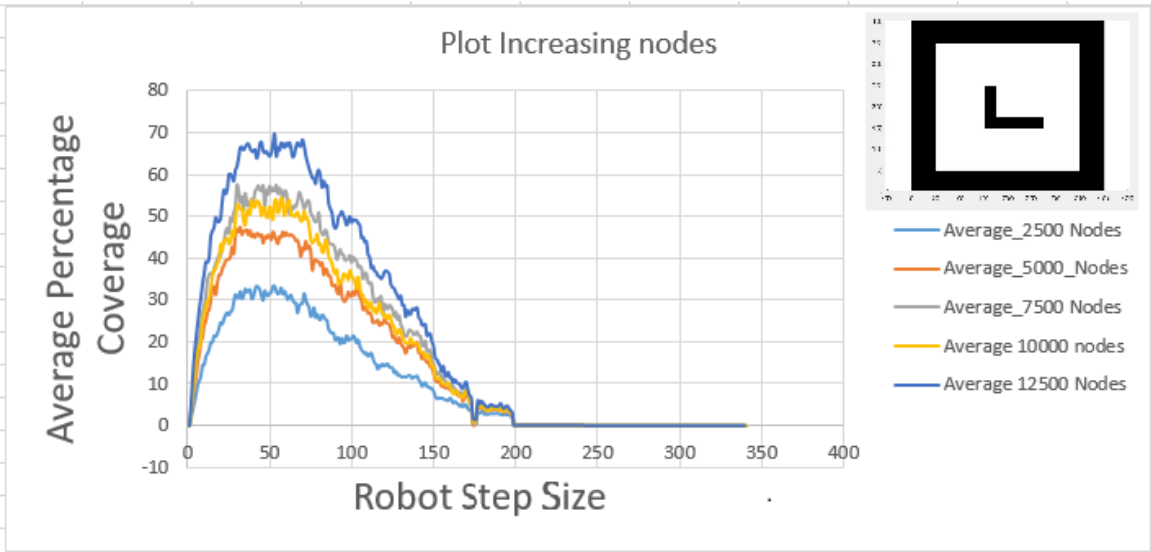


Figure 3.6: Comparison of the average (of 10 trials per robot step size) percentage coverage and robostep using the environment shown in the right-hand corner when the grid size is 40. Also includes a comparison of the average percentage coverage and number of nodes of the RRT tree.

3.1.4 Changing the Initial Distance Between Robots

This section examines if the initial positions of the robots affects the performance of the RRT with the same number of nodes and robostep. To study the effect of initial position, we considered eight different cases, and in each case the initial positions of the robots are closer. We change the initial positions of the robots and measure its effect on coverage. As we go from case 1 to case 8 the distance decreases as shown in Figure 3.7.

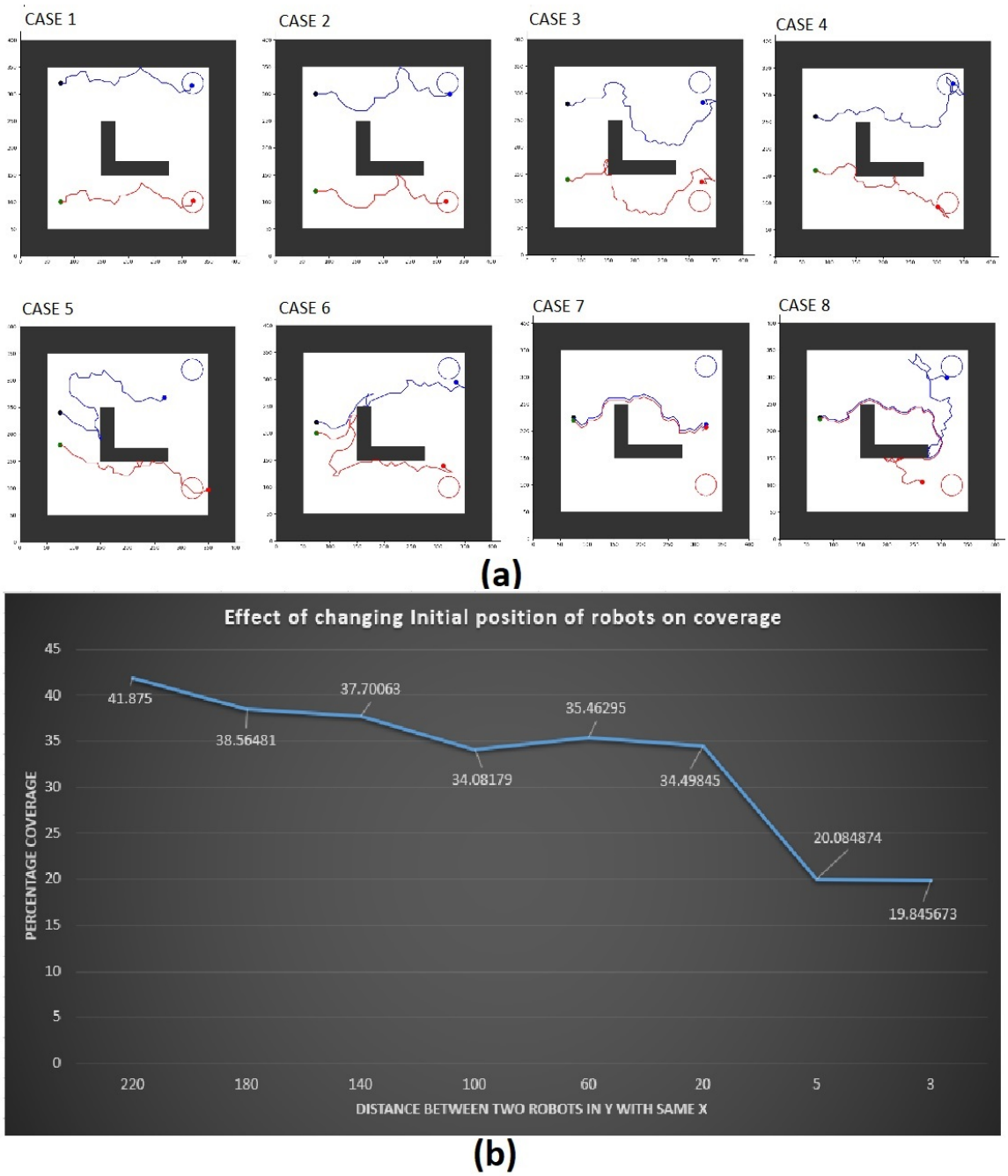


Figure 3.7: a) Maps with different cases. The initial distance between two robots decreases as we go from case 1 to case 8 b) Graph of Percentage coverage for all the cases in Figure 3.7 (a).

As shown in Figure 3.7 (b) the initial separation of the robots does not have effect until a critical distance, after which the coverage drops quickly. We can infer

from these experiments that the motion planner could not cover the map area if the robots are initialized too close together because it must separate the robots to achieve the goal. If the goal locations of the robots are at the same relative distance as the short positions, it would be easy for the motion planner to solve it.

3.2 Environment 2: Short Narrow Passage

As shown in Figure 3.8 environment 2 has obstacles (shown in black) on the sides and two obstacles (in black) in the center creating a narrow passage. The starting locations for the robots are shown in the pictures as black and green discs for robot 1 and 2 respectively. The goals for the robot 1 and robot 2 are shown respectively as a blue and red circle. Also, the RRT for robot 1 is shown in cyan and robot 2 is yellow. The whole space is in 4D dimensions, but the picture shows the projection of the 4D space into 2D. The aim is to simultaneously bring each robot to its goal using global inputs as discussed earlier.

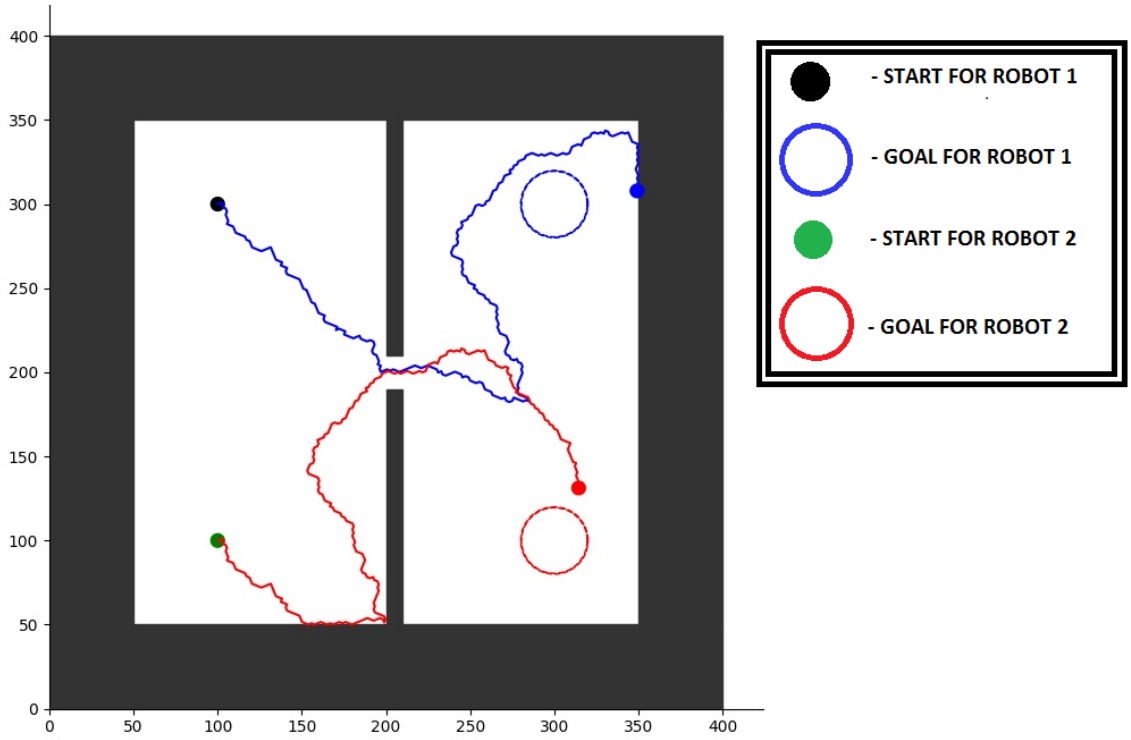


Figure 3.8: Solved environment 2 using our motion planner with the short narrow passage.

This environment has an obstacle in the middle, creating a narrow passage for the robots. As shown in Figure 3.9, increasing robostep size from 0 to 40 increases coverage. However, coverage is saturated after length step 40 till 120, where the step size of the robot has not created much effect. Interestingly, for all numbers of nodes, the coverage drops to zero for step length longer than 143 because further increase in the length cause collisions with the obstacles.

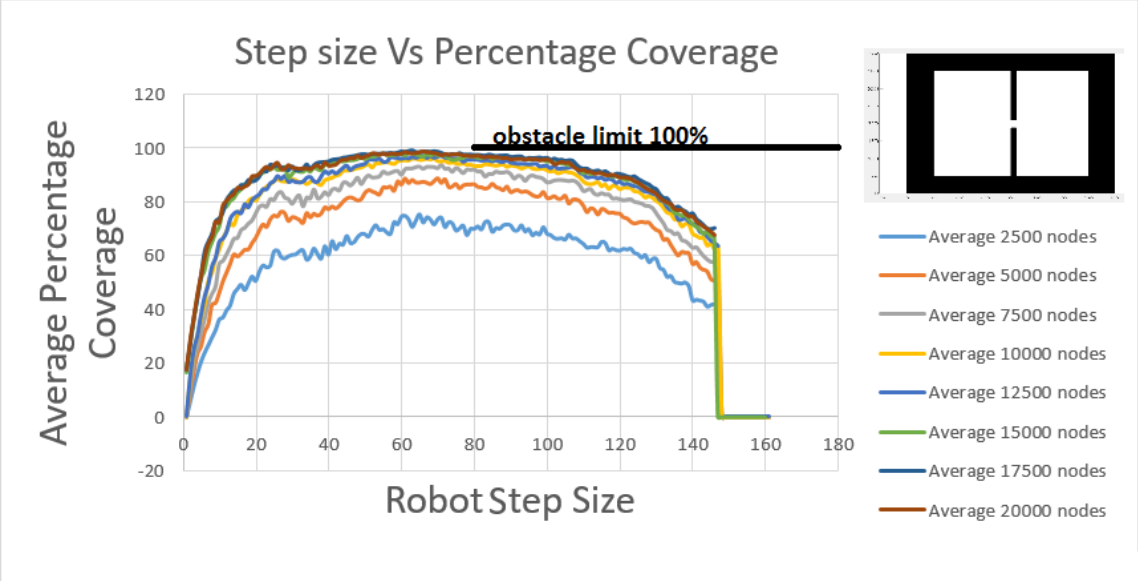


Figure 3.9: Average percentage of area covered compared with robot step size for a different number of nodes across 10 trials for each step for environment 2. The coverage was measured with grid size 50.

3.3 Environment 3: Long Narrow Passage

As shown in Figure 3.10 environment 3 has obstacles (shown in black) on the sides and two obstacles (in black) in the center, creating a narrow passage. The starting locations for the robots are shown in the pictures as black color and green discs for robot 1 and 2 respectively. The goals for the robot 1 and robot 2 are shown respectively as a blue and a red circle. The aim is to simultaneously bring each robot to its goal using global inputs as discussed earlier.

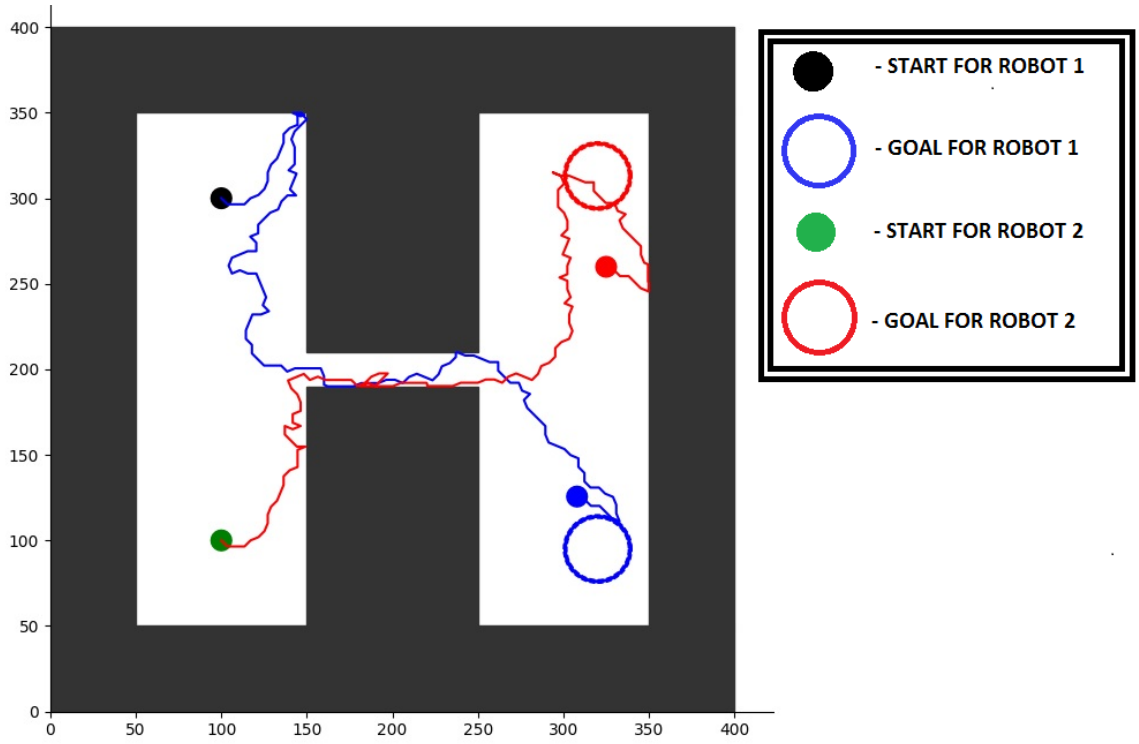


Figure 3.10: Solved environment 3 using our motion planner with the longer narrow passage.

As compared to environment 2, this environment has a broader narrow passage. From Figure 3.11 we can infer that increasing the nodes after 12500 has no significant effect. Also the coverage starts dropping after step size 100 and drops quickly after 180 because the robot cannot cross the narrow passage. After certain step-size, one of the robot would collide with the obstacles and decreases the coverage if the step size is too long.

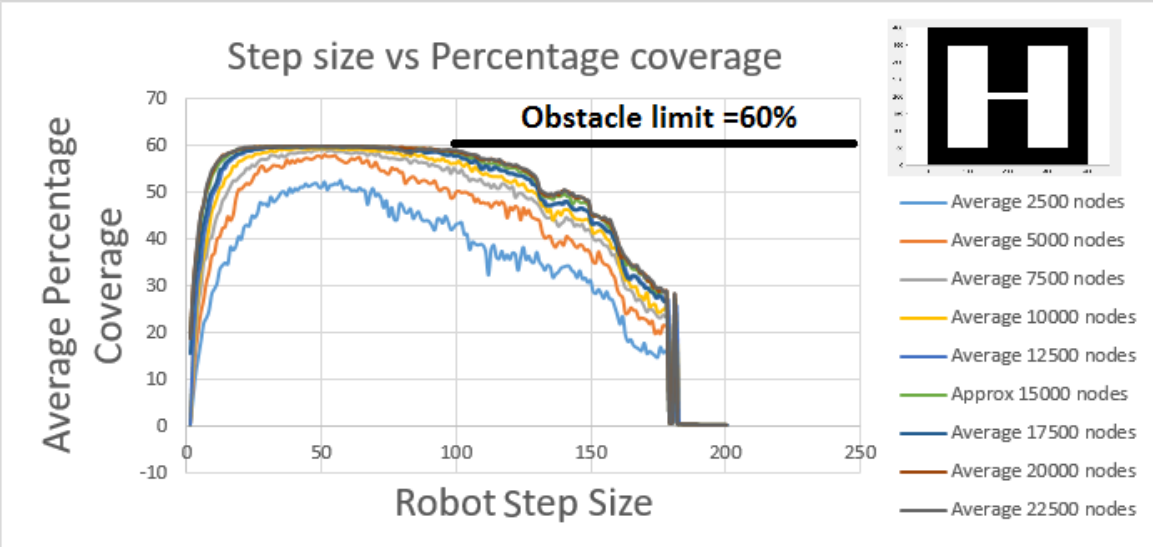


Figure 3.11: Average percentage of area covered compared with robot step size for a different number of nodes across 10 trials for each step in environment 3. The coverage was measured with grid size 50.

3.4 Environment 4: Zigzag Narrow Passage

Figure 3.12 environment 4 has obstacles (shown in black) on the sides, and Obstacles in the center create a narrow but zigzag passage. The starting locations for the robots are shown in the pictures as black and green discs for robot 1 and 2 respectively. The goals for the robot 1 and robot 2 are shown as a blue and a red circle. The aim is to simultaneously bring each robot to its goal using global inputs as discussed earlier.

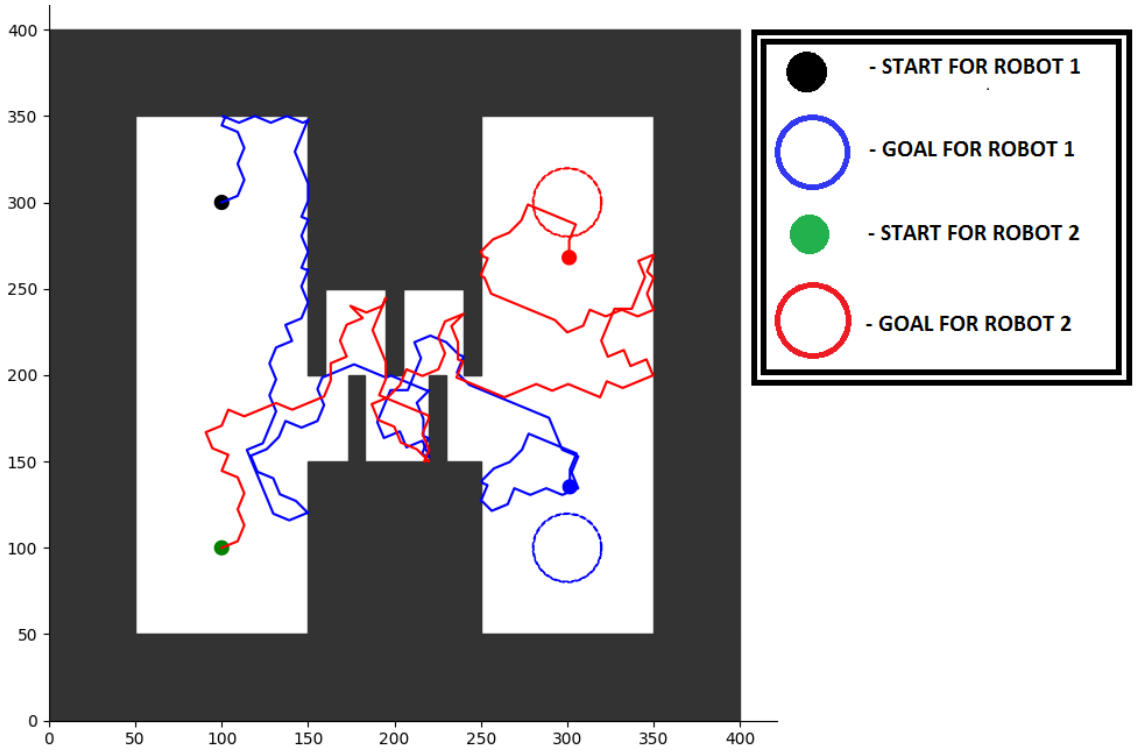


Figure 3.12: Solved environment 4 using our motion planner with the narrow passage and a zigzag obstacle pattern.

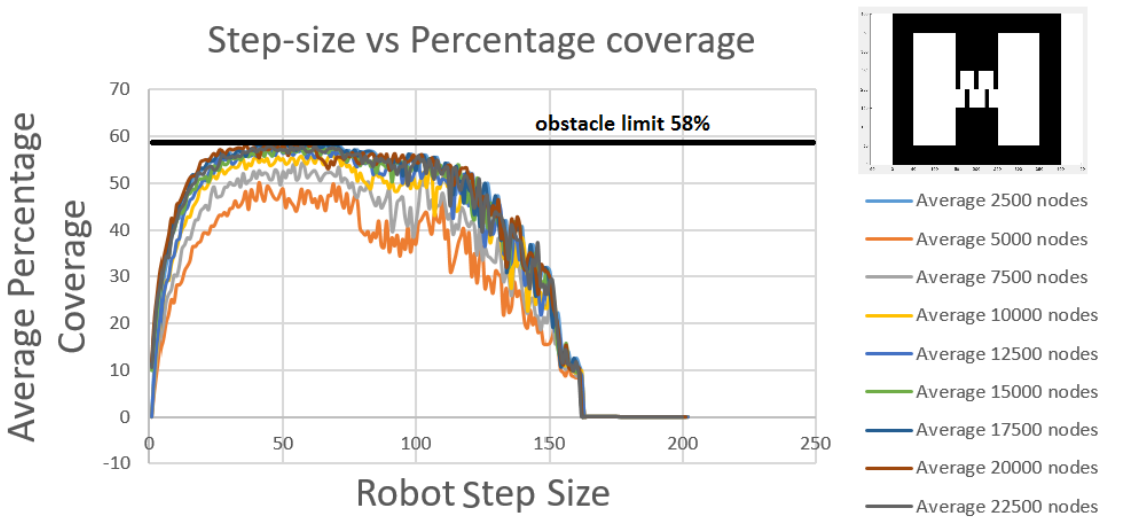


Figure 3.13: Average percentage of area covered compared with robot step size for a different number of nodes across 10 trials for each step for environment 4. The coverage was measured with grid size 50.

3.5 Environment 5: Human Digestive System

As shown in Figure 3.14 environment 5 has obstacles (shown in black) on the sides and other obstacles create an environment like a human digestive tract. The outputs show the behaviors of robots in this difficult environment. The starting locations for the robots are shown in the pictures as black and green discs for robot 1 and 2 respectively. The goals for the robot 1 and robot 2 are shown as a blue and a red circle respectively. Also, the RRT for robot 1 is shown in cyan and the robot 2 is in yellow. The whole space is in 4 dimensions, but the picture shows the projection of the 4D space into 2D. The aim is to simultaneously bring each robot to its goal using global inputs as discussed earlier.

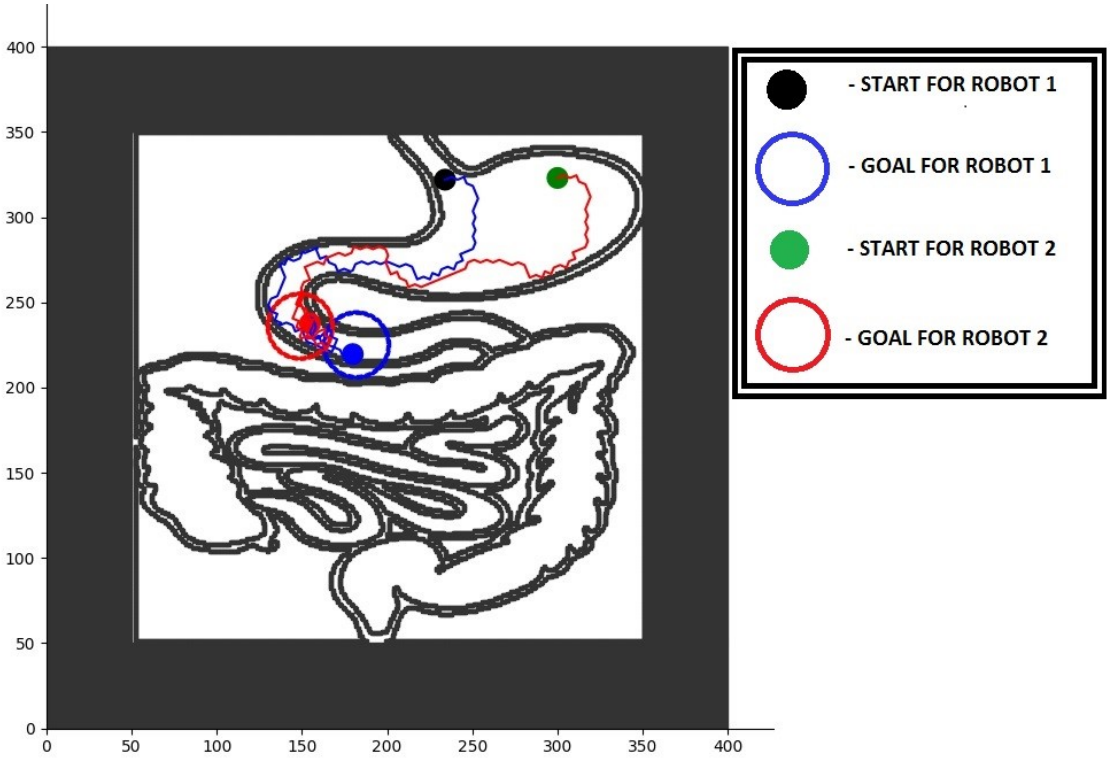


Figure 3.14: Solved environment 5 using our motion planner. This environment represents the human digestive system.

This environment is designed using small rectangles to create the shape of the

human digestive system. In Figure 3.14 we showed that the robot is moving in the area which does not have sharp corners. Our algorithms work fine with these different shapes of the obstacles. Figure 3.15 shows that the robots can move to two entirely different locations at the same time. Robot 1 with a start location shown by in a black disc moves to the stomach area of the digestive system while the robot with the start location in green moves to the large intestine area of the human digestive system. We moved the robots to different parts of the digestive system simultaneously.

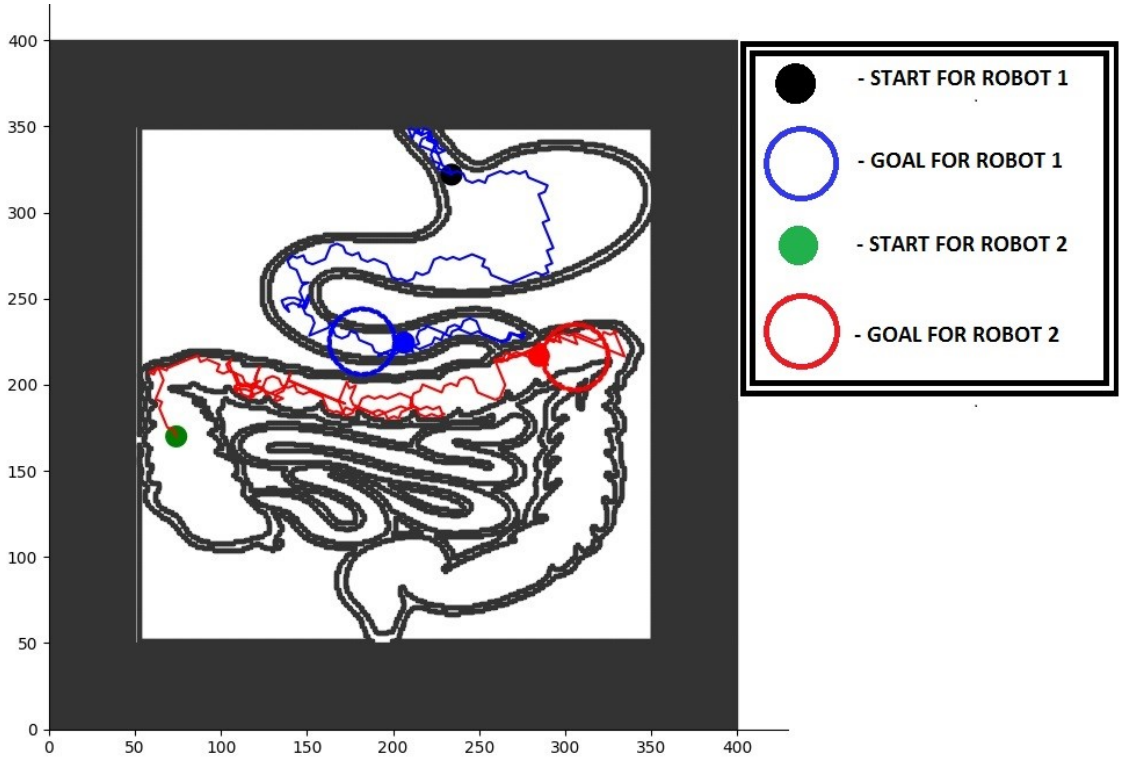


Figure 3.15: Solved environment 5 using our motion planner. This environment represents the human digestive system. The start and goal locations of the robots are in different location of the digestive system than Figure 3.14.

3.6 Environment 6: Vascular Network

After solving some basic maps we applied the motion-planning to a complex map of a leaf's vascular network. Environment 6 has a border of obstacles creating an environment of a vascular network space. The start for the robots are shown in

the pictures in black and green circles for robot 1 and 2 respectively. The goals for the robot 1 and robot 2 are shown as a blue and a red circle respectively. The aim of each robot is to reach it individual goal even being under the effect of global inputs as discussed earlier.

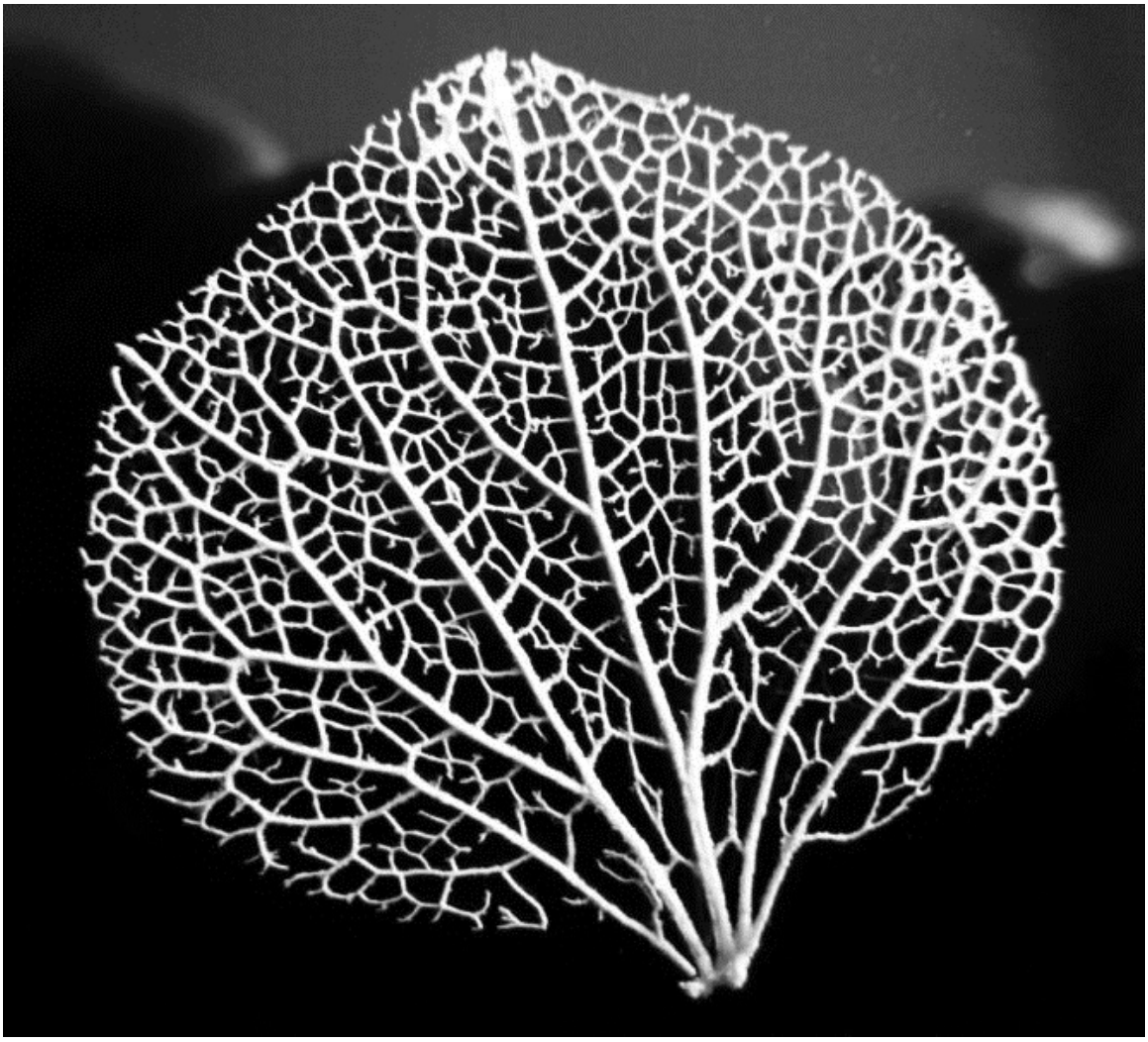


Figure 3.16: Leaf vascular network.
http://browpicz.pw/Black-leaf-vein-pattern-WALL-PANELS-t-Leaves-Leaf.html

We created this network from picture we found on the website. We converted the Figure into an environment using boundary detection techniques in MATLAB image processing tool box. We found the coordinates of the border and converted

the borders into small rectangles with a length and width of 1 unit. The robots successfully solved the complex map as shown in Figure 3.17.

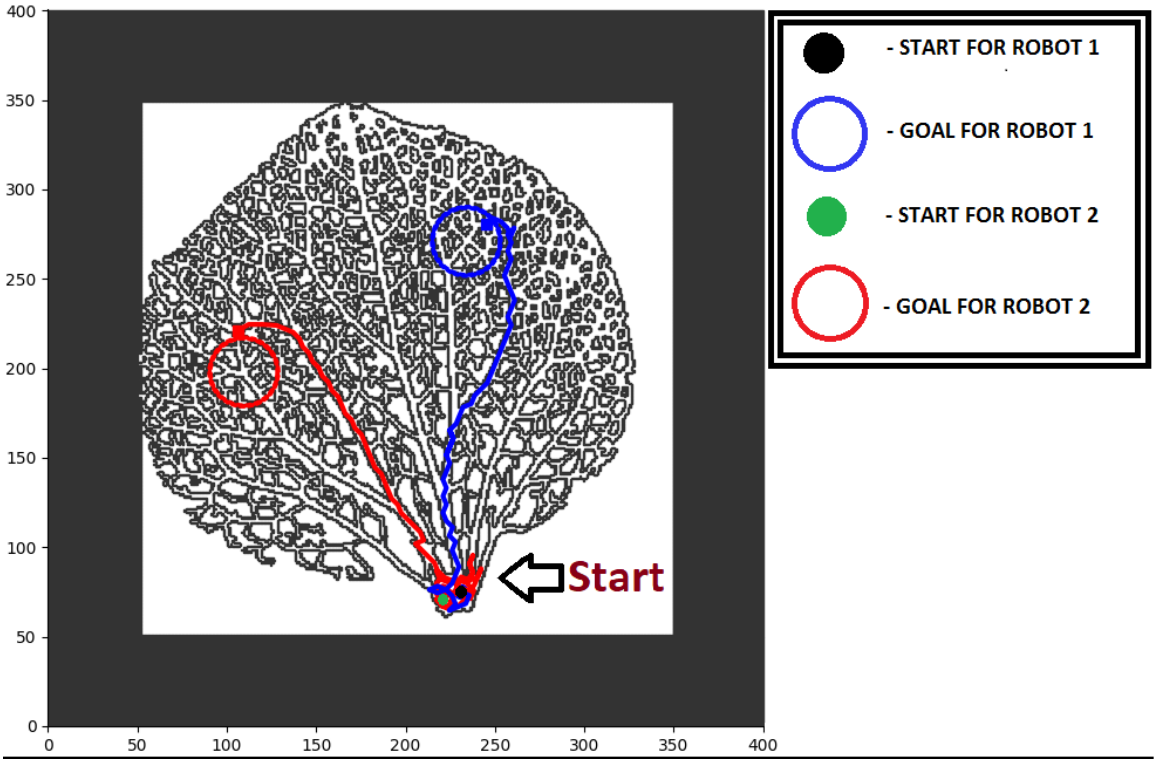
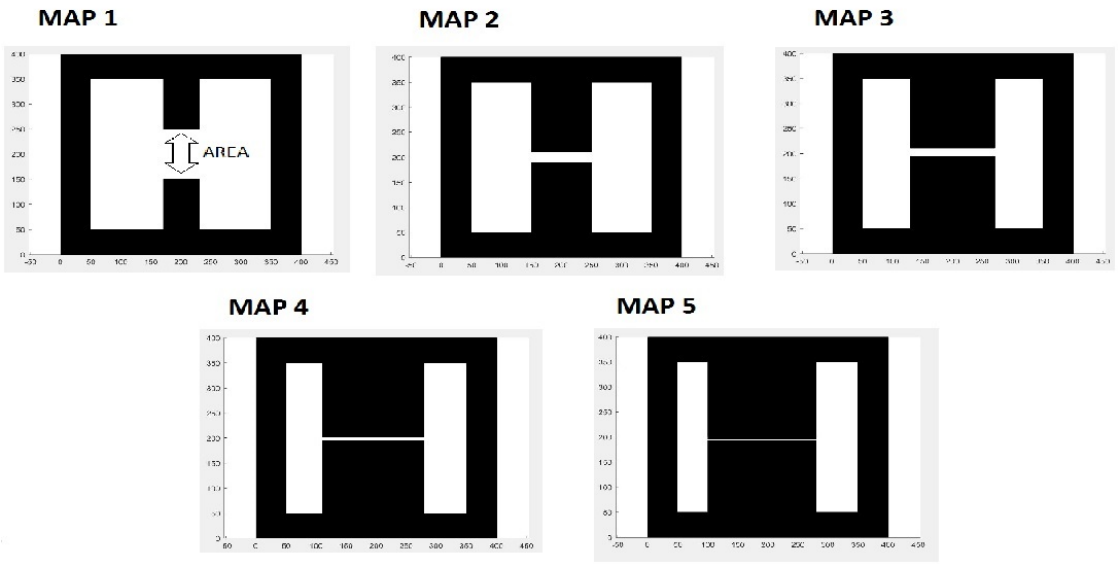


Figure 3.17: Solved environment 6 using our motion planner. This environment represents the leaf vascular network . The environment developed from Figure 3.16.

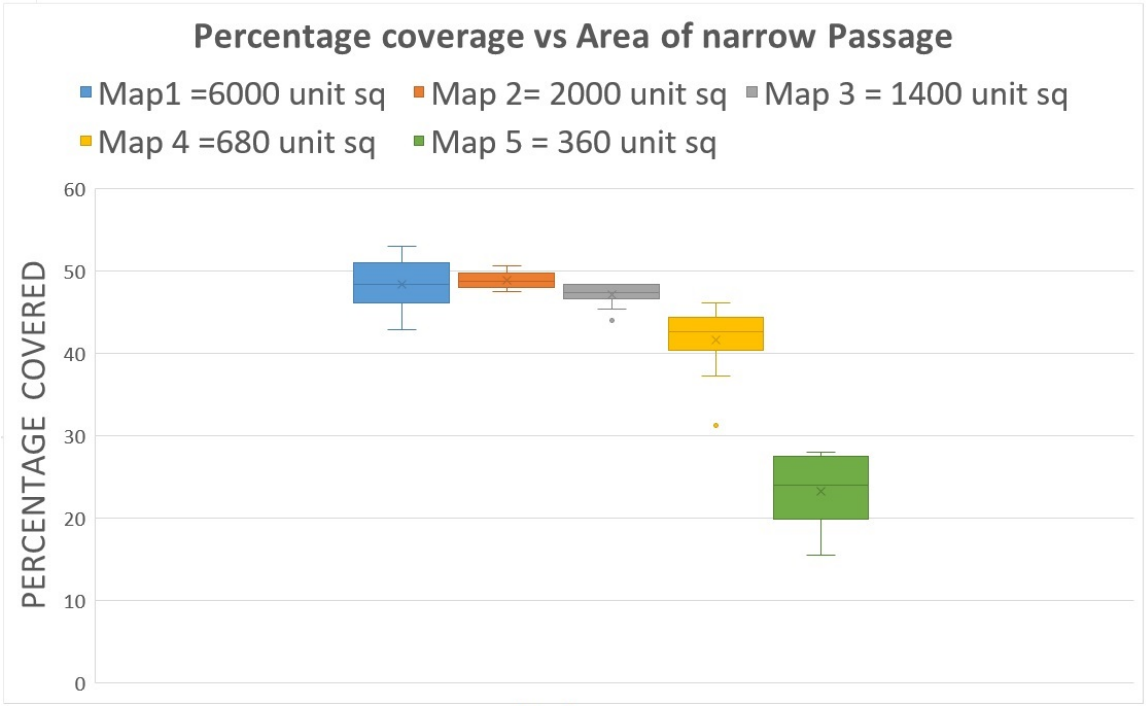
3.7 Varying Narrow Passage Gap Experiment

In this section studies the effect of decreasing the narrow passage width for the robots and studied the effects on the coverage. We performed this experiment with two robots keeping the same initial and goal positions for all five maps. Also, the number of nodes for the tree was set to 12500 nodes and the step size for both robots was set to 5 units. We kept all the other variables constant and just changed the size of the narrow passage and plotted the result on the map as shown in Figure 3.18 (b). The map used for this simulation is also shown in Figure 3.18 (a).

Figure 3.16 b demonstrates that when the narrow passage becomes more difficult for the planner to solve, the coverage of the map decreases. This suggests that we need to exploit the map more when we have a small narrow passage in our map. The coverage does not seem to have much effect on the Environment 1, 2 and 3 but for Environments 4 and 5, there is a drastic effect on the coverage.



(a)



(b)

Figure 3.18: (a) Maps used to explore the effect of narrow passage on coverage. (b) Plots for the coverage each of map in Figure 3.18(a).

Chapter 4

Application

4.1 Multi-robot Motion-planning

In this section, we generalize the implementation of our RRT planner to more than 2 robots. The time to solve an RRT increases exponentially with the number of the robots. In other words, the number of nodes required to find the solution for the group of robots increases according to Z^n, where n is the number of robots.

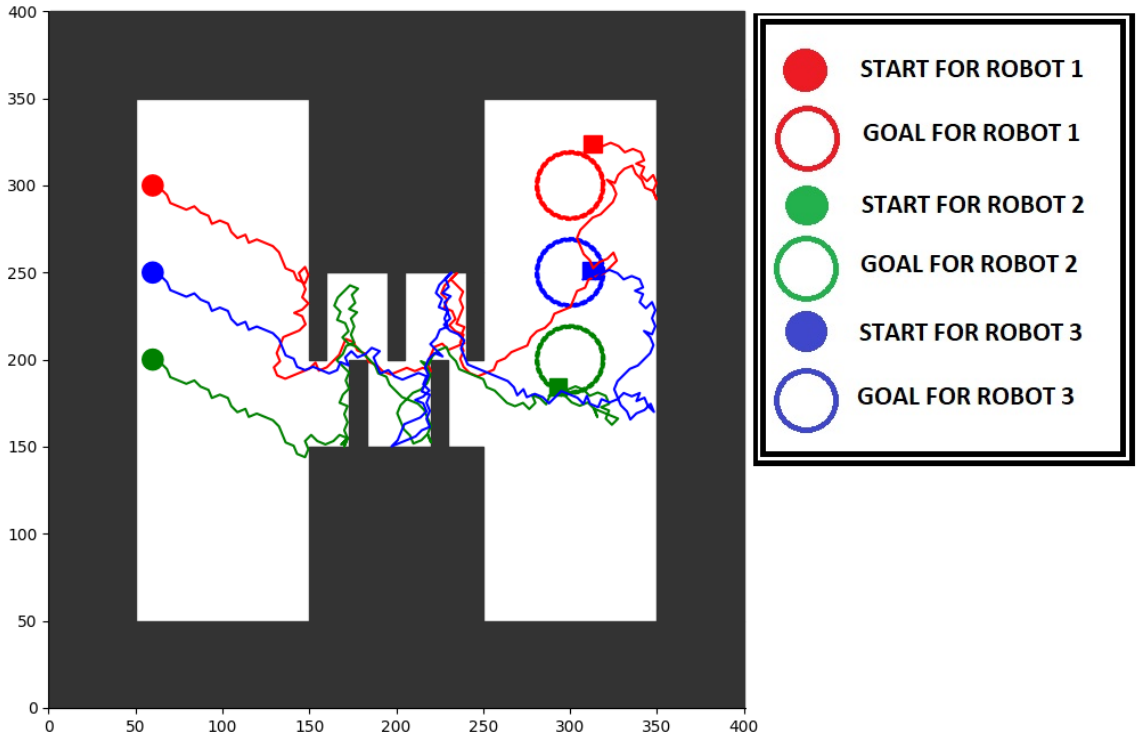


Figure 4.1: Solving the environment 4, Zigzag narrow passage, with 3 robots moving under global inputs.

We implemented the algorithm for 3 robots in environment 4-Zigzag narrow

passage, the same environment we used in earlier chapters. The result can be seen in Figure 4.1

As shown in Figure 4.2 we have four robots moving from their start to the goals and all the robots are moving under the global inputs. Every robot acts as a constraint for every other robot to reach the goal. Some of the functions are affected by the number of the robots. The planner in the RRT uses a distance function to find the shortest distance to the sampled points. To reach the goal, the distance function should satisfy the goal of each robot. Also, the grid function is directly proportional to the number of robots k^n, where n is the number of robots and k is the number of cells in one dimension. It is difficult to compute the coverage with the method we used because the memory required for the grid increases exponentially as the number of the robots.

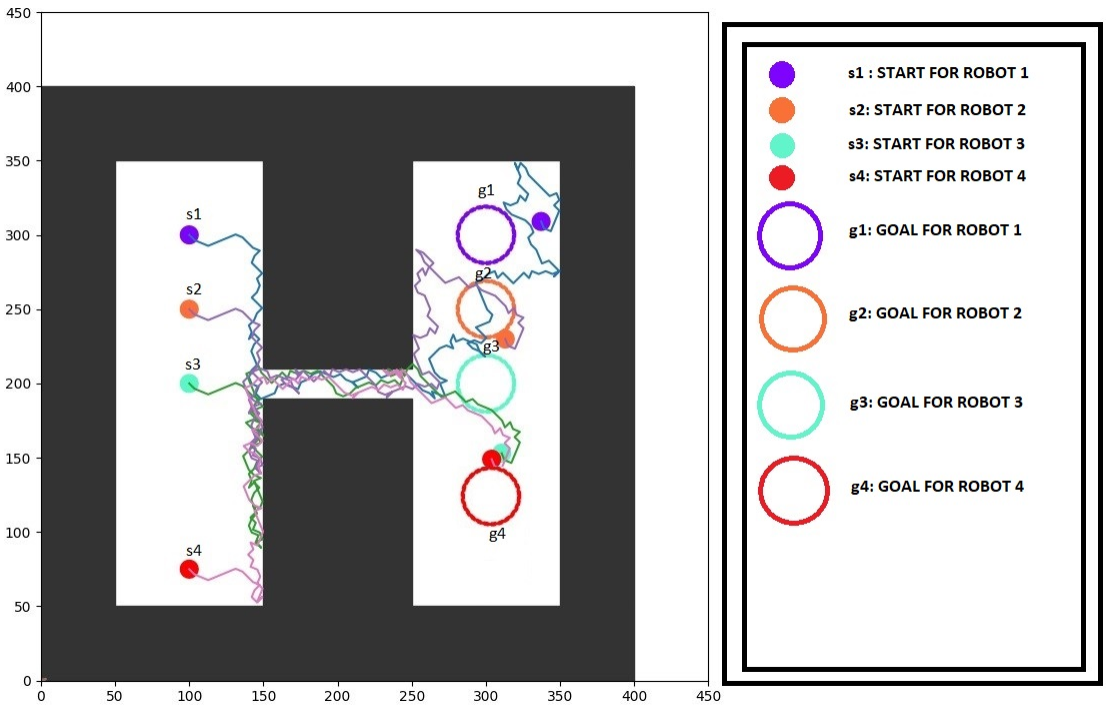


Figure 4.2: Solving the environment 3 long narrow passage with 4 robots moving under global inputs.

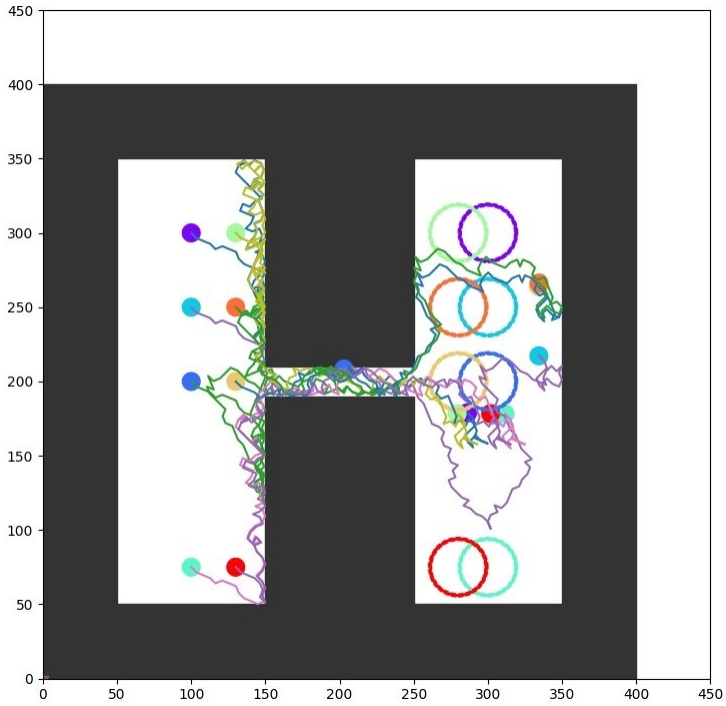


Figure 4.3: Solving the environment with 8 robots moving under global inputs

We did a similar experiment for 8 robots. As shown in Figure 4.2, it took a larger number of nodes for RRT to solve the environment for 8 robots because the degree of freedom of the robot increases. With increased number of nodes of the robots, it could not reach all the goals. We also solved environment 6 leaf vascular network for 3 robots as shown in Figure 4.4.

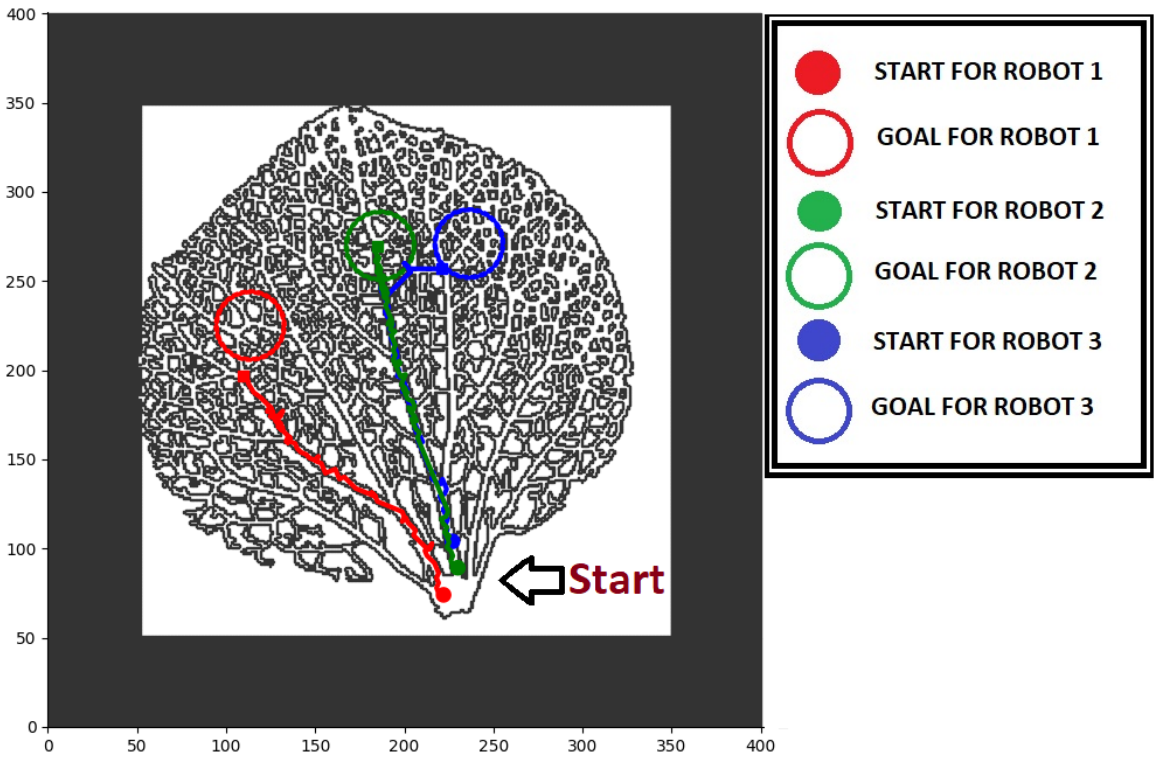


Figure 4.4: Solved environment 6 with 3 robots moving under global inputs.

4.2 Reuse of RRT Tree

To reduce the computation time for building a new RRT for every change in goal position, we want to reuse the RRT tree developed for the same environment with the same initial positions. Also, this method will allow us to reach different goals in the same environment. However, the percentage coverage of the map is an important factor as we cannot reach an area which is not explored by the RRT. All the experiments we performed in chapter 3 are helpful to give us insight into the question, How good is our RRT? We proved in earlier sections that parameters such as the number of nodes, RRT step size, narrow passage length, initial positions affect the coverage of the map. Mapping the paths for the environment and reusing it enables multiple queries using one tree. This allows us to quantify distances in this high dimensional space.

4.2.1 Swapping the Goal Positions of two Robots

We have used environment 2 and selected the start and the goal positions. We generate a RRT tree for this environment as shown in Figure 4.5. We then reuse the same RRT and swap the goal to reach the new goals as shown in Figure 4.6. From the Figure 3.9 we discovered that 10000 nodes and a step size of 40 to 80 is ideal to achieve almost 100 % coverage of this map.

As before, the environment 2 in Figure 4.5 and Figure 4.6 has obstacles in black. The start for the robots are shown in the pictures in black and green circles for robot 1 and 2 respectively. The goals for the robot 1 and robot 2 are shown as a blue and a red circle respectively.

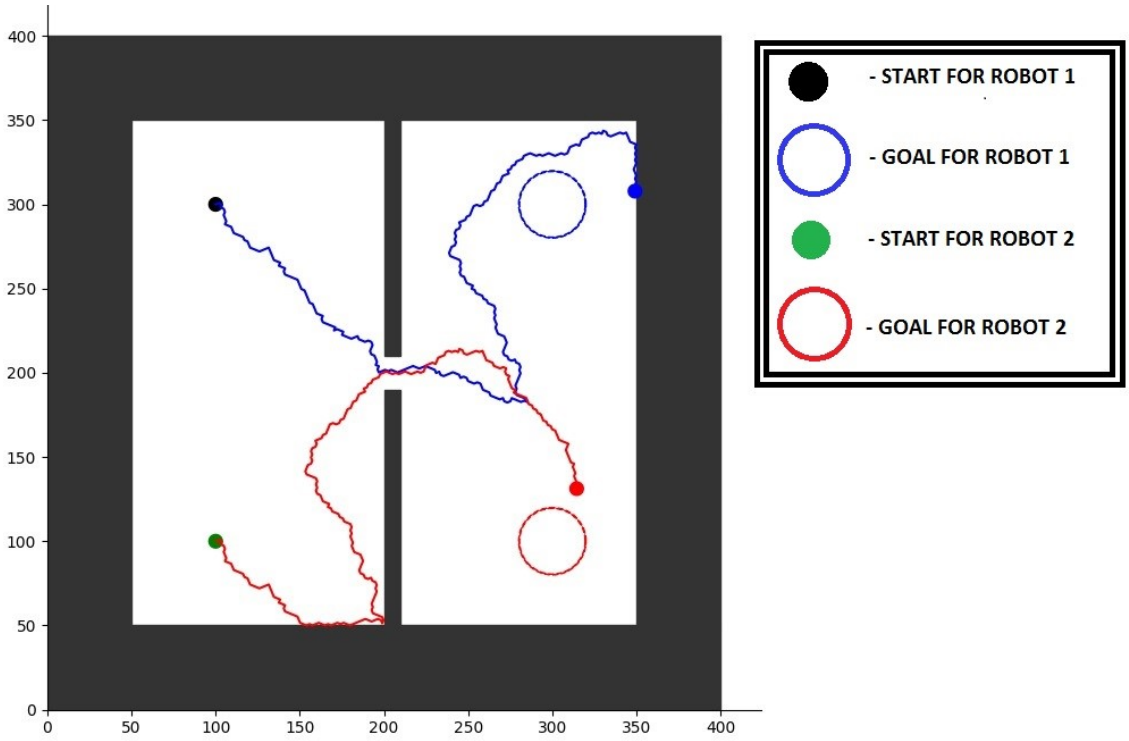


Figure 4.5: Solving RRT for two robots using global inputs for environment 2 with 10000 nodes.

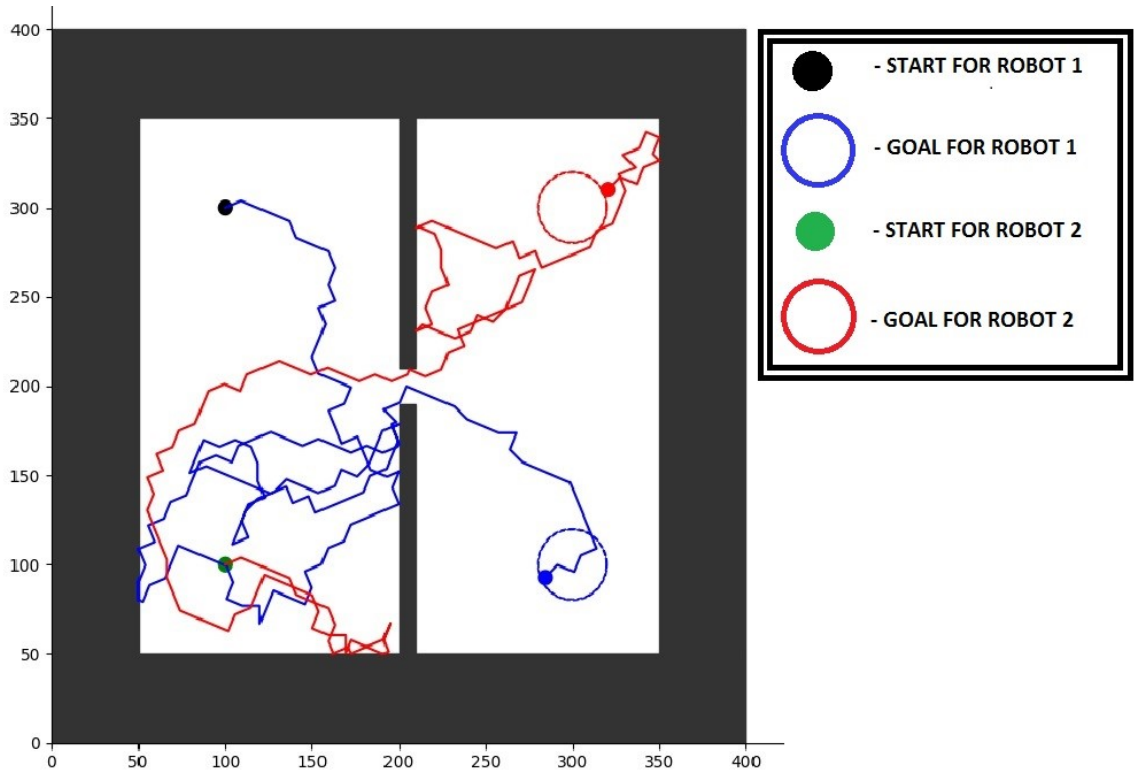


Figure 4.6: Reusing an RRT generated to reach the other goals for 2 robots.

4.2.2 Reusing an RRT and swapping the goal positions for multiple robots

To solve the problem for multiple robots, we ran the algorithm for a longer time to create a dense RRT tree. Once the tree is finished we can reuse it to solve multiple problems for 4 robots. As shown in the Figures 4.7 and 4.8 we solved the problem for 4 robots and reuse the developed tree to solve the problem for 4 robots in the same environment but with different goals.

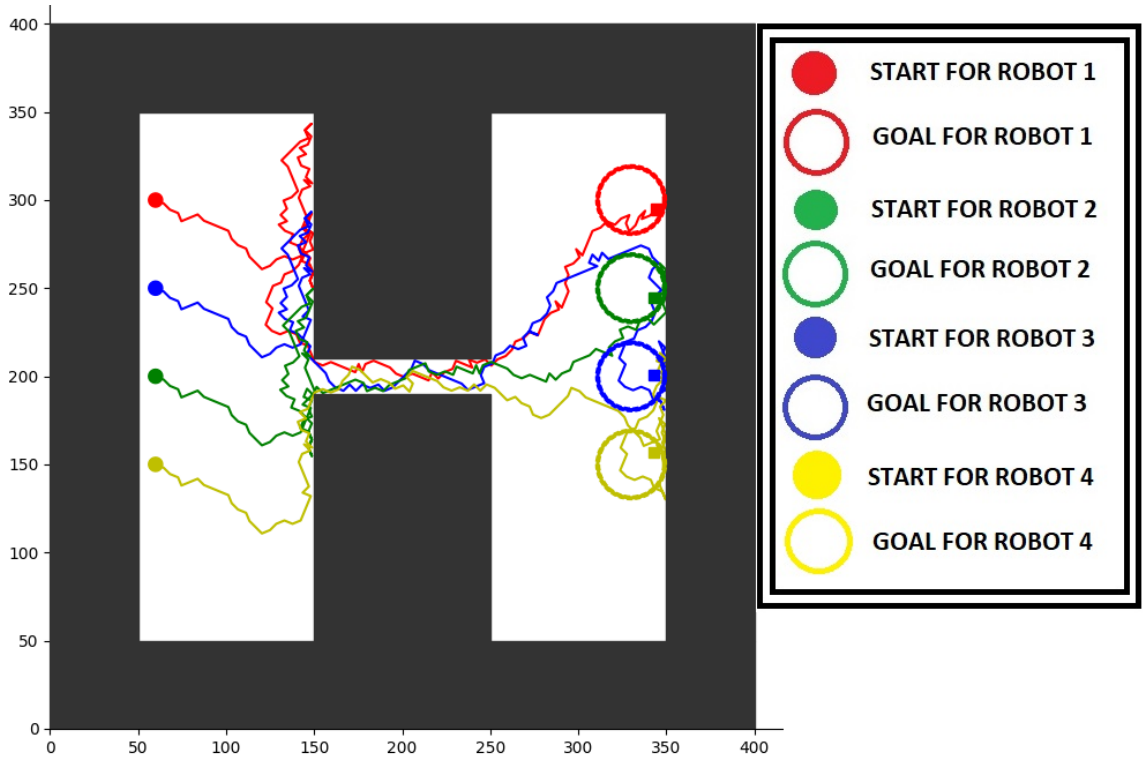


Figure 4.7: Solving environment 3 for four robots.

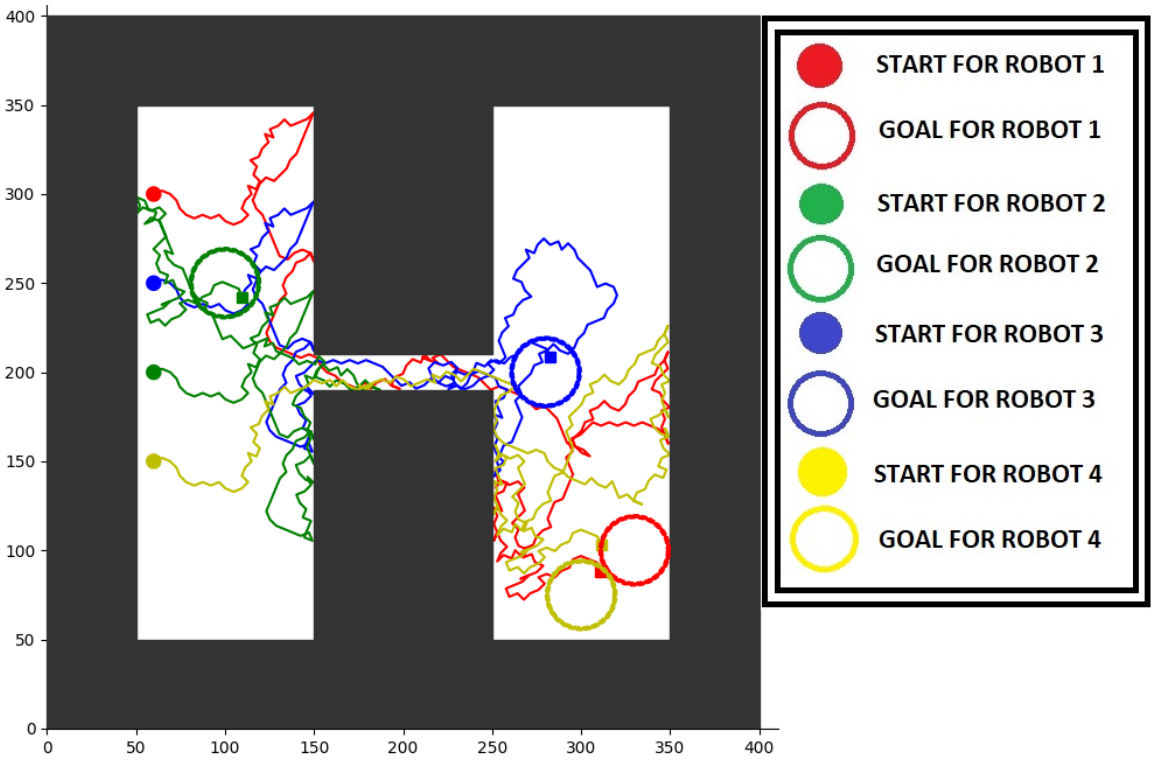


Figure 4.8: Reusing an RRT generated for four robots.

Chapter 5

Conclusions, Discussion and Future Work

5.1 Conclusion

Motion-planning for a swarm of robots is a challenging task when all the robots are moving under a global input. The motion performed by robots to reach from one state to another must be planned while considering other robots. The main goal of the thesis is to use a preexisting motion-planning algorithm to solve the motion-planning problem for a swarm of robots moving under a global input and study the effect of different parameters and environments on the coverage of the configuration space. The motion-planning becomes complicated when we consider a swarm of robots because the planning must be performed in a higher dimension and dimensions increase exponentially with the number of robots for a coupled approach (the so-called curse of dimensionality).

We use open source planners and libraries to create a motion planner to satisfy our goals. We used a Rapidly-exploring Random Tree (RRT) [12] for developing our motion planner. We modeled our robots as small particles in simulation and used rectangle-shaped obstacles to create the different environments. When moving under global inputs all the robots move in the same direction until their motion is obstructed by an obstacle. We also assumed that the robots can slide on the edges of the obstacles to create the simulation. Our first goal was to move the robots under global inputs and find a goal for individual robots from their start positions using our planner which is developed using RRT. We implemented RRT for a swarm of robots and successfully solved the motion-planning problem using the OMPL library.

Our second goal was to study the effect on the coverage of the map by changing one parameter and keeping another parameter constant. To study the map coverage, we divided the map into a grid, found the voxels filled by all the robots of the swarm, and calculated the percentage of filled voxels. We created the grid in $2n$ dimensions when we have n robots which can move in the X and Y direction. As the number of robots increases, the memory requirement for the grid increases exponentially. We tested changes in the parameters, including the step size of the robot, the number of nodes of the RRT tree, the grid size of grid we use to measure coverage, the initial positions of the robots, changing narrow passage width between the obstacles in the environment, and different environments and observed its effects on their map coverage. As RRT is based on a distance function from moving from one state to another, restricted the distance the robot can move at every sample (step size). We experimentally proved that the step size of the robot is directly proportional to map coverage for small step sizes. However, there are diminishing returns and the percentage coverage starts decreasing after a certain step size. We observed from the experiments that the motion planner can cover the maximum area when the robot step size fits most of the places in the environment, because it covers the maximum area without colliding the obstacles. We observe that as the number of nodes increases the coverage increases to a fixed value and saturates after a certain number of nodes. The effect of the grid size is proportional to the coverage. The result from the test of changing initial positions allowed us to find its relationship with the performances of the robots in terms of coverage. The coverage percentage drops drastically when robots are very near to each other. We also discovered that when the narrow passage width decreases, the coverage decreases. We extended our result on the parameters affecting RRT and discovered that we can reuse the RRTs for the same environment and have the same initial locations but different goals.

5.2 Extension

This thesis has opened new areas on the possibility of using available motion-planning tools for the swarm of robots with global controls. An observation which has been made during this research has provided new directions for future research which are recommended as follows:

5.2.1 Practical Implementation in the Lab

During this thesis, the entire research was performed using OMPL and simulations. However, the idea can be implemented using small iron particles and a magnetic manipulator or magnetic MRI scanner. This implementation could prepare for implementation of these ideas for practical use [21].

5.2.2 Alternative to RRT

We used RRT as our motion planner but have not compared other sampling-based planner such as PRM to solve this problem. We could use a planner called *DRRT* which will sample the environment using PRM and grow an RRT using PRM sampling points [22].

5.2.3 Simulation of 1000+ Robots

In this thesis, we are worked with coupled robots and implemented the program for 8 robots. The same can be implemented for 1000+ robots and to measure the coverage with more robots. However, a more efficient coverage measurement algorithm must be designed for the larger number of robots in the swarm [23].

5.2.4 Consideration of Velocity and Acceleration

During this thesis, we have not considered practical particle parameters like friction, velocity or acceleration of the robots. The same problem could be solved by using a control space function and differentiating the equations.

5.2.5 Considering 6 Degrees of Freedom Per Robot

The robots described in this thesis only move in X and Y direction, so the entire thesis uses 2D workspaces. However, robots can move in 3 planar directions X, Y and Z. Also, the robot can have roll, pitch, and yaw movement. We could develop a different sampling method to increase exploration and exploitation of the space (For example using a genetic algorithm and finding the optimum step).

5.2.6 Extend to Different Shapes of the Robot

The robot we considered is circular but the same algorithm can be implemented by using different shapes. It is an open problem to discover which shape works the best.

5.2.7 Map Coverage Algorithm

The method used in this thesis for measuring coverage can be improved. Currently, we divide the space in the grids in the dimension $2n$, n is number of robots in the swarm (considering all the robots are in $2D$). We can improve this algorithm by using the concept of Voronoi bias [24].

5.2.8 Use Microorganism as a Robot Using our RRT

When seeding a culture tube containing the microorganism called $T. pyriformis$ and iron oxide nanoparticles, we can create a steerable robot that responds to the magnetic field as per [25]. Future work could extend our thesis idea for swarms of microorganisms and practical implementation in a biological system.

References

[1] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic Roadmaps for Path Planning in High-dimensional Configuration Spaces," in *IEEE Trans. Robot. Automat.*, Aug 1996, pp. 566–580.

[2] I. A. Sucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," in *IEEE Robotics and Automation Magazine*, 2012, pp. 72–82.

[3] S. M. LaValle, "Planning Algorithms," in *Cambridge University Press*, 2006.

[4] I. Paprotny and S. Bergbreiter, "Small-Scale Robotics: From Nano-to-millimeter-sized Robotic Systems and Applications," in *ICRA*, 2013.

[5] B. J. Nelson, I. K. Kaliakatsos, and J. J. Abbott, "Microrobots for Minimally Invasive Medicine," in *Annual Review of Biomedical Engineering*, 2010.

[6] Y. TAN and Z. yang ZHENG, "Research Advance in Swarm Robotics," in *Defence Technology*, vol. 9, 2013, pp. 18–36.

[7] D. J. Cappelleri, M. Fatovic, and Z. Fu, "Caging Grasps for Micromanipulation Microassembly," in *International Conference on Intelligent Robots and Systems*, 2011.

[8] A. Das, P. Zhang, W. Lee, D. Popa, and H. Stephanou, "μ^3 : Multiscale Deterministic Micro-nano Assembly System for Construction of On-wafer Micro-Robots," in *International Conference on Robotics and Automation (ICRA)*, 2007.

[9] A. Chanu, O. Felfoul, G. Beaudoin, and S. Martel, "Adapting the Clinical MRI Software Environment for Real-time Navigation of an Endovascular Untethered Ferromagnetic Bead for Future Endovascular Interventions," in *Magn Reson Med*, 2008, pp. 1287–1297.

[10] P. Vartholomeos, M. Akhavan-Sharif, and P. E. Dupont, "Motion Planning for Multiple Millimeter-scale Magnetic Capsules in a Fluid Environment," in *IEEE International Conference Robotics and Automation*, 2012, pp. 1927–1932.

[11] P. Vartholomeos, C. Bergeles, L. Qin, and P. E. Dupont, "An MRI-powered and Controlled Actuator Technology for Tether-less Robotic Interventions," in *Int. J. Rob. Res*, 2013, pp. 1536–1552.

[12] S. M. LaValle, "Rapidly-exploring Random Trees: A New Tool for Path Planning," in *TR 98-11, Computer Science Dept., Iowa State University*, Oct 1998.

[13] A. T. Becker, O. Felfoul, and P. E. Dupont, "Simultaneously Powering and Controlling Many Actuators with a Clinical MRI Scanner," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, IL, USA*, 2014, pp. 2017–2023.

[14] S. Shahrokhi and A. T. Becker, "Object Manipulation and Position Control Using a Swarm with Global Inputs," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2016.

[15] C. Rodriguez, J. Denny, S. A. Jacobs, S. Thomas, and N. M. Amato., "Blind RRT: A Probabilistically Complete Distributed RRT," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[16] M. W. Spong, S. Hutchinson, and M. Vidyasagar, "Robot Modelling and Control," in *Robot Modeling and Control*, 2004.

[17] T. Lozano-Perez and M. A. Wesley, "An Algorithm for Planning Collision-free Paths among Polyhedral Obstacles," in *Communications of the ACM*, Oct 1979, pp. 560–570.

[18] A. T. Becker, "Controlling Swarms of Robots with Global Inputs: Breaking Symmetry," in *Microbiorobotics: Biologically Inspired Microscale Robotic Systems by Ed. M.J. Kim, A.A. Julius, and U.K. Cheang, Elsevier*, 2017, pp. 3–20.

[19] A. Atramentov and S. M. LaValle, "Efficient Nearest Neighbor Searching for Motion Planning," in *Proceedings IEEE International Conference on Robotics and Automation*, 2002, pp. 632–637.

[20] A. Yershova and S. M. LaValle, "Improving Motion Planning Algorithms by Efficient Nearest-neighbor Searching," in *IEEE Transactions on Robotics*, 2007, pp. 151–157.

[21] S. Shahrokhi, J. Shi, B. Isichei, and A. T. Becker, "Exploiting Non-Slip Wall Contacts to Position Two Particles Using the Same Control Input."

[22] K. Solovey, O. Salzman, and D. Halperin, "Finding a Needle in an Exponential Haystack: Discrete RRT for Exploration of Implicit Roadmaps in Multi-robot Motion Planning," in *The International Journal of Robotics Research*, 2016, pp. 501–513.

[23] A. Shkolnik and R. Tedrake, "Path Planning In 1000+ Dimensions, Using a Task-Space Voronoi Bias," in *IEEE International Conference on Robotics and Automation*, 2009.

[24] S. R. Lindemann and S. M. LaValle, "Incrementally Reducing Dispersion by Increasing Voronoi Bias in RRTs," in *IEEE International Conference on Robotics and Automation*, 2004.

[25] P. Seung, S. Kim, A. T. Becker, Y. Dal, H. Kim, A. A. Julius, and M. Kim, "Magnetic Swarm Control of Microorganisms," in *Microbiorobotics: Biologically Inspired Microscale Robotic Systems, Second Edition*, 2017, pp. 221–243.