DECENTRALIZED, NONCOOPERATIVE MULTIROBOT PATH PLANNING WITH SAMPLE-BASED PLANNERS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

William Le

COMMITTEE MEMBERSHIP

TITLE:	Decentralized, Noncooperative Multirobot Path Planning with Sample-Based Planners
AUTHOR:	William Le
DATE SUBMITTED:	March 2020
COMMITTEE CHAIR:	Xiao-Hua Yu, Ph.D. Professor of Electrical Engineering
COMMITTEE MEMBER:	Andrew Danowitz, Ph.D. Professor of Electrical Engineering
COMMITTEE MEMBER:	Joseph Callenes-Sloan, Ph.D. Professor of Electrical Engineering

ABSTRACT

Decentralized, Noncooperative Multirobot Path Planning with Sample-Based Planners

William Le

In this thesis, the viability of decentralized, noncooperative multi-robot path planning algorithms is tested. Three algorithms based on the Batch Informed Trees (BIT*) algorithm are presented. The first of these algorithms combines Optimal Reciprocal Collision Avoidance (ORCA) with BIT*. The second of these algorithms uses BIT* to create a path which the robots then follow using an artificial potential field (APF) method. The final algorithm is a version of BIT* that supports replanning. While none of these algorithms take advantage of sharing information between the robots, the algorithms are able to guide the robots to their desired goals, with the algorithm that combines ORCA and BIT* having the robots successfully navigate to their goals over 93% for multiple environments with teams of two to eight robots.

ACKNOWLEDGMENTS

Thanks to:

- Dr. Xiao-Hua (Helen) Yu
- Dr. Andrew Danowitz
- Dr. Joseph Callenes-Sloan
- Ji Jin
- Genevieve Duchesneau
- Cal Poly Robotics

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

This thesis will focus on the problem of multi-robot path planning. Multi-robot path planning consists of determining paths for robots in a team to maneuver to their goal locations to accomplish their higher level task in a way such that they do not collide with each other or any element in the environment.

Multi-robot path planning algorithms can be grouped into two subgroups: centralized algorithms and decentralized algorithms. Centralized algorithms use one agent, which can be an outside computer or team leader robot, to determine the paths for all the robots in the team [16]. This method results in plans that are more optimized in terms of cost to travel, but they have the drawback of being more computationally intensive. Additionally, if the problem has more robots, the dimensionality of the centralized planning problem increases exponentially, which leads to the algorithm being unable to solve the problem in a reasonable amount of time. Decentralized algorithms allow for each robot in the multiagent team to plan their own paths and then rely on communication between robots to have them coordinate with each other in order to avoid each other [21].

This approach results in quicker plan generation with the drawback of less optimized paths. Additionally, another consideration for multi robot planning algorithms is if the robots are cooperative or noncooperative [18]. A cooperative algorithm has the team of robots sharing more information about themselves such as their position and their planned path. In a noncooperative algorithm, less information is shared,

as robots will only know the qualities of the other robots through their individual sensing capabilities.

Another consideration is for planning in continuous space where graph-based planning methods or sampling-based methods can be used. Graph-based planning methods are more capable of finding optimal solutions, but the space must be rigidly discretized in order for the graph search algorithms to be applied [16]. Sampling-based planning methods can better handle continuous space since they rely on randomly sampling the space for valid waypoints for the path [16].

In order to efficiently solve a multi robot path planning problem, the algorithm selected must be fast in order for the robots avoid collisions with each other. Additionally, for problems with more robots, slower techniques may become infeasible. Therefore, this thesis will use a decentralized planning method with each robot planning their paths with a sample-based path planning method. While this methodology would result in suboptimal plan, the multi robot path planning problem is known to be a PSPACE-hard problem, so feasible solutions will be good enough at best [8].

There has been some research into using a decentralized sampling-based algorithm. One such algorithm is known as DMA-RRT or Decentralized Multi-Agent Rapidly-exploring Random Tree which uses a token passing system to have robots communicate and improve their plans with the other robots in the team [4]. Another approach uses a reactive technique and is known as ORCA-RRT* [9]. This technique combines a reactive technique ORCA or Optimal Reciporcal Collision Avoidance with a sampling-based algorithm, RRT* which is an asymtopically optimal version of RRT. RRT* is used as the basis for planning all the paths for all the robots. The algorithm then amends the paths to make them collision-free by using ORCA to simulate collision-free velocities.

The approach taken in this thesis is similar to the ORCA-RRT* approach in that it uses a sampling-based planner in conjunction with ORCA, but it has each robot planning their own paths. The sample-based planning algorithm used will be Batch Informed Trees or BIT* since it employs incremental search techniques and can be run in an anytime fashion [6]. Additionally, other variations of the technique are tested such as using artificial potential fields to allow the robots to react and avoid the other robots and replanning the robot's path.

The contributions of this thesis in the form of the various approaches used for this thesis. First is the use of BIT* with the combination of reactive techniques. Even though the combination of using a sample-based planning (SBP) algorithm with a reactive technique is not a new idea, BIT* has not been used for this purpose before [9]. Second is the decentralized formalization of ORCA with an SBP. Previous combinations of sample-based planning and ORCA used a centralized formalization [9]. Third is the application of the combinations of SBPs with APF to be applied to a multi-robot setting instead of a dynamic setting. Fourth is the application of replanning with multi-robot problems. Fifth is a version of BIT* that supports replanning.

Following this introduction is Chapter 2 which goes into more detail about other algorithms used for multi-robot path planning problem. Chapter 3 introduces and develops the basis for the approach taken by this thesis by discussion single query sample-based planning and reactive planning algorithms. Chapter 4 describes the approach in depth and presents the results of taking the approach in various simulated environments. Chapter 5 states the conclusions from this thesis and potential future work.

Chapter 2

LITERATURE REVIEW

The problem of multi robot path planning has been addressed by many other techniques. In this section, centralized approaches, prioritized search, and reactive methods are discussed and evaluated. Some of these methods are not explicitly for multi robot path planning, but have been shown to work in environments with moving obstacles since multi robot path planning can be seen as a similar problem to planning in dynamic environments.

2.1 Centralized Multi Robot Planning

Centralized multi-robot planning algorithms rely on a centralized computer to determine the paths for all robots in the configuration space. This centralized computer can be a source outside of the robots in the space or a leader robot in the space.

2.1.1 M*

M* is a multi robot path planning algorithm that is based on what the algorithm refers to as subdimensional expansion [21]. Multi robot planning can be considered as a higher dimensional planning problem if each robot is considered as a joint.

For example, a planning problem for two robots in two dimensions can be considered as a four dimensional problem since the configurations for the problem includes the positions for both robots. So, the dimensionality of a multi robot planning problem grows exponentially which makes the problem very difficult to solve. Subdimensional

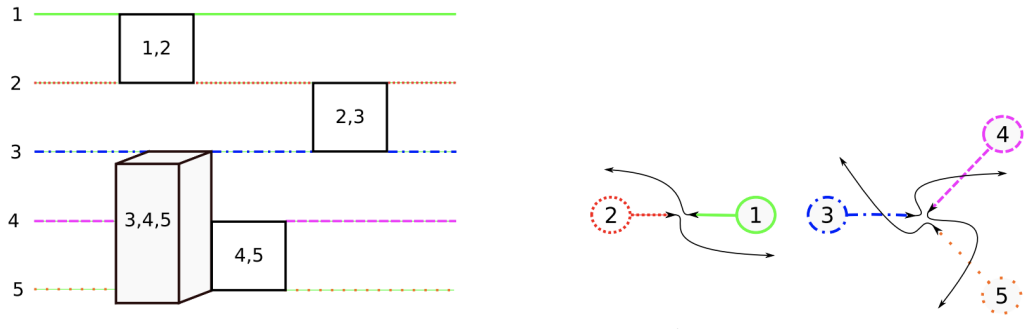


Figure 2.1: These images illustrate the concept of subdimensional expansion. The dimensionality for the problem involving each robot is represented as a line. The squares and cubes represent the problem when the robots get into a near collision state. For example, the square of 1, 2 is an instance of where robots 1 and 2 are in a near collision which represents the problem going from a one dimensional problem to a two dimensional one[21].

expansion mitigates this problem by only expanding the dimensionality of the search space when collisions between robots are found to occur[21].

The basic workflow of solving a planning problem with subdimensional expansion is to allow for plans to be generated for each individual robot until a robots come into close contact with each other. When this occurs, a subproblem in the area of conflict with a higher dimensional search space is generated as illustrated in Figure 2.1.

This selective expansion of the search space minimizes the dimensionality of the search space which makes eases the difficulty of the problem and increases the chances of generating a feasible solution.

M* implements this strategy by using the A* search algorithm to plan paths for the individual robots and tracking where collisions would occur [21]. In those collision areas, the algorithm allows the robots to deviate from their plans by resolving the conflict by solving a higher dimensional path planning problem. Additionally, there has been an extension of the subdimensional expansion method for sample-based

planning methods like rapidly-exploring random trees and probabilistic roadmaps in place of A* [22].

In general, the subdimensional expansion method has been found to find minimal cost solutions for multi robot planning problems, but the method has been found to struggle with finding solutions with problems that have more than around ten robots within twelve minutes.

2.1.2 Rapidly-exploring Random Graphs (RRG)

The rapidly-exploring random graph, RRG, algorithm combines ideas from RRT, PRM or probabilistic road maps, and prioritized planning to solve the multi robot path planning problem [10] [12]. The algorithm has an exploration and an exploitation stage. In the exploration stage, an RRT is grown from all the starting configurations for the agents, then the individual RRTs are joined when they grow into areas in proximity of the other trees. After a stopping criterion is reached, the resulting roadmap of the space is finalized and paths are produced for each robot. These paths are generated are in a prioritized fashion where each robot is given an individual path. This plan has taken the other generated paths into account during its planning phase. This process is show in Figure 2.2

This method was not tested with more than four robots, so it is not stated how well this algorithm would scale. But, it is likely to face similar problems to those found with the subdimensional expansion based algorithms since it has all agents in the system using the same roadmap which could result in difficult to resolve deadlocks.

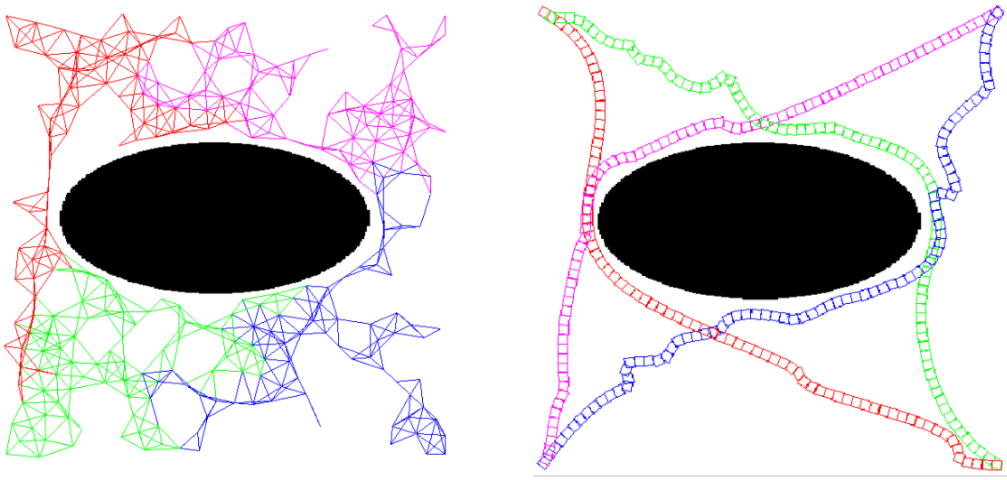


Figure 2.2: The image on the left shows a rapidly-exploring random graph. The graph is created from RRTs from each starting configuration. The graph is colored to reflect where each tree grew from. The image on the right shows paths generated for the robots by traversing the graph [10].

2.1.3 ORCA-RRT*

ORCA-RRT* uses a multi robot variant of RRT* to generate paths for all the robots in the system [9]. The multi robot variant of RRT* treats each robot in the system as a joint and finds a path in the higher dimensional search space that connects all the initial positions of the robots to their goal positions. Individual paths are drawn from this solution and converted into trajectories for each robot. These trajectories are input into the ORCA algorithm to generate a safe trajectory for each robot up to a certain number of timesteps to avoid generating deadlocks between robots. The technique results in a high success rate of all robots in the team finding feasible, near-optimal trajectories in a relatively quick fashion.

2.2 Decentralized Multi Robot Planning

Decentralized multi-robot planning algorithms have the multiple robots plan paths in the configuration space. They do not rely on a single computer to determine all the paths for the robots in the space, the robots themselves are capable of doing that themselves. After planning their paths, the robots must be able to communicate their intentions in order to avoid collisions with the other robots to allow them to adjust their paths appropriately.

2.2.1 DMA-RRT

Decentralized Multi-agent RRT is a multirobot planning algorithm based on CL-RRT or closed-loop RRT and a merit-based token system [4]. CL-RRT is a variant of RRT that forward simulates the motion of the robot using a dynamic model of the robot in order to generate feasible trajectories. The merit-based token system consists of the token passing algorithm and waypoint passing.

Every robot in the team plans their own paths with CL-RRT and maintain the search tree they used to generate their individual paths. On initialization, a robot is randomly assigned a token which indicates it has planning priority. This robot broadcasts the waypoints to its path to all the other robots. The other robots take the waypoints and simulate the path of the token owning robot and amend their internal map of the environment to account for that path. Each robot is continually improving their paths by expanding their internal search tree.

The robots that do not own the token bid submit a bid for the token with the value equal to how much they have improved their paths. The token owning robot listens

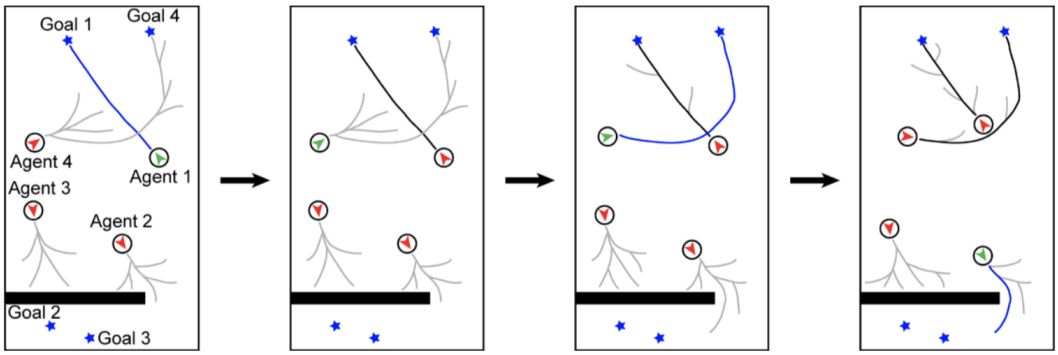


Fig. 3 Example scenario illustrating merit-based token passing

Figure 2.3: An example of the token passing system in use for DMA-RRT. In this scenario, agent 1 is given the token to begin with and plans a path to its goal. Then, agent 4 gains the token since it is the next best potential path and it plans its path to goal. After that, agent 2 is the agent that has improved its path the most, so it receives the token [4].

to all the bids and relinquishes the token to the robot that has improved its path the most. This overall process repeats until all robots reach their goal configurations.

This take on prioritized planning rewards the robots that improve their paths the most in order to attempt to have all robots reach their goal configurations as soon as possible. The token passing process is illustrated in Figure 2.3.

For this algorithm's evaluation, the metric used was number of goals reached by the agents within a certain timespan. The merit-based token method was shown to have improved the rate of goals being reached by 20% when compared to a round robin token passing method since by giving priority to the robots that have improved their path the overall distance the robots must travel is reduced.

This algorithm mostly demonstrates the usage of sample-based planning in a real world multi robot implementation. One key issue with the algorithm is that it requires over air communication which can sometimes be unreliable.

2.3 Dynamic Environment Planning

In this section, planning algorithms for dynamic environments are explored. Dynamic environments present a difficult problem for planners since the paths they generate will have to be altered in order to account for changes in the environment. The approaches below mitigate that problem by using gradient-based methods to adapt quickly.

2.3.1 Path-Guided APF-SR

Path-guided APF-SR is dynamic path planning method that is capable of navigating complex environments with dynamic obstacles [3]. It is a combination of three ideas: sample-based path planning, artificial potential fields, and stochastic reachability sets.

The algorithm uses a sample-based path planner to construct a path in the initial environment. Stochastic reachability sets for each kind of dynamic obstacle are pre-calculated and are used to define the probabilities of where the dynamic obstacles will be moving to. These sets are calculated based on a dynamic model of the obstacles. From there, an artificial potential field technique is used to guide the robot towards the goal orientation.

This technique varies from the usual technique of having a goal and obstacle gradients by adding attractive gradients to the next waypoint in the path and to the line between the last waypoint and the next waypoint. The stochastic reachability set is used to determine the obstacle gradient for the dynamic obstacles. While this method has been shown to be very successful in the complex environment it is tested in, it has not been applied to multi agent path planning, but it seems promising since it is capable of handling many obstacles. A potential drawback of the algorithm is that it relies on

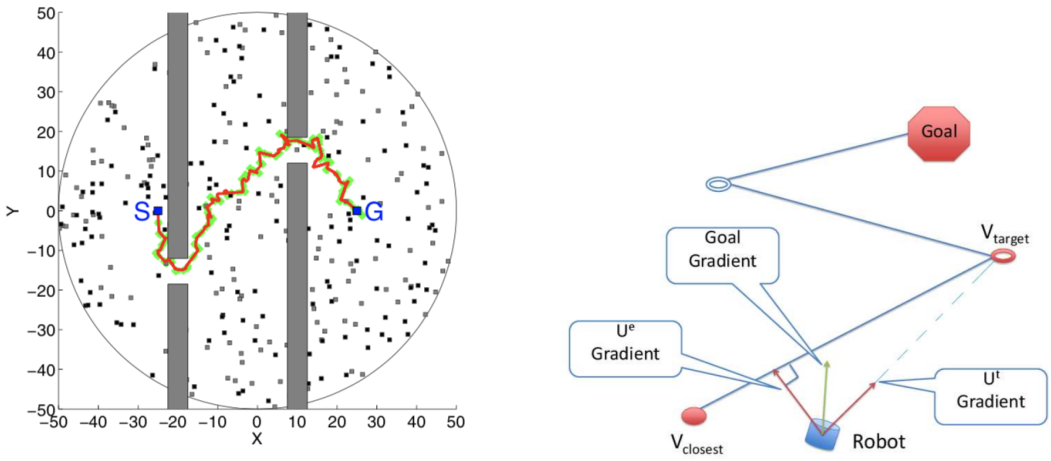


Figure 2.4: The image on the left shows the path generated by a sample-based planning algorithm in green and the path traversed by the robot in green. The image on the right illustrates the calculation of the attractive gradients for path-guided APF [3].

knowing the dynamics of the dynamic obstacles in the environment. Figure 2.4 shows the a robot traversing an environment with path-guided APF-SR and an explanation of how the artificial potential field algorithm works.

2.3.2 Dynamic APF

This algorithm aims to have an unmanned aerial vehicle, or UAV, be capable of navigating a dynamic environment with a moving goal [2]. In order to accomplish this goal, the algorithm amends the classical formulation of the APF algorithm to handle the dynamic obstacles and the moving goal.

In order to handle the moving goal, the attractive potential for the drone is the weighted square of UAV's distance to the goal. The repulsive potential from obstacles to the UAV are defined with a common formulation for the repulsive potential. Finally, there is a coordination force that pushes the drone in the forward direction

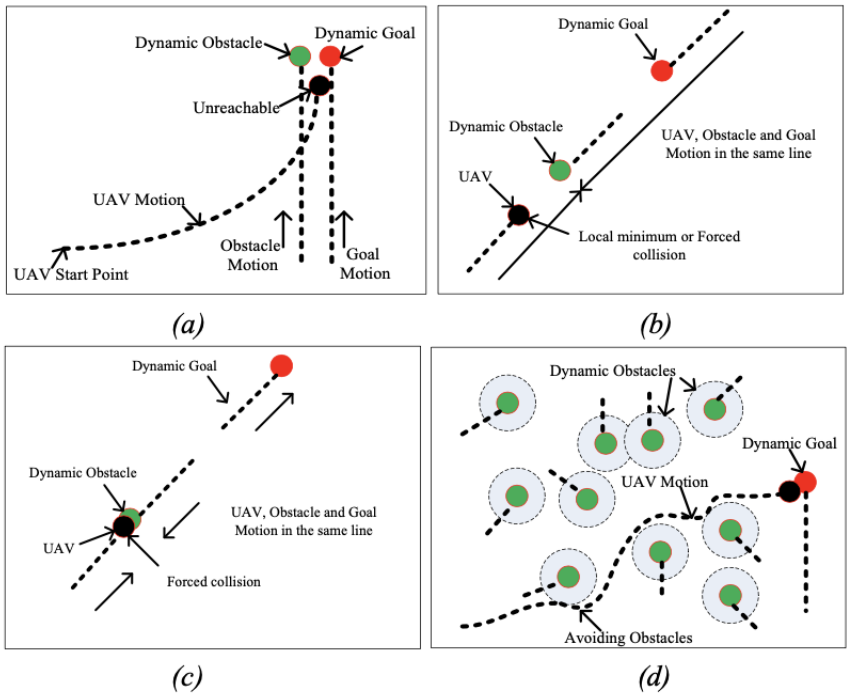


Figure 2.5: This figure shows the various cases a UAV would see moving in an environment with dynamic obstacles and dynamic goal locations [2]

and to the right. The magnitude of this force is proportional to the distance to the goal and the total repulsive magnitude. This is shown in Figure 2.5

This algorithm is shown to significantly improve the UAV's ability to navigate the dynamic environment. But, it has not been tested in multirobot systems or in environments with static obstacles in addition to dynamic ones.

Chapter 3

BACKGROUND INFORMATION

In this chapter, the basis for the approaches used in this thesis is discussed. First, path planning algorithms are observed and evaluated in order to determine the method used to plan the initial path for the robots. Afterward, reactive algorithms are examined in order to list the ways others have dealt with dynamic environments.

3.1 Path Planning Algorithms

In this section, algorithms that plan paths from a starting configuration to a goal configuration in a static environment are explored. These algorithms are split into two categories: graph-based and sample-based methods.

3.1.1 Graph-Based Planning

Graph-based planning uses graph search algorithms in order to plan a path for a robot. These algorithms usually start by discretizing the robot's environment into a grid and then running a search algorithm. The resulting path is the best path taken by the search algorithm.

3.1.1.1 A*

The A* algorithm is one of the most famous and most utilized search algorithms. It has been applied to the path planning problem and has served as the basis for many

other path planning algorithms. Its widespread use can be attributed to it's relative efficiency and intuitiveness.

A* is a greedy, heuristic-based search algorithm that uses a cost-so-far and an estimated cost-to-go in order to determine the best nodes in a graph to expand in order to find the goal [7].

The algorithm starts by placing the node that contains the robot's starting position into a priority queue. While this queue contains nodes or the goal configuration has not been reached, A* uses the lowest cost node in the queue to search. This is done by popping the node from the top of the queue and either updating the cost of the nodes in the queue that neighbor the lowest cost node or enqueuing neighboring nodes that have not been enqueued previously. The adjusted nodes will then identify the lowest cost node as their parent node.

A* executes this search by using the cost function shown in Equation 3.1. This cost function reduces the size of the search space for the shortest path which leads to the algorithm running faster.

$$f(x) = g(x) + h(x) \tag{3.1}$$

The cost to get from the current node from the start is denoted as g(x). Whereas, the cost to get from the current node to the node is denoted as h(x).

When the goal is found, the path to the starting configuration is found by backtracing the parent nodes from the goal to the start.

A* is able to solve path planning problems for single robots in a static environments, but it struggles in dynamic environments since the algorithm would not be able to account for the changes to the environment. As a result, the only way for the

algorithm to repair the robot's path is to recalculate the path from the start. This issue can be alleviated with the use of incremental search techniques which efficiently account for changes in the environment by reusing information from their previous search to repair the path in the new environment [13].

3.1.2 Sample-Based Planning

While the previous graph-based search methods [7] found optimal solutions to path planning problems, they have problems with providing a solution quickly especially in larger or higher dimensional environments since the search space becomes large. To combat this, randomized strategies are used. These randomized strategies became known as sample-based path planning since they randomly sample the configuration space to find a solution to their path planning problems.

There are two kinds of sample-based planning algorithms: single-query and multi-query algorithms. Single-query planners solve one path planning problem and will need to be completely rerun in order to solve another path planning problem in the same configuration space. Multiple-query planners are able to solve multiple path planning problems in the same configuration space. For this thesis, the discussion is limited to single query planning algorithms.

3.1.2.1 Rapidly-Exploring Random Trees (RRT)

The first single query sample-based planning algorithm discussed is the rapidly-exploring random tree or RRT. The algorithm, as shown in Algorithm , begins by adding the starting point to the search tree. Then, the algorithm begins its search until it finds the goal configuration.

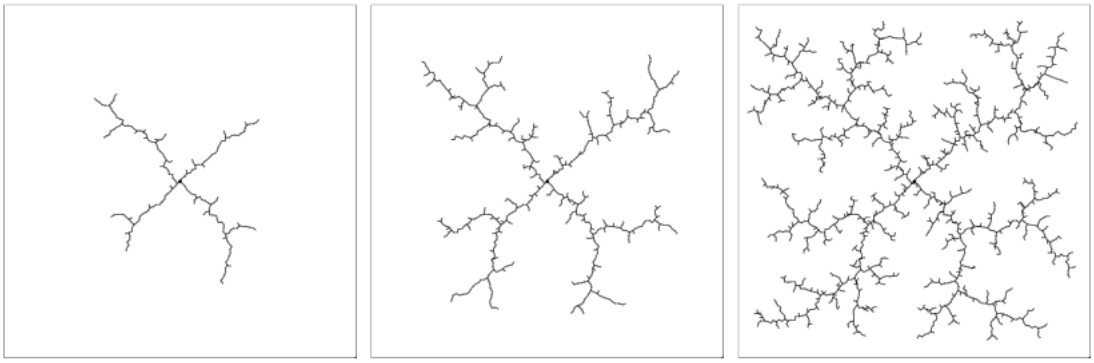


Figure 3.1: This image shows an RRT being expanded into an empty configuration space. The algorithm has a tendency to expand into the least searched spaces [15]

The search is done by randomly sampling the configuration space. Then, the nearest node in the tree to the random point is found. RRT will try to extend a new node from that nearest node towards the random sample by a set step size. If the extension from the nearest node towards the random sample results in a collision-free path, the extended node will be added to the tree [15].

RRT works since the tree will always continually extend itself outward in all directions and usually into the least explored of the configuration space, as shown in Figure 3.1. As a result, the algorithm searches the space very quickly. This is an improvement over random walks where it is possible that the search does not advance outward.

There have been many extensions to the basic algorithm such as goal sampling [19], bidirectional search [14], and the use of kD-trees to find nearest neighbors [23]. These extensions have greatly improved the algorithm's performance. Also, RRT cannot create an optimal cost path since the algorithm randomly searches the space, but there has been an RRT-based algorithm known as RRT* that is asymptotically optimal by using an A*-like cost function [11].

```
buildRRT(start, numberOfSamples, stepSize)
begin
    tree = Tree();
    tree.addNode(start);
    for numberOfSamples do
        xRandom = getRandomState();
        xNear = getNearestNeighbor(xRandom, tree);
        xNew = extendTree(xRandom, xNear, stepSize);
        if xNew is valid and Edge(xNear, xNew) is collision-free then
            tree.addNode(xNew) tree.addEdge(xNear, xNew)
        end
    end
    return tree
end
```

Algorithm 1: buildRRT(start, numberOfSamples, stepSize)

3.1.3 Batch Informed Trees (BIT*)

The BIT* can be considered as the state of the art in sample-based path planning and is the basis for the algorithm used in this thesis [6]. This algorithm uses the concepts of creating batches of samples and incremental search techniques to find and improve path planning solutions.

Additionally, the algorithm uses the concept of restricting the planning problem to an informed set from an extension of the RRT* algorithm known as Informed-RRT* [5]. This extension speeds up RRT* by restricting the algorithm's search space to a reasonable subproblem that is defined by an ellipsoid around the start and goal configurations that is bound by the current best cost to the goal node so far and the distance between the start and goal nodes, as shown in Figure 3.2.

BIT* starts by initializing a search tree, informed space, search radius, and priority queues for vertices and edges. The algorithm starts off with no information of the environment, so the informed space is initialized as the entire environment.

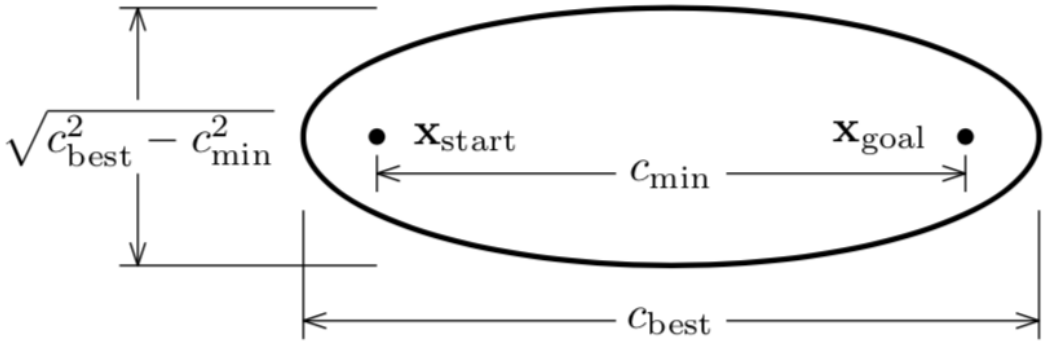


Figure 3.2: Ellipsoid of informed subproblem. The ellipsoid is used to constrain the area used to generate new samples for RRT*. This is done since it can be proven that the ellipsoid will allow a path to be found while also continually bettering the solution path. This constraint helps the algorithm find a path to the goal significantly faster [5].

The informed space is initialized as the entire environment, as it does not have more information on the environment. The search radius is used to determine which nodes would be considered as neighbors. The search radius is dependent on the size of the informed space since as the informed space decreases, the density of sampled nodes increases. So, in order to not process too many nodes, the radius decreases with the decrease in informed space size. The vertex and edge queues can be thought of as data structures used to contain the current state of the known map for the search algorithm to use.

After initialization, the algorithm continues until a termination condition is reached. The general flow for the algorithm follows as such. First, if possible, all nodes that have been previously sampled in the environment are samples that cannot improve the path to the goal are removed. This process is called pruning and it is an expensive process. Pruning will only be done if the informed space has changed for efficiency.

The informed space denotes the area where new nodes can improve the cost to reach the goal. If the cost to the goal is improved, the space will decrease in size. For

efficiency, pruning events will only occur if the informed space becomes smaller, otherwise, there would be no point in doing so.

After pruning, the informed space is sampled a specified number of times. After that, the vertex queue is filled with the nodes of the current search tree. This can be seen as loading the current map that the algorithm has found.

After the vertex queue is loaded, the lowest cost vertices are popped from the queue. If those vertices are within the informed space, the edges from those vertices are added to the edge queue since they are worth exploring.

After the edge queue has been updated, the best edge in the queue is analyzed. First, the edge is checked to see if it can improve the cost to the goal since that cost may have improved since the last edge was checked.

If this was not the case, both the vertex and edge queue would be cleared since there are no more vertices or edges that can improve the path to the goal. Otherwise, the edge is checked to see if it is collision free and that it improves the path to the end point of the edge.

If the edge is determined to be collision free and improves the path to the end point of the edge, the edge is added to the search tree.

The edge adding process continues until there are no more vertices to process or there are no more edges within the informed space to search.

The entire process continues until a defined termination criteria is reached.

```
BIT*(start, goal)
begin
    Initialize tree with only the start in the tree;
    Initialize the edge and vertex queues as empty queues;
    Set the initial connection radius to be infinity;
    while the termination condition has not been reached do
        if the queues are empty then
            Remove all samples that are outside of the informed subproblem;
            Add samples to the space from the informed subproblem;
            Add all vertices in the tree to the vertex queue;
            Adjust the radius based on the current tree size and number of
             samples taken;
        end
        while there are vertices worth exploring do
            Add edges from the best vertex to new samples that can improve
             the current solution to the edge queue;
            Add edges from the best vertex to vertices in the tree that can
             improve their current cost;
            Get the best edge from the edge queue;
            if the best edge is within the informed subproblem then
                if after collision checking the edge, the edge still could improve
                 the solution then
                    if the edge improves the cost to the edge's endpoint then
                        if the endpoint was in the tree then
                            Remove the old connection from the tree
                        end
                        else
                            Add the endpoint to the tree and remove it from the
                             sample set
                        end
                        Add the new edge to the tree and remove all edges that
                         do not improve the path to the edge endpoint from the
                         edge queue;
                    end
                end
            end
            else
                Flush both queues
            end
        end
    end
    Return the tree
end
```

Algorithm 2: BIT*(start, goal)

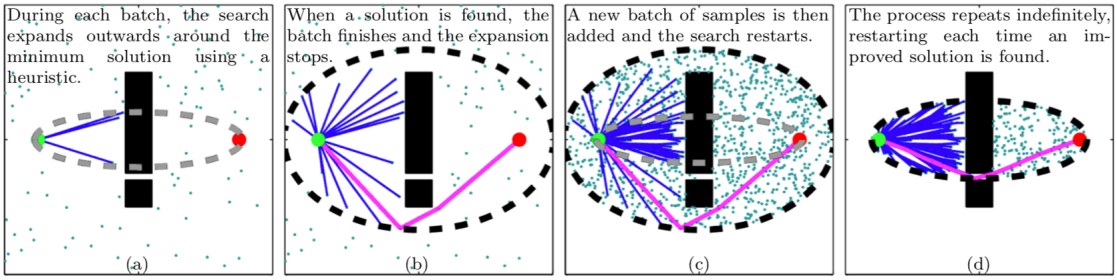


Figure 3.3: This image shows the growth of a batch informed tree. The tree initially searches within a smaller subproblem that is expanded until a path to the goal is found. After that, the subproblem is resampled and the path is improved continually[6].

3.2 Reactive Algorithms

What is termed as reactive algorithms can be defined as planning algorithms that update quickly to the surrounding environment. These algorithms do not use grids or sampling to plan paths. They use gradient and optimization-based approaches instead.

3.2.1 Artificial Potential Fields (APF)

Artificial potential field, or APF, path planning is an older path planning technique that is based on the idea of using potential gradients to avoid obstacles while being drawn towards the goal configuration [1]. In order to do so, the method uses an attractive potential to lead the robot towards the goal. This potential field is superimposed with the repulsive potentials generated by the obstacles as shown in Figure 3.4. The robot then follows the potential gradient of the surface down towards the global minimum in the environment, which is the goal configuration. The potential calculation can be done very quickly since the potentials are based on the robot's distance from the goal and obstacles. But, the robot can easily get trapped in local

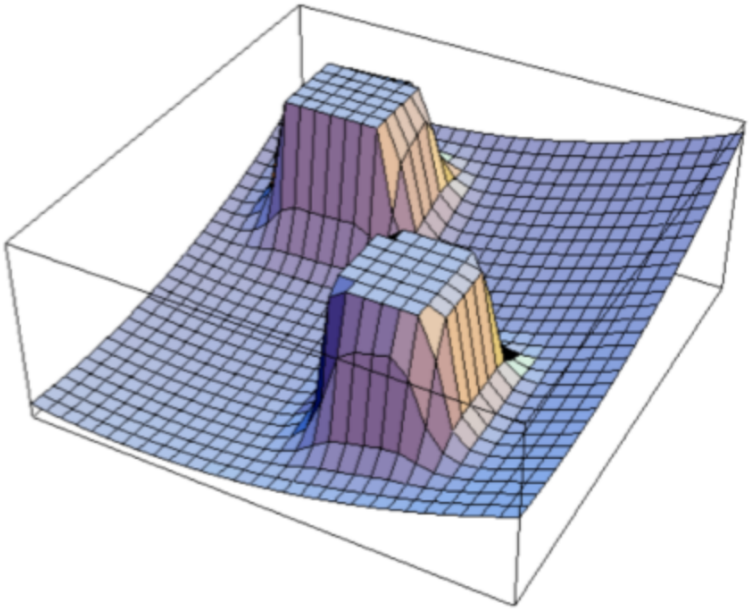


Figure 3.4: This image shows an example of a potential gradient generated for the artificial potential field algorithm. The robot is starting in the rightmost corner and attempting to travel to the leftmost corner [1].

minima. This can be avoided by using sample based algorithms to guide the robot towards the goal and away from local minima.

3.2.2 Reciprocal Collision Avoidance

Another class of reactive algorithm is the reciprocal collision avoidance algorithms. These algorithms are used in multi robot path planning algorithms where robots react in similar ways to avoid each other.

One of the main reciprocal collision avoidance algorithms is ORCA [20]. This technique based on the concept of a velocity obstacle which has been used for collision avoidance with dynamic obstacles for a long time. A velocity obstacle defines a set of velocities that will result in a collision between the robot and the moving obstacle.

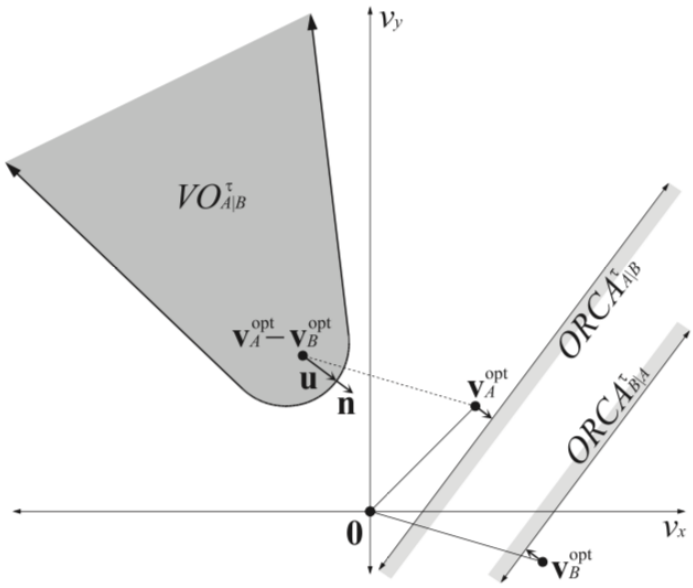


Figure 3.5: This image shows an example of a velocity obstacle and the ORCA planes generated from the velocity obstacle This half plane represents the set of velocities the robot can take to avoid a collision with the other robot. Half of the minimum velocity needed to ensure a collision free velocity is used since it is assumed the other robot will do the same. As a result, the robots share half of the responsibility to avoid each other, hence the reciprocal nature of the algorithm[20].

ORCA uses velocity obstacles induced by the robots onto other robots to help them avoid colliding with each other.

The ORCA algorithm starts by detecting the current position and velocities of all other robots in the free space. Then, the velocity obstacle induced by another robot is calculated in the velocity space based on the relative position and velocity of the ego robot to the other robot. From this velocity obstacle, a half plane is found by calculating the vector sum of the ego robots' desired velocity and one-half the minimum velocity needed to get out of the velocity obstacle as shown in Figure 3.5.

Once the safe velocity half planes induced from all other robots are found, a linear programming algorithm is used to determine the safe velocity for the ego robot that is closest to the robots' desired velocity as shown in Figure 3.6.

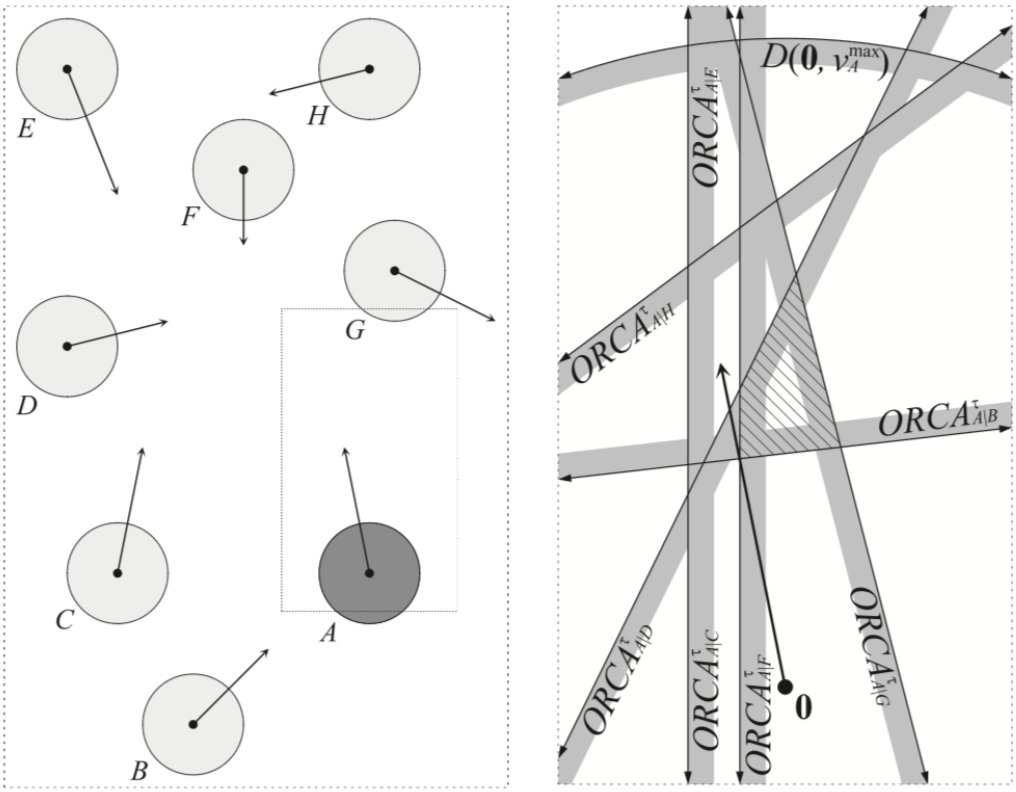


Figure 3.6: This image shows how the optimal safe velocity is determined for a robot. The optimal safe velocity is determined by using the ORCA plane intersections [20]. From this formulation, all the robots are guaranteed to have collision-free velocities. As a result, all robots should be able to follow collision-free trajectories to their goal configurations without the need for explicit communication. But, as this is a purely reactive algorithm, robots can get into deadlock situations in dense environments.

Chapter 4

APPROACH

In this chapter, the approach taken for this thesis will be described. The chapter begins with describing the overall workflow for the simulations used to test the algorithms used in this thesis. This is followed by the workflow used by the individual robots in the simulation. An in depth look is given for the various algorithms tested in the thesis.

4.1 Simulation Workflow

The simulation environment is implemented in Python using the PyQt5 library using Python version 2.7. The QGraphicsView and QGraphicsScene libraries and associated modules are used to visualize and provide functions such as collision detection.

The simulation starts by loading in an environment configuration which is defined in an XML file. This file specifies the locations of the static obstacles for a test environment. The test environment file also defines the location of the goals for the environment. These locations were selected since they would serve as interesting locations for the robots to travel to.

After setting up the test environment, the robots are placed at their starting positions. The starting locations are selected for the set of goal locations in the environment. The particular starting locations are randomly selected from the set of goal locations. This was done in order to increase the chances of robots passing each other which would stress the various algorithms that were tested.

The simulation runs at 20 frames per second in order to simulate real time performance. On each time step, each robot in the simulation will determine a velocity to use in order to traverse the environment in order to reach its assigned goal. These velocities are determined based on a snapshot of the test environment on the previous time step. Each robot moves in a prioritized order with the robots moving in order according to when they were initialized. In this way, the trajectory planning for the robots is not prioritized, but the movement for the robots is.

PyQt5's signalling system is used in order to process events of interest such as when a robot reaches its goal and when a robot collides with an element in the environment. A test in the simulation will end if all robots reach their goals, a robot collides with an obstacle, or if a time limit is reached.

The signals used are monitored by a robot watchdog module. The module tracks the state of all robots. The states of interest are if the robot has collided or if the robot has reached its goal. If the robot has done neither of those actions, it is assumed that it still navigating the environment.

When a robot reaches a spot within the goal radius of the goal, the watchdog sends a signal to simulator which records that that robot has done so and that robot will no longer move until the completion of the test. When the simulation has marked down that all robots have reached their goals, the simulation terminates and records relevant test results.

Collision checking is handled by the collidingItems() function in the QGraphicsScene module. This function returns a list of all simulation elements in the current simulation scene that are overlapping with the robot. If any of those elements are a static obstacle or another robot, the robot raises a flag indicating that it has collided with an obstacle. The watchdog sees that flag and signals the simulation that a collision

Parameter	Value
Robot Size	25 x 25
Robot Speed	15 units per frame, 300 units per second
Environment Size	1000 x 1000
Simulation Maximum Runtime	30 s
Goal Radius	15 units

Table 4.1: This table lists some of the key parameters for the simulation setup. These include robot dimensions and abilities and environmental constraints.

has occurred. The simulation will record all relevant test results and then terminate the test.

A timer is set at the beginning of the simulation for 30 seconds. Once that timer has expired, the simulation records all relevant results and terminates the test. This timer exists to prevent tests from running indefinitely. Tests can run indefinitely if the path planning module for an agent fails to find a path or if multiple robots are deadlocked and none of those robots can progress towards their goals. The most similar paper to this thesis used time limits of one and five seconds [9]. The reason why the time limit for this thesis is significantly higher is because the replanning and APF extensions take considerably longer than the ORCA extension to complete.

The simulations were run on Google Cloud Compute Engine using a virtual machine with N1 general purpose compute engine. The N1 engine uses an Intel Haswell CPU that runs at 2.3 GHz. The computer instance was configured to have 8 vCPUs or equivalently 8 CPU cores.

All relevant parameters for the simulation are listed in Table 4.1.

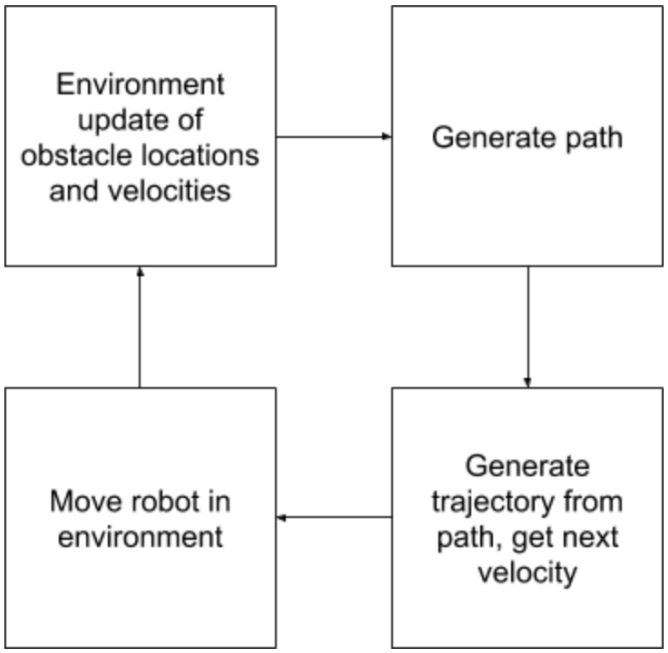


Figure 4.1: This figure represents the general workflow for robots running in the simulation.

4.2 Robot Workflow

In this section, the workflow for generating a velocity for a robot is described. As previously stated, the algorithms used for this thesis is a decentralized algorithm, therefore, the discussion of the approach will be limited to a single robot. The general workflow for the agent is described in Figure 4.1.

The robot retrieves all relevant data from its environment. From that data, it will generate an obstacle-free path to its goal and use a trajectory based on that path to reach that goal.

It is assumed that the robot has perfect knowledge of the environment and is able to know the location and size of all objects in the environment. To the robot, everything that is not itself is an obstacle, including other robots. The velocity of the other robots is interpolated from the difference in position of the other robots from

time step to time step. This knowledge of the obstacles is fed to the path planning module and the trajectory generation module, if necessary, of the robot. This is an idealized simulation and an actual real world implementation would likely need robust localization and perception systems to match this simulation.

The path planning module of the robot runs the BIT* algorithm as described in the Background Information section. The algorithm will generate a collision-free path and the waypoints from that path will be fed into the trajectory generator in order to move the robot towards its goal.

4.2.1 Implementations of Robot Workflow

This section describes the three approaches used in this thesis. These methods could be combined to test more variations, but they are only run independently to test the merits of each method.

Before moving forward with the discussion of the approaches, it must be stated that the most similar approach taken to the problem presented by this thesis is the ORCA-RRT* algorithm. The main difference between the approaches taken in this thesis and ORCA-RRT* are that this thesis uses a decentralized approach. ORCA-RRT* creates an RRT in higher dimension space with each robot providing three dimensions to the problem. After solving for paths for all the robots, it uses ORCA to generate collision-free velocities for each of the robots [9]. The approaches used in this thesis use BIT* instead of RRT* and uses a decentralized method for all the robots to plan their paths. Each robot plans their own path individually, not in one large tree with the other robots.

4.2.1.1 Path Planning

As stated previously, the path planning module of the robot uses the BIT* algorithm. In particular, for this implementation, less samples are used than described in the original paper. This is due to the fact that this implementation of the algorithm is written in Python and the need to plan for multiple robots in a relatively short amount of time. This could be improved by more accurately simulating the planning for the robots by using multi threading and have individual threads plan for individual robots.

Additionally, in the implementation of checking path plan collisions, a buffer zone is added to checking for valid samples and valid edges. In the most basic implementation for extending a connection from nodes, the connection is considered valid if a straight line from the starting node to the desired node does not intersect with any obstacles. For this implementation, a rectangle whose length is the distance between the nodes with a width equal to twice the width of the robot is used to check for valid edges. If an obstacle overlaps with the rectangle that is extended between the nodes under test, the connection is considered invalid. This technique ensures a safe area that the robot can maneuver when following the path. This extra space is especially helpful when the reactive algorithms are used since the robot will have more space to avoid other robots.

Additionally, the termination criteria for the algorithm is when the algorithm finds a path that reaches the goal for the first time. This misses out on one of the main benefits of the algorithm in that it can improve the path found for the robot with searching the informed space more, but due to implementation issues, this is found to be good enough.

Parameter	Value
Number of Samples	10
Termination Condition 1	5 seconds
Termination Condition 2	Goal reached 3 times

Table 4.2: These are the parameters used for BIT*. These parameters were selected primarily based on generating a path quickly.

Finding the goal three times or within five seconds was selected since it was difficult to set a consistent time limit in order for a path to be found for all start and goal combinations. All parameter values used for this BIT* implementation are shown in Table 4.2

4.2.1.2 Trajectory Generation

The robot's trajectory is generated by moving the robot in a straight line towards the next waypoint in the path. The magnitude of this movement is defined by the maximum speed of the robot. The velocity calculation starts with finding the vector from the current position of the robot to the next waypoint from the robot's planned path. Then, the vector is normalized and scaled by the robot's maximum speed, which is noted in Table 4.1. This resulting vector is the velocity or commanded input for the robot on the next time step. The magnitude of this vector is a floating point value that is less than the robot's maximum speed. The robot is capable of changing velocity instantaneously based on these calculations.

The desired waypoint is updated when the robot comes into a close enough proximity with the desired waypoint. The desired waypoint is then changed to the next waypoint in the path.

Parameter	Value
Number of Samples	1
Termination Condition 1	5 seconds
Termination Condition 2	Goal reached 1 times

Table 4.3: These are the parameters used for replanning the path of the robot. These conditions are constrained that that for the initial plan in order to allow for the robot to plan and move quickly out of the path of the other robots or obstacles.

4.2.1.3 Replanning

This implementation is relatively straightforward during the test the robot will adjust its path when necessary. This is done by rerunning the BIT* algorithm with less samples in order to find a new path quickly. This method works out well since BIT* is based on the idea of implicit connections in a random graph and the use of incremental search. As a result, the search tree can be rebuilt quickly and a new path can be found.

A replan is triggered when an obstacle is within the sum of the maximum velocities of the robot and the obstacle and if that obstacle breaks the path the robot is following. Replanning in general is an expensive operation and this method limits the number of replans that the robot will perform. Another issue with replanning is that it can result in deadlocks due to the random nature of the BIT* algorithm which also does not account for the direction that the blocking obstacle is moving in. Table 4.3 shows the parameters used for the replanning the path with BIT*.

4.2.1.4 Optimal Reciprocal Collision Avoidance (ORCA)

The second algorithm that is tried by this thesis is to generate safe velocities using the Optimal Reciprocal Collision Avoidance algorithm. For this method, the path planner runs BIT* only after the simulation is initialized. The robot will follow this

Parameter	Value
Time Horizon for Agents	2.125 frames
Time Horizon for Static Obstacles	2 frames
Search Radius	63.75 units

Table 4.4: These are the parameters chosen for ORCA. They were selected in order for the robots to avoid each other with a minimal degree of separation.

initial path, but the velocities generated from the method described in the Trajectory Planning section are passed in as the robot's desired velocity to the ORCA algorithm. The algorithm should produce a safe, collision-avoiding velocity for the robot. The resulting velocities generated by ORCA will be a floating point number that is less than the robot's maximum speed. The parameters for the algorithm are listed in Table 4.4.

4.2.1.5 Path-guided Artificial Potential Fields (APF)

The final approach that is taken for this thesis is based on the path-guided artificial potential field algorithms discussed in the Literature Review section. Like in that algorithm, a path is generated when the simulation is initialized with BIT*. Then, the robots are to navigate the environment using artificial potential field methods. The method uses commonly used potential functions to handle the cases with static obstacles and has an additional potential function for handling dynamic obstacles.

The attractive potential function is used to guide the robot towards the next waypoint in the global plan provided by BIT*. The force has the robot follow the generated path closely as shown in Equation 4.1.

The repulsive force put onto the robot is the sum of the repulsive forces imparted by all obstacles in the environment. This is repulsion is inversely related to the distance

of the robot to a particular obstacle as in commonly used repulsive potential functions as shown in Equation 4.2.

Finally, there is an adjustment potential that is used to avoid the other robots in the system. This potential is only used when robots are close enough to each other and if those robots are approaching each other head on. The magnitude of this force is equal to the sum of all the obstacle potentials. The direction of the force is to the right of the direction of travel for the robot as shown in Equation 4.3.

For all equations, $\vec{w}$ is the vector that starts at the robot's current position and goes to the next waypoint. $\vec{o}$ is the vector that extends from the robot's current position to an obstacle. D_o is the obstacle range. W_{att}, W_{repl}, W_{corr} are the weights for the attractive, repulsive, and adjustment forces respectively.

The potential functions and their scales are shown in the Table 4.5.

$$F_{att} = W_{att}\frac{\vec{w}}{|\vec{w}|} \tag{4.1}$$

$$F_{repl} = W_{repl}(\frac{1}{|\vec{o}|} - \frac{1}{D_o})(\frac{1}{|\vec{o}|})(\frac{\vec{o}}{|\vec{o}|}) \tag{4.2}$$

$$F_{coord_x} = 0; F_{coord_y} = W_{corr}\sum_i \frac{1}{|\vec{o}|} - \frac{1}{D_o} \tag{4.3}$$

Parameters	Values
W_{att}	0.3
D_o	500
W_{repl}	30,000
W_{corr}	32.0

Table 4.5: This table has the weights for the APF algorithm. They were decided upon since they lead to the best chance of robots avoiding each other.

Chapter 5

TESTING

In order to test the various methods used in this thesis, multiple environments were used. For each environment, the number of robots run during the test was varied from two robots to eight. Each environment was designed to stress each of the algorithms in order to determine which would be the best approach in terms of how successful and how efficient they were.

5.1 Environments

Six environments were used for testing the various approaches. The environments were designed to stress both the path planning portion and the reactive portions of the approaches. For each of the figures below, the obstacles in the environment are the black rectangles and the start and goals for the environment are red circles. All obstacles in the environments are static. The start and goal circles represent the points where the robots are either start or finish. Each of these points is hand selected. For each trial, a robot is randomly assigned one of these points to start from and another point to end at. The first environment used is an obstacle free environment, shown in Figure 5.1. For planning, the solution is trivial and the environment mostly tests the performance of the reactive portions of the approach. With the freedom to move in any direction, the reactive algorithms should be highly successful in navigating the environment. The goals are designed in a way such that encourages the robots to come into proximity with each other. This is the default environment used to test multirobot path planning problems since if the algorithms used fail while not in

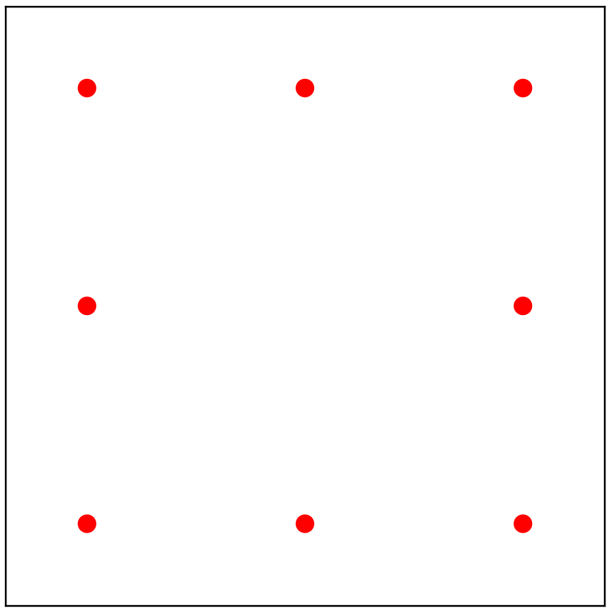


Figure 5.1: Empty Environment has no obstacles. The red dots indicate the start and goal points.

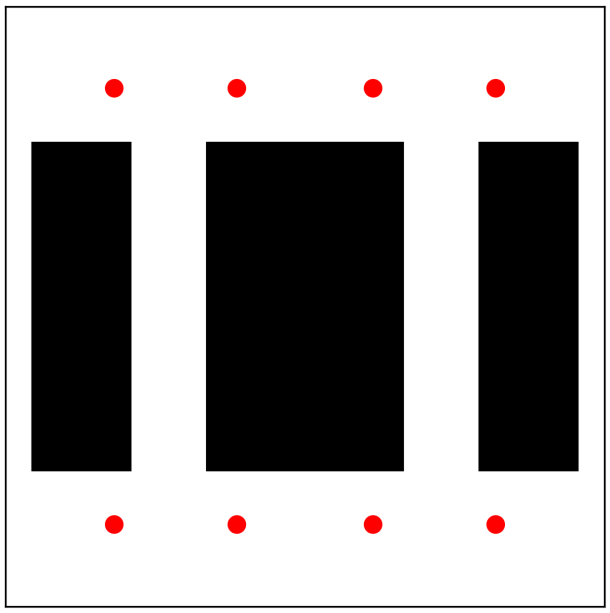


Figure 5.2: Two Corridors Environment has three obstacles and two narrow corridors through to the other side of the environment.

the presence of obstacles, there is very low likelihood that they will succeed in the presence of obstacles. This kind of environment was seen with [9], [20], and [2].

The next environment is the two corridors environment which has two narrow corridors for the agents to pass through, shown in Figure 5.2. This makes it a little difficult for the path planning to find a plan quickly. Additionally, the confined space in the corridors limits the area the robots have to deviate from those paths in order to avoid each other in the corridors, making it difficult for the reactive algorithms. The width of the corridors in this environment is 140 units. The goals were created in order to have the robots traverse the corridors as much as possible. This is a commonly used environment to test similar algorithms as seen in [3].

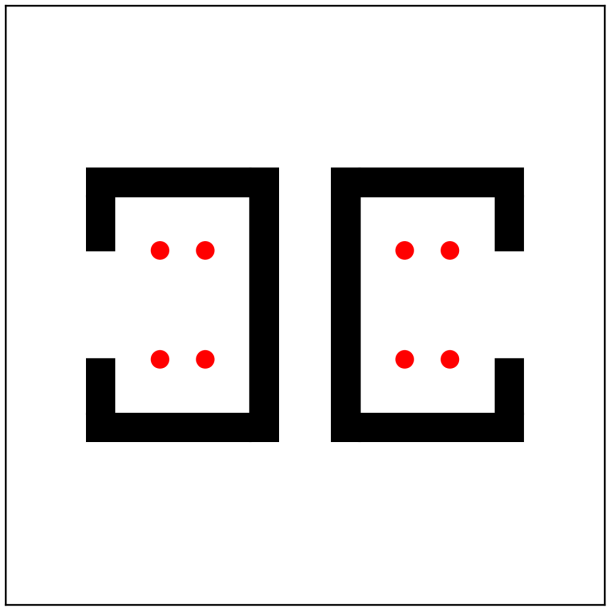


Figure 5.3: Double Bug Trap Environment has robots in two room-like structures. The robots will have to travel from one room to the other room.

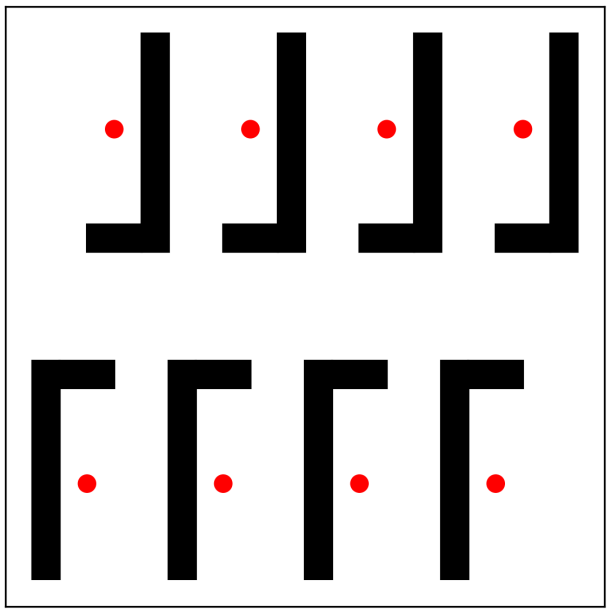


Figure 5.4: Office Environment features obstacles placed in an office-like setting.

The next environment is the double bug trap environment, shown in Figure 5.3. This environment provides a significantly more difficult challenge for planning since the path must leave one trap and into the other trap. The traps also cause issues for the reactive algorithms since there is less space for robots to deviate from their planned paths. This is commonly used environment to test path planning algorithms and their ability to find paths when the greediest path is unavailable as seen in [3].

The office environment has the robots navigating from one side of the environment to the other in a way that will force them to travel through a single corridor, shown in Figure 5.4. The idea behind this environment is that it is a slightly less contrived as the other environments.

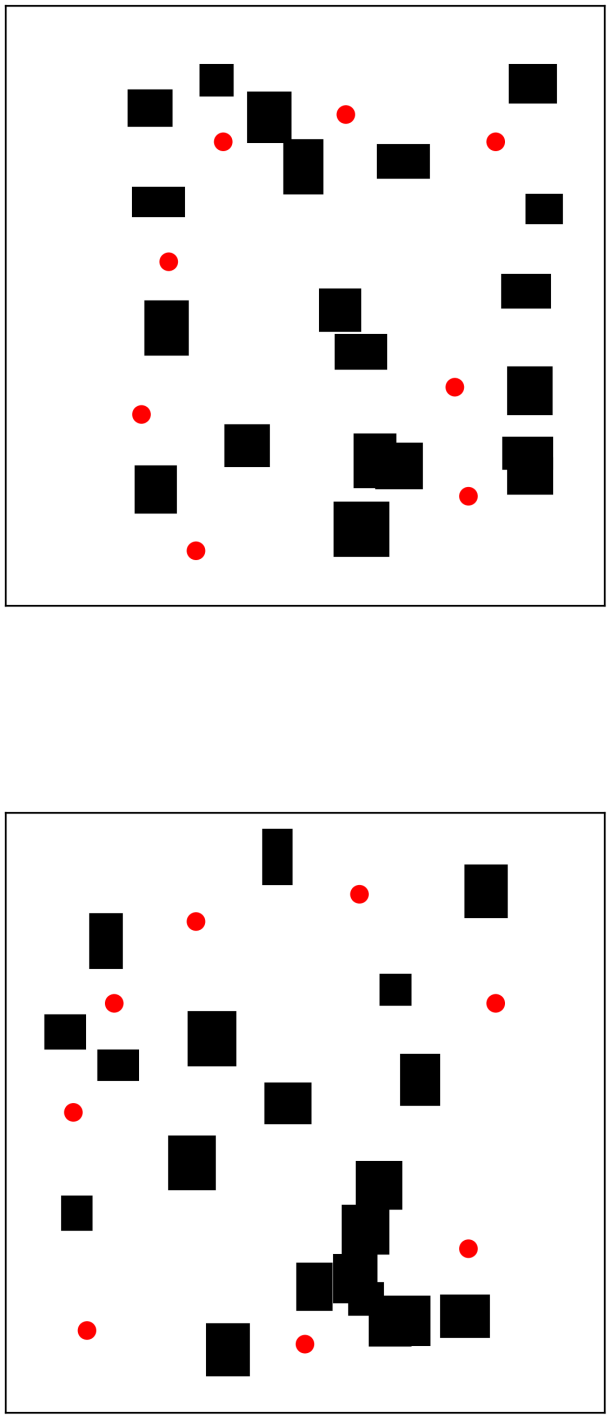


Figure 5.5: Random Environments are environments with randomly generated obstacles.

Finally, there are the two randomized environments, shown in Figure 5.5. These environments were created in order to test how the approaches would perform in general. These environments were generated by randomly assigning the positions and sizes of the obstacles in the environment. The positions of the obstacles were placed by randomizing the location of the top-left corner of the obstacles. The size of the obstacles range between 50 and 100 units in both width and height. These environments are similar to the ones used to test BIT* [6].

Chapter 6

RESULTS

In this section, the results of the trials run for testing the algorithms presented in this paper will be analyzed. For each algorithm, the main competencies that will be analyzed are efficiency and completeness. Efficiency refers to how quickly and/or efficiently an algorithm is able to perform. Completeness refers to the ability of the algorithm to have robots reach their goals. Additionally, each algorithm is compared to planning a single robot with BIT* in both measures. The replanning algorithm is observed in more depth to gain a better understanding of the operation of that approach.

6.1 Metrics

In total, ten metrics were used in evaluating the various approaches. These metrics can be subdivided into more general categories as they are related to each other. The categories follow as such: efficiency metrics, success metrics, failure metrics, and computational metrics.

6.1.1 Efficiency Metrics

Efficiency metrics aim to measure how efficiently and how quickly the approaches were able to move the robots from their starting configuration to their final configuration. The two measures that will be employed are average distance traveled and average runtime for individual robots. Average distance traveled tries to capture how far an

individual robot traveled in its set of trials. Set of trials refers to the combination of test environment, approach, and number of robots. Average runtime captures how long it took for a robot to reach its goal for a particular set of trials.

6.1.2 Success Metrics

Success metrics show how capable the approaches were in having a robot travel from its starting configuration to its goal configuration. This is captured in the success rate metric which is the ratio of the number of robots that were successful able to reach their goals compared to all robots run in a set of trials.

6.1.3 Failure Metrics

Failure metrics show how a set of trials failed to complete. The two measures used here are the collision rate and the timeout rate for a set of trials. Collision rate is the ratio of trials that ended with a robot colliding with an obstacle or other robot to total trials. The timeout rate is the ratio of trials that reached the 30 second time limit to the number of total trials. Timeout rate is a measure that indicates how many times a set of robots in a trial would arrive in a deadlock situation where neither robot(s) could progress toward their respective goals.

6.1.4 Computational Metrics

Computational metrics aim to determine the amount of computational resources needed to run the approaches. The metrics used to indicate this measure are the nodes added to the search tree, the nodes explored by the algorithms, and the nodes sampled by the algorithm. These node-based metrics capture the memory cost for

Metric	Description
Success Rate	Number of robots that were able to successfully reach their goals vs total number of robots
Average Distance	Average distance traveled by an individual robot
Runtime	Average amount of time taken to by an individual robot to move in the environment
Collision Rate	Ratio of number of trials that ended in a collision to the total number of trials
Timeout Rate	Ratio of number of trials that ended in a timeout to the total number of trials
Nodes Added	Average number of nodes added to search tree by a robot
Nodes Explored	Average number of nodes explored by a robot
Nodes Sampled	Average number of nodes sampled by a robot

Table 6.1: Summary of metrics used to evaluate the multi-robot approaches

approaches. For the CPU resource measures, the runtime metric is used here as well.

6.2 APF-based Approach

6.2.1 Completeness

In general, the success rate for the APF approach decreased with an increase in the number of robots, as shown in Figure 6.1. The approach resulted in close to a 90% success rate in the empty and double bug trap environments for variations in number of robots. The approach performed worse in the office and first random environment with the success rate dropping to between 70 and 80 percent when more than six robots were in the environment. The APF approach struggled greatly in the two corridors and second random environment with success rates falling from 85 percent with two robots to 50 percent with eight robots.

6.2.2 Efficiency

In general, the average distances traveled by the robots increases slightly, as shown in Figure 6.2. But, for the two corridors environment, the distance traveled is significantly higher. For trials with more than four robots, the average distance traveled

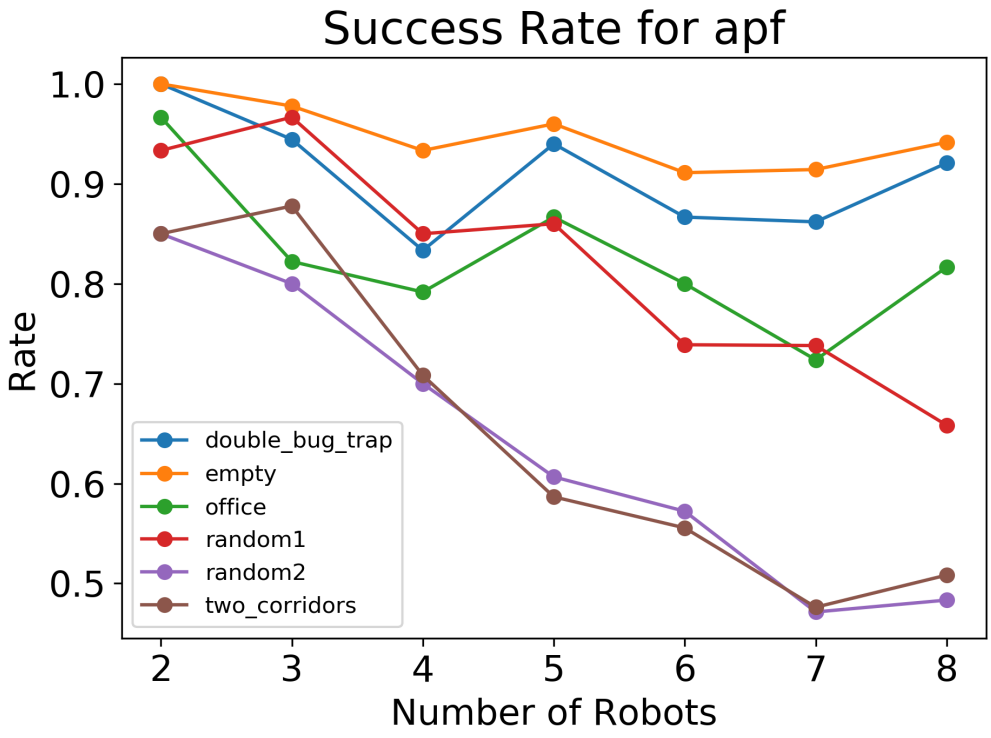


Figure 6.1: Success Rate for APF approach is shown. The double bug trap and empty environments had the highest success rate, while the random2 and two corridor environments had the lowest.

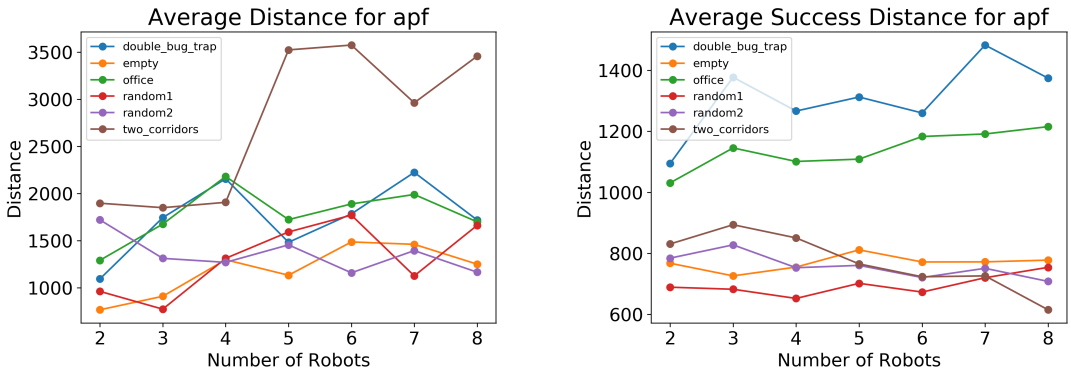


Figure 6.2: Average distance traveled by robots using the APF-based approach is shown on the left. The average distanced traveled by successful robots using the APF-based approach is shown on the right. The average distance traveled in the two corridors environment jumps from 2000 units to 3500 units once the number of robots in the environment increases beyond four robots. When controlling for only successful robots, the average distance traveled remains mostly the same for all variations in number of robots.

by robots in the two corridors environment jumps from around 2000 units to between 3000 and 3500 units.

When filtering out the robots that were unsuccessful in reach their goals, the average distance traveled for all environments remain relatively stable for all variations in number of robots. The average distance traveled for successful robots in the double bug trap and office environments is higher than for other environments, but that is due to the design of their environment. The average distance traveled by successful robots in the two corridor environment drops to around 800 units which is similar to the empty and random environments.

6.2.3 Cost

Figure 6.3 shows a plot for the average runtime for all robots in all trials and a plot for only successful robots in those trials. Similar to the plot for the average

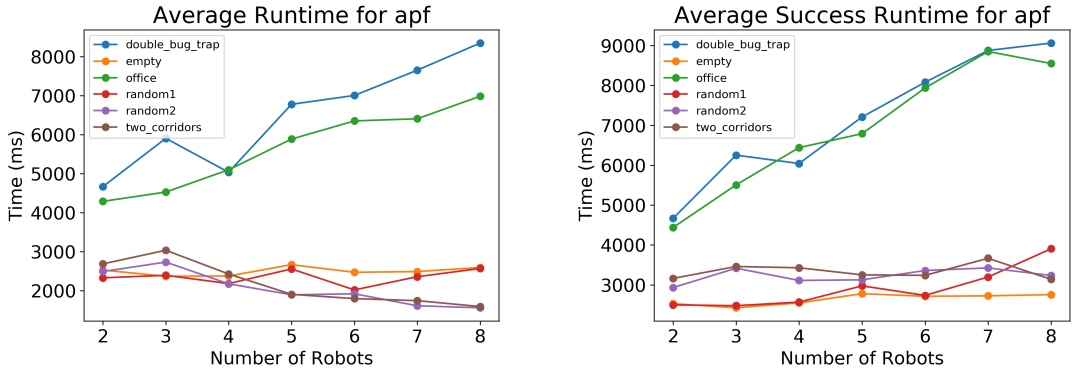


Figure 6.3: Average runtime for robots using the APF-based approach is shown on the left. The average runtime for successful robots is shown on the right. These plots show that the robots took considerably longer to get to their goals in the double bug trap and office environments.

successful distance traveled, the double bug trap and office environments had higher average runtimes than all other environments. Unlike the other environments, the APF-based approach's runtime increased with an increase in number of robots in these environments. Another interesting result is that the average runtime for robots was higher for successful robots than it was for all robots. This is due to an error in the way runtimes were recorded. The runtime for a robot would only be recorded if that robot successfully reached its goal. As a result, if a robot failed to reach its goal, it would appear that its runtime was zero milliseconds instead of being the maximum value of 30000 milliseconds.

6.2.4 Mechanisms of Failure

Figure 6.4 shows the collision rate and timeout rates for the APF-based trials. For the empty, double bug trap, and office environments, the collision rate remained below 10% for all variations in number of robots. For the two corridors and first random environment, the collision rate increased from about 3% to around 20% with the increase in number of robots. The second random environment had that most

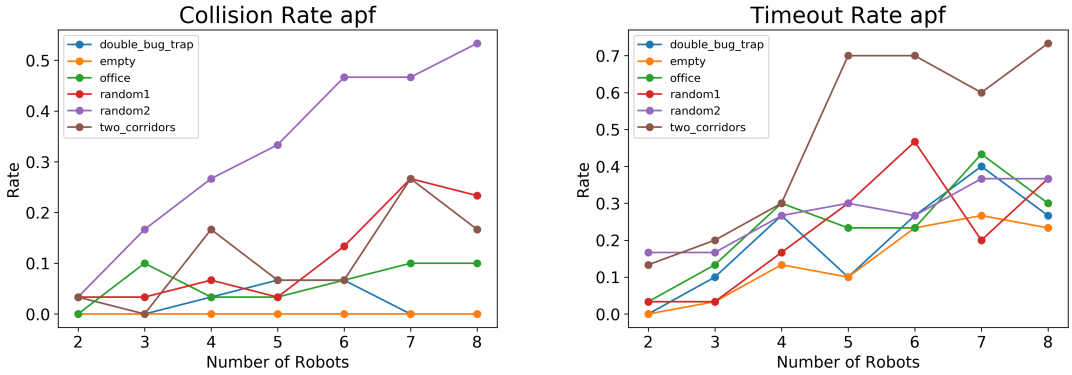


Figure 6.4: The collision rate for trials with robots using the APF-based approach is shown on the left. The timeout rate is shown on the right. Collisions were only common in the random2 environment. Timeouts were very common in the two corridors environment.

collisions out of all environments with the collision rate gradually increasing from around 3% to over 50% as the number of robots increased from two to eight robots.

The timeout rate for all environments except for the two corridors environment increased from less than 1% to around 30% to 40% as the number of robots increased. For the second random environment, the timeout rate increased from 20% to 40%. The timeout rate for the two corridors environment is similar in shape to the average distance traveled by all robots with a jump in timeout rate from 30% to 70% in trials with more than four robots.

6.2.5 Summary of Results for APF-based Approach

The APF-based approach was able to achieve decent results in having the robots reach their goals, but it was far from perfect and the approach has issues with certain environments. To gain insight into why the approach struggled with these environments, the success rate plots from Figure 6.1 serve as a good starting point.

The two environments that the APF-based approach struggled with the most were the two corridor and the second random environments. As the number of robots in the environment increased, the success rate decreased at a steady rate from 85% to below 50% success. The reasons why the approach failed in these environments are distinct from each other and show some of the shortcomings of the approach.

The two corridor environment, as shown in Figure 5.2, consists of two narrow pathways though a large static obstacle in the middle of the environment. Figures 6.2 and 6.4 help illustrate why the APF-based approach failed.

Figure 6.2 shows that the average distance traveled in the environment was 1500 units longer than the other environments. This is most likely due to multiple robots trying to pass by each other in one of the corridors as shown in Figure 6.6. Due to the combination of the coordination force from the other robots and the repulsive force from the static obstacles, the robots exhibit a jittering behavior where the robots in the corridor shake rapidly left and right in an effort to pass the other robots. The corridor is small enough such that the side of the corridor push the robots to their left such that they are unable to pass the other opposing robot. As a result, the average distance traveled appears to be much higher since while the robots cannot progress, they are still moving with a sizable velocity. This claim is supported by the timeout rate plot in Figure 6.4 which shows that approximately 70% of all trials in the two corridor environment with more than four robots end in a timeout due to the robots reaching a deadlock.

For the second random environment, the primary mechanism of failure is a collision, as shown in the collision rate plot in Figure 6.4. The second random environment, as shown in Figure 5.5, has many areas where the static obstacles form a corner or contain a tight space that is just large enough to allow a path through. As a result, robots traversing the environment can reach a local minimum in the environment or

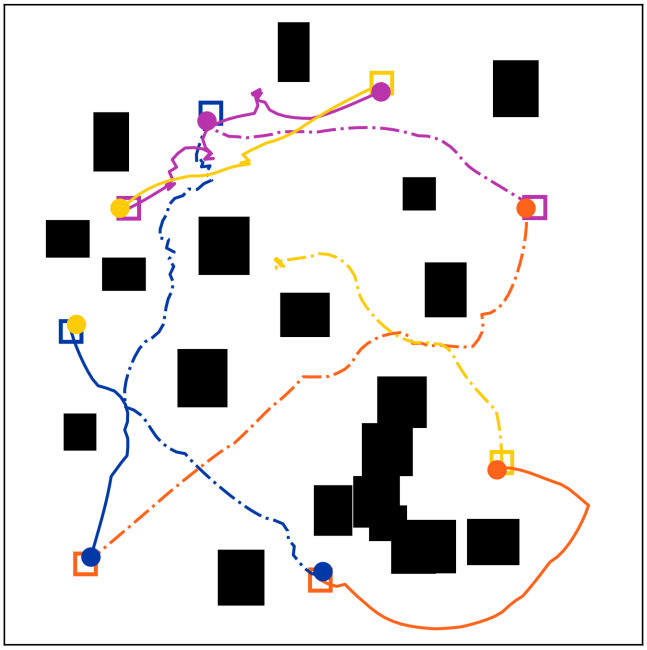


Figure 6.5: The yellow dotted robot has been trapped in a local minimum. This is likely due to the path planner planning a route through a gap that is too small.

collide with a static obstacle. A robot could get stuck in a local minimum if the coordination force on that robot forces the robot to move into a corner, as shown in Figure 6.5. A robot could collide with an obstacle if its planned path attempts to go through a tight space since the opposing repulsive forces on the robot could cause the robot to deviate from its path into that obstacles.

The APF-based approach does help the robots reach their goals, but there are issues with the approach. These issues are most apparent in the two corridor and second random environment. The two corridor environment shows that the approach can result in multiple robots reaching a deadlock since the robots are unable to balance the coordination force and repulsive forces to allow the robots progress toward their goals. The second random environment results in many robots colliding with static obstacles since there are many areas with local minimums.

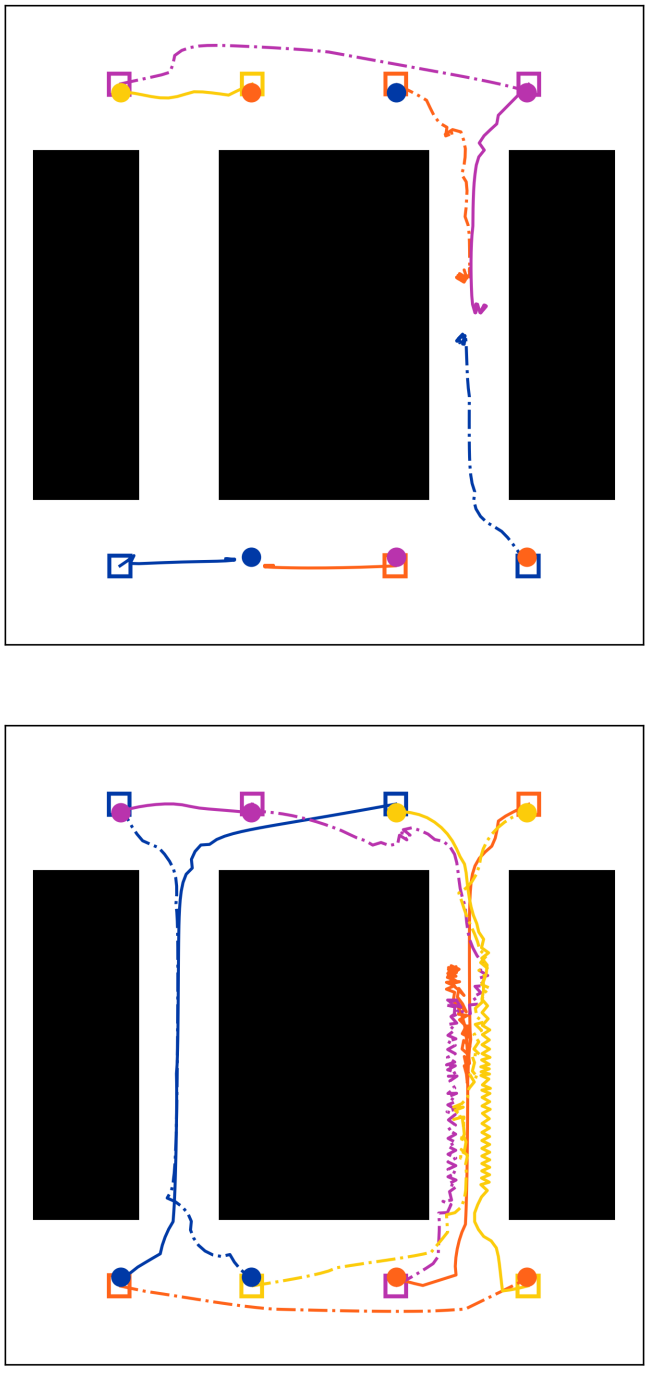


Figure 6.6: The robots on the right are unable to pass each other. Through rare, it is possible for the robots to make it through the pass.

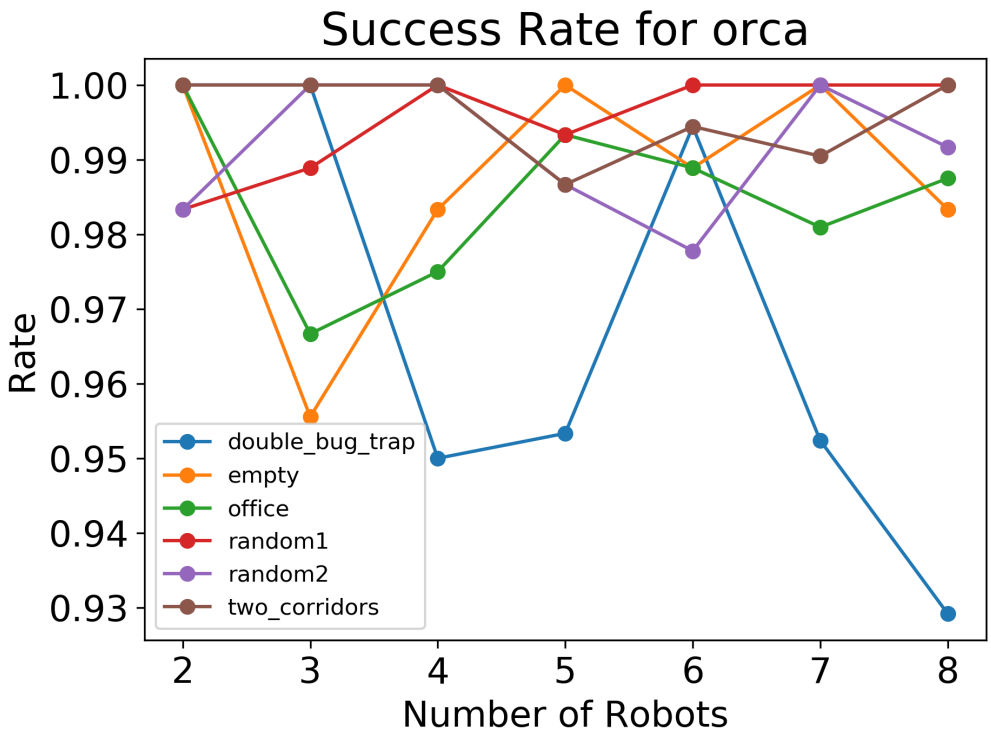


Figure 6.7: Success rate for ORCA-based approach was at least 93% for all experiments.

6.3 ORCA-based Approach

6.3.1 Completeness

The ORCA-based approach had a high success rate, at least 93 percent, for all environments and all variations in number of robots in those environments, as shown in Figure 6.7. This indicates that the approach was able to handle the variance in number of robots and all environments well.

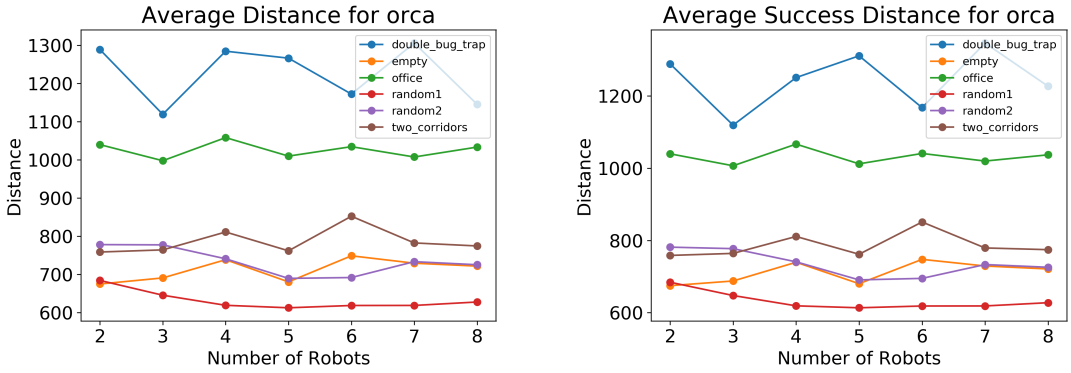


Figure 6.8: The plots for average distance traveled for robots using ORCA-based approach and successful average distance traveled look virtually identical. The distance traveled did not increase with an increase in robots for all environments.

6.3.2 Efficiency

Since the success rate of the ORCA approach was so high, the distances for both the overall average distance traveled and the successful average distance traveled were virtually the same, as shown in Figure 6.8. As with the success rate, the average distance traveled was not affected by the increase in number of robots at all. This indicates that the ORCA approach does not vary the distance traveled for the individual robots in the trial even though there are robot-robot interactions.

6.3.3 Cost

Figure 6.9 shows the average runtimes for the robots which generally increase for an increase in number of robots. For the double bug trap and the office environments, they increase faster, but this is due to the structure of the environment which requires the robots to travel further on average.

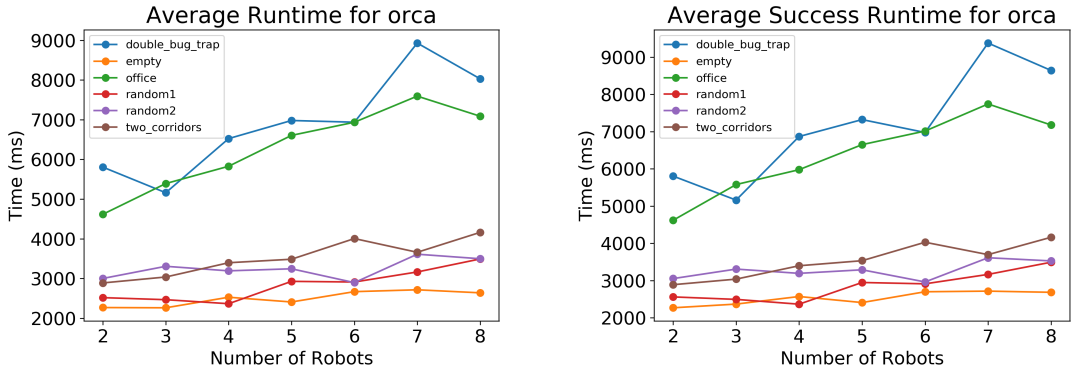


Figure 6.9: Average runtime for robots using ORCA-based approach shows similar behavior to the average runtime for robots using the APF-based approach with the double but trap and office environments having longer average runtimes than the other environments.

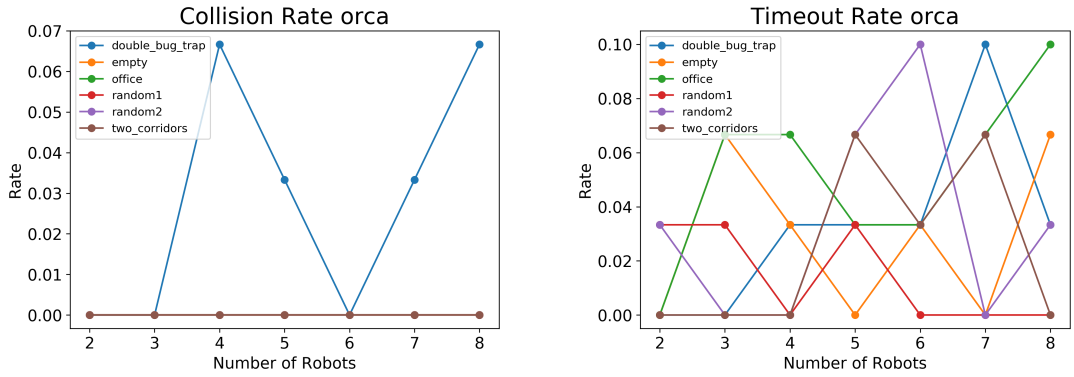


Figure 6.10: The ORCA-based approach rarely resulted in either a collision or timeout

6.3.4 Mechanisms of Failure

The ORCA-based approach rarely resulted in a failure, but when failures did occur, it was most likely due to a deadlock, as shown in Figure 6.10. This would make sense since ORCA intends on generating collision-free velocities. But, due to the reactive nature of the ORCA algorithm, robots managed to get into deadlocked states and local minimums as shown in Figures 6.11 and 6.12.

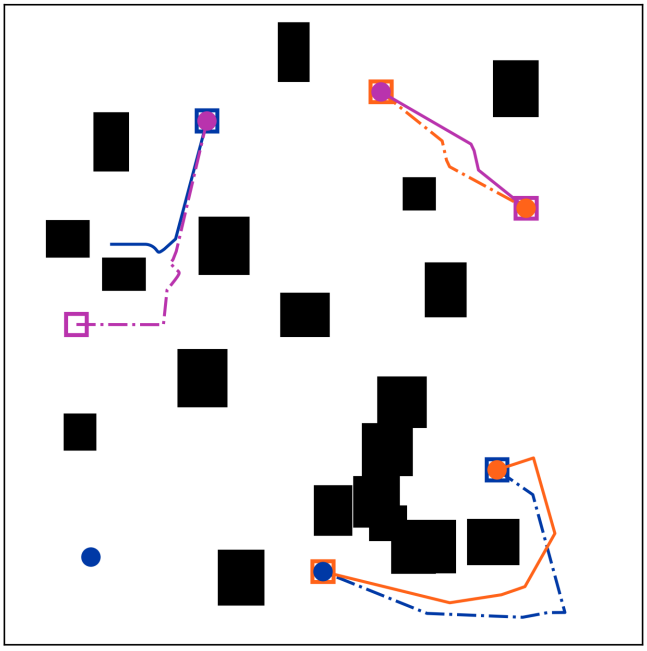


Figure 6.11: The solid blue robot fails to reach its goal due to being trapped in a local minimum

The scenario shown in Figure 6.11 is one where a robot is trapped in a local minimum. This kind of failure would result from one robot moving into position where there is an obstacle between the waypoint and the robot. This event typically would occur when one robot is trying to pass another in a narrow pass through static obstacles. The robot would avoid the other by selecting a velocity that happens to bring the robot towards a position behind an obstacle. After the other robot has passed, the robot will attempt to select a velocity that will take it directly towards the next waypoint in the plan. But, the only safe velocity that can be selected by the ORCA algorithm is close to zero velocity since the robot is surrounded by static obstacles, so the robot sits in that position until the end of the simulation.

Another situation that could result in a robot failing to reach its goal is reaching a deadlocked state, as shown in Figure 6.12. This deadlock occurs when robots approach each other at an obtuse angle with their next waypoints being very close to

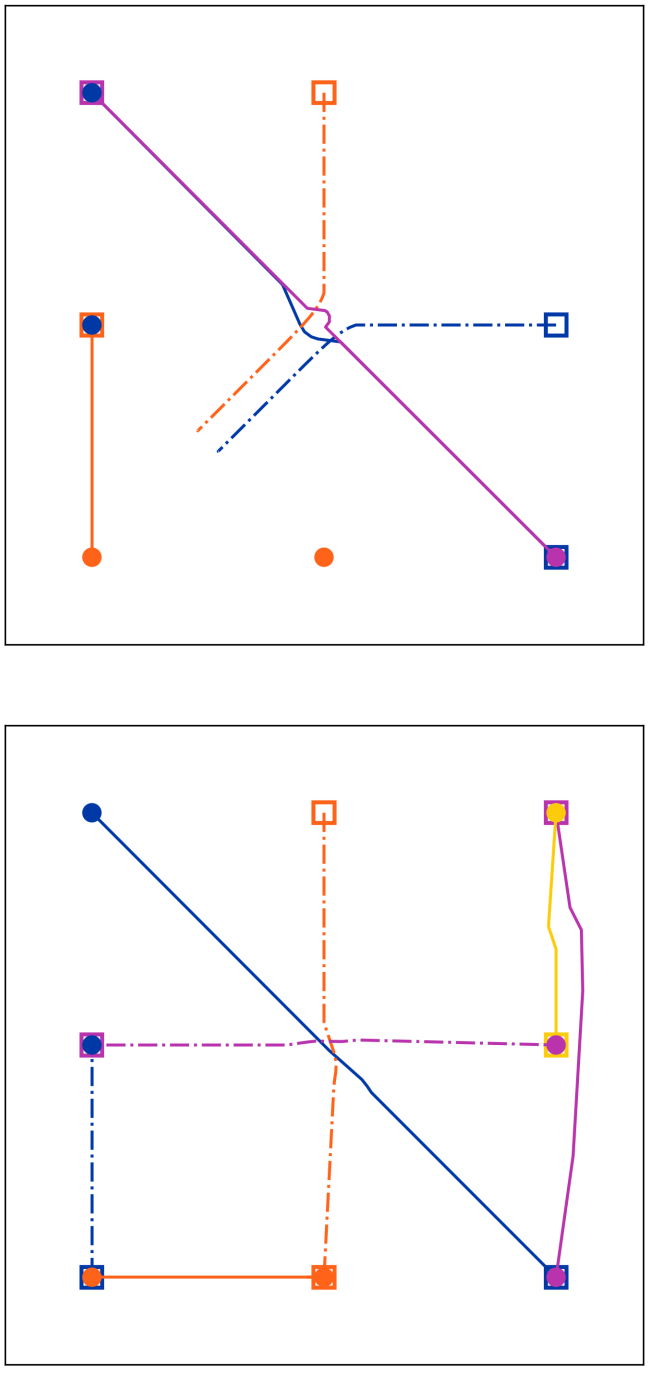


Figure 6.12: It is possible for the ORCA approach to reach a deadlock. This is likely due to the planner.

each other. In most cases, where the next waypoints for the robots are far from each other, the colliding robots will be able to avoid each other with little deviation from their paths, as shown in the figure on the right. But, if the waypoints are too close to each other, the robots will continually want to travel to a point that is directly through the other robot. Since the robot is only capable of knowing the velocity of the other robot, they travel at an angle that bisects them continually until the simulation end or if they reach another obstacle to break the deadlock.

6.4 Replanning Approach

6.4.1 Completeness

The replanning approach's success rate was heavily influenced by the number of robots in the environment. For all environments, the approach was virtually perfect for the trials with two robots, and then the success rate gradually decreased as the number of robots increased. Two groups seem to have formed in the plot shown in Figure 6.13. The first group consists of the empty and random environments which did not experience as extreme of a reduction in success rate with the increase in number of robots. The double bug trap, office, and two corridor environments saw a much more drastic decrease in success rate with the success rate dropping close to 0% in trials with eight robots for the double bug trap and office environments.

6.4.2 Efficiency

The success rate has a great influence of the average distance traveled metric for the replanning approach, shown in Figure 6.14. This is because the robots would not move until all replans were completed due to how the simulation was run. So,

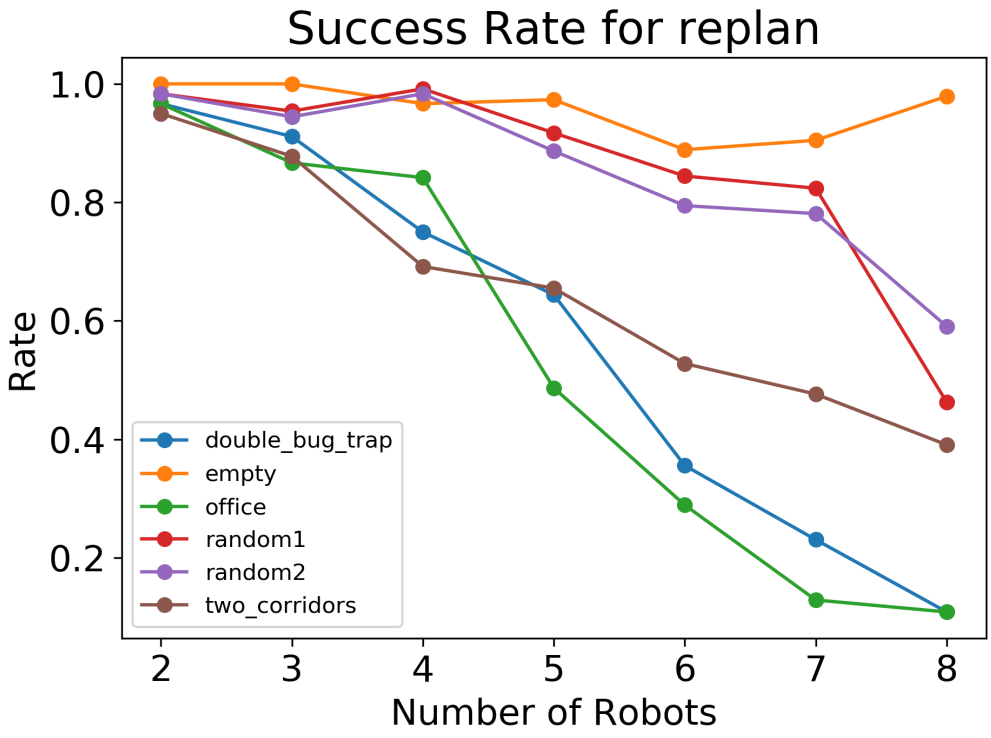


Figure 6.13: Success rate of robots using the replanning approach decreases in all environments, more so in the double bug trap, office, and two corridors environments.

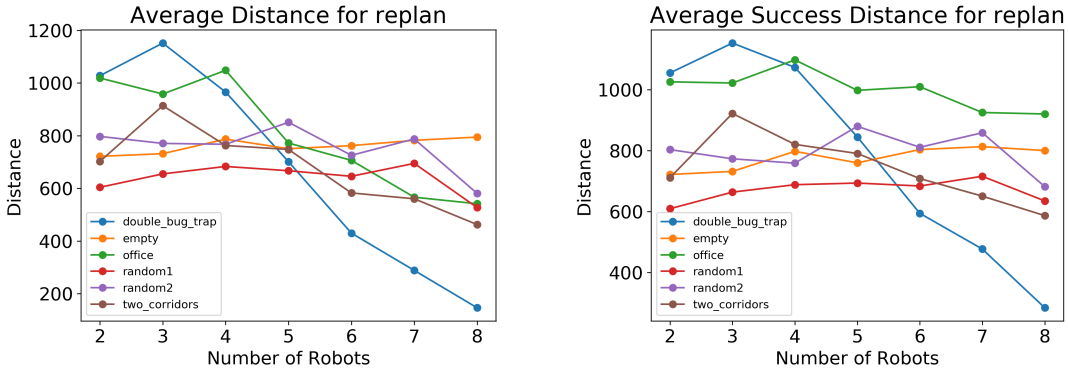


Figure 6.14: Plots for average distance traveled for robots using replanning approach that for most environments the distance traveled was about the same for all variations in number of robots. But, in the double bug trap environment, the average distance traveled decreases rapidly since most robots end up in a deadlock.

if a single robot took a long time to find a feasible path, the simulation could time out before all the robots had the opportunity to replan or to move. Therefore, for cases where the approach was mostly unsuccessful, especially with the double bug trap environment, the average distance traveled decreased with an increase in robots. This measure is accurate and reflects the fact the robots did not move as far since they were stuck in a deadlocked state. Interestingly, in cases where most robots were successful, the average distance traveled remained the same even though the number of robots increased.

This could indicate one of two outcomes. The first would be that the replanning approach was only successful if the goal was close to the start. The second would be that the distance traveled reflects the fact that only certain sets of start and goal pairs were solvable by the approach. Since the distance between these pairs would be the same, the average distance traveled would remain the same since these were the only pairs being completed.

6.4.3 Cost

Figure 6.15 shows plots of the average runtime and the successful average runtime metrics which were also greatly affected by whether or not a robot was successful in traversing its path. The average runtime chart shows that for the office and double bug trap environments, two of the most unsuccessful environments, that the average runtime decreased with a higher number of robots. Both environments showed a similar trend with the average runtime peaking in trials with four robots before steadily decreasing to under 2000 milliseconds in trials with eight robots. This result can be explained by a deficiency in the way the runtimes were recorded. If a robot failed to reach its goal, its runtime was not recorded. This was an oversight in implementation as those cases should be noted as having a runtime of 30000 milliseconds.

This result can be explained a few factors. Both the double bug trap and office environments force the robots to travel through tight spaces. This could cause multiple robots to have to perform a replan. There is a chance that some of those robots will not be able to complete a replan since they would no longer have a feasible path they could follow. Secondly, on average, the distance that a robot would need to travel in these environments is higher than all other environments, as seen in the results from the APF-based and ORCA-based approaches. Due to the distance, timeouts could occur since the robots need more time to traverse the environment. In addition, due to the implementation of the simulation, the robots could not move until all robots had finished their replans. Therefore, one long replan could make the other robots unsuccessful since they would no have enough time to reach their goal.

In Figure 6.15, the plot for the average successful runtime is different from the plots for the same metric for the other approaches. For the other approaches, typically, all environments except for the double bug trap and office environments, the runtime

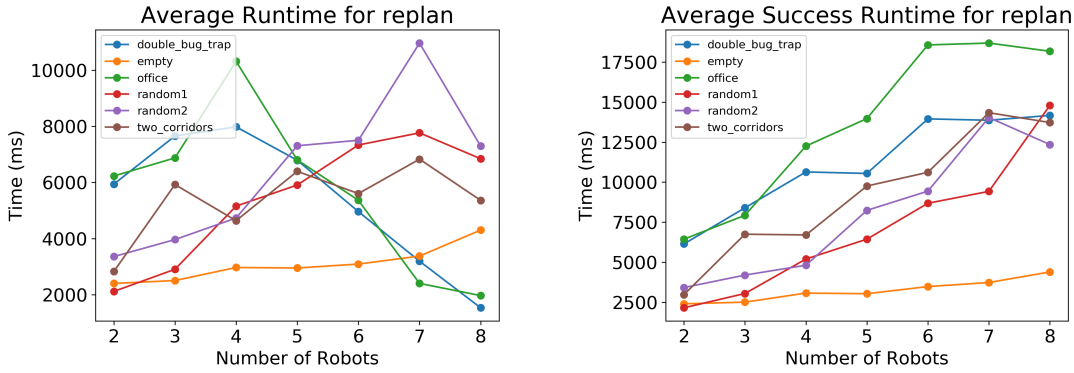


Figure 6.15: Average runtime for robots using the replanning approach in general increases as the number of robots increase for all environments. The average runtime for double bug trap and office environments in the plot on the left is slightly misleading since many robots got deadlocked and were unable to record a runtime. As a result, the average runtime decreased in those environments when the number of robots increased.

is about the same. This is reflected in the average distance traveled as for those environments, the average distance traveled is about the same. Instead, for all environments except for the empty environment, the average runtime for successful robots increased at a faster rate than for the other environments. This is due to the amount of time needed to complete a replan. The other approaches do not require as much calculation time as the replanning process.

6.4.4 Mechanisms of Failure

Figure 6.16 shows the plots for the collision and timeout rate for trials with robots using the replanning approach. The approach resulted in some trials ending in collisions, but more trials ending in timeouts. The collision rate for all environments generally increased with an increase with number of robots in the environment from around 5% in two robot trials and rising to about 15% in eight robot trials. For double bug trap, office, the timeout rate rose to around 80% in trials with more than

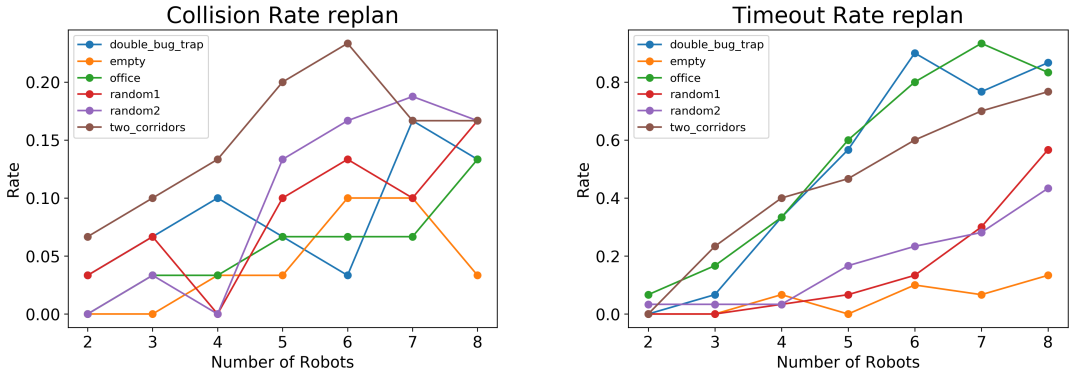


Figure 6.16: The replanning approach would result in some collisions, but most trials ended in timeouts, especially in the double bug trap, office, and two corridors environments.

six robots. For the two corridor environment, the timeout rate gradually increased to about 60%.

For the empty environment, the collision and timeout rates were low. In the random environments, the collision rate jumped from 0% to around 15% as the number of robots increased beyond four robots. The timeout rate for the random environment seemed to increase more rapidly when six or more robots were involved with the rate increasing from about 15% to between 30% and 40%.

6.4.5 Additional Replanning Notes

The amount of nodes added, explored, and sampled per replan were recorded. Additionally, the amount of time those replan instances took was recorded as well. Those instances were further filtered in the categories of whether or not the individual replan was completed or not. The figures for all of these metrics are included in the appendix.

The results show that all of these metrics increased with an increase in number of robots. As previously stated, this can likely be explained by the fact that more

robots in an environment results in a more complex environment since there are more obstacles in the environment. Additionally, the robot-robot interactions become more complex as well.

There are two distinct groups that form here as well. There are the more complex environments which have more constrained spaces and the simpler environments with less constrained spaces. In the simpler environments, there is more space so it is more likely that the replanning algorithm will be able to generate a new path quickly.

6.4.6 Summary of Replanning Approach Results

All the results presented in the previous subsections reflect a common theme about which environments gave the replanning approach the most difficulty. This theme does not become apparent without first starting with the success rate, which affected the other metrics. In Figure 6.13, two groups form. The first group is the empty and random environments. The second consists of the double bug trap, office, and two corridor environments. The replanning approach was more successful in the first group of environments than the second group of environments.

The main difference between these environments is the presence of long corridors or tight spaces. In the empty and random environments, long corridors do not exist and the approach is more successful. In the other environments, long corridors and tight spaces exist. The replanning approach fails in these environments primarily due to timeouts, as shown in Figure 6.16. These timeouts generally occur due to the amount of time the replanning portion of the approach takes.

As shown in the Additional Notes subsection, for all variants in number of robots, the amount of nodes explored in the replan is higher in the tight space or long corridor environments. The replanning approach uses BIT* to replan a path if the robot's

current path is no longer valid. The implementation of BIT* uses a rectangular area to connect the nodes it explores. In areas with tight spaces or long corridors, the BIT* implementation will be unable to find a new path because of this rectangular area since it is unlikely that there is enough space to find a valid configuration. As a result, the replan would timeout after five seconds.

Even if robots were able to complete some successful replans, eventually, they would move into a state where they could be too close to each other such that replans would not longer be feasible.

In the environments lacking these tight spaces, the approach was more successful since it would be more likely for a new path to be found by a replan.

The collisions are most likely due to the non-cooperative nature of the replanning approach as robots could replan paths that would overlap each other.

6.5 Comparison to Single Robot Case

When comparing how the approaches fared in a multi-robot trial to how they fared in a single robot trial, a couple trends emerged. First, the amount of nodes sampled was roughly the same for approaches. For the number of nodes added to the search tree, number of nodes explored by BIT*, and planning time for the initial plan, no discernable pattern emerged. This was likely due to the random nature of the BIT* planning algorithm. Secondly, the runtime of the robots increased as the number of robots in the trial increased when compared to the single robot runtime. This behavior is expected since in the multi-robot trials, the robots will have to handle robot-robot interactions which result in the robots deviating from their near optimal paths. All figures relating to these results are in the appendix.

Approach	Advantages	Drawbacks
APF	Relatively quick and efficient	Unable to handle long corridors or local minimums
ORCA	Relatively quick and efficient	Still reached local minimums and deadlocks
Replanning	Did not get caught in local minimums	Slow and Unable to handle tight spaces

Table 6.2: This table summarizes the strengths and weaknesses of the approaches experimented with in this thesis. Overall, the ORCA-based approach outperformed the others.

6.6 Comparison of Algorithms

In general, the ORCA-based approach out performed the APF-based and the replanning approaches. While the APF-based approach was able to match the ORCA-based approach in terms of speed and efficiency, it was unable to handle all environments as well as the ORCA-based approach. It was difficult to evaluate the replanning approach since most of the robots in those trials were unable to reach their goals before the trial timed out after 30 seconds.

Another drawback for both the APF-based and replanning approaches is how those approaches failed. The APF-based approach would fail mostly due to a robot reaching a local minimum or multiple robots reaching a deadlocked state. The replanning approach would mostly fail due to robots reaching a deadlock.

Both approaches resulted in different behavior when reaching a deadlock. For the APF-based approach, the robots would oscillate in an attempt to move past each other. For the replanning approach, the robots would stop and wait for a replan to complete. These findings are summarized in Table 6.2.

6.7 Comparison to Reactive Multi-robot Path Planning

Trials were run with robots only using reactive algorithms, such as ORCA and APF, to show the effectiveness of the addition of adding a global planner to these algorithms.

The results showed that the addition of a global planner, in this case BIT*, improved the success of the robots reaching their respective goals. This is illustrated in the success rate and success distance metrics. The success rate metric shows that most robots in all environments failed to reach their goals. Success distance supports this as the distances were short in comparison to the distances of the trials with the global planners.

The reactive multi-robot path planning algorithms were only capable of solving problems where a robot had a direct line of sight of its goal from its starting point, as shown in the figures below. Additionally, this result is supported by the success rate in the empty environment being the highest for all reactive algorithms tested. The empty environment did not have obstacles, so all robots had a direct line of sight to their goals.

Chapter 7

CONCLUSION

In this thesis, three approaches were experimented with to address the problem of multi-robot path planning. These approaches can be categorized as decentralized, noncooperative algorithms as robots do not explicitly coordinate with each other and plan their own paths independently. All approaches used the BIT* algorithm as the basis for path planning for each of the robots. Of those approaches, the method that used ORCA to safely avoid the other agents was by far the most successful. The ORCA-based approach was able to successful navigate the agents to their desired goals in almost all occasions.

The other methods failed to match that mark and the agents using this methods failed to reach their goals in a timely manner if at all. The success of the other methods, an APF-based approach and a replanning approach, were heavily dependent on the environment they were deployed in. The APF-based approach struggled with environment containing many environmental local minimums and long, narrow corridors. The replanning approach struggled with corridors and tight spaces.

The replanning approach was the less viable out of all the approaches tested as it was not fast enough to get all robots to their goals within the simulation time limit in the majority of cases. The APF-based approach was more successful that the replanning approach for this reason. But, since the ORCA-based approach performed at such a high success rate in all environments, it can be considered as the only promising approach to handling the multi-robot planning problem in a decentralized, noncooperative way.

7.1 Future Works

There are elements of this thesis that need to be improved and there are also elements of this thesis that could be extended.

As for improvements, the implementation of BIT* that was used in this thesis was many orders of magnitude slower than that of the implementation used in the original paper [6]. This could be due to the fact that this implementation was done in Python. If the algorithm could be reimplemented more efficiently in a language like C++, the outcomes of some of the approaches could be improved. For example, the replanning approach could have been more successful if the BIT* planner was faster.

Another potential reason for the slowness of the planners is that the simulation ran on a single thread. As a result, the approaches tested in this thesis were not totally accurate to their intentions since robots would not be able to plan paths simultaneously. This did not have a great effect on the outcome of the thesis, especially in regards to the replanning algorithm which would have probably resulted in many simulation timeouts any way. In future experiments, a multi-core machine or multiple CPUs should be used to simulate the robots individually to help improve results.

Speaking of replanning, the logic behind how replans would be triggered and how they would be executed could be reworked. A potential solution is to use D*-Lite as an inspiration for the incremental search algorithm in place of LPA*. Since D*-Lite is based on LPA*, this seems like it could be a feasible solution.

A common problem for the artificial potential field and ORCA-based methods was that the robots still managed to be trapped in local minima despite their planned paths. This issue could be attributed to the waypoint management system used in these algorithms. The next waypoint to travel to for these algorithms would only

update if a robot traveled close enough to them. If a robot failed to so, the robot would continually attempt to travel towards their previously determined next waypoint and remain trapped. There could be other approaches to mitigate this problem such as unordering the waypoints. This method would attract the robots to the nearest waypoint regardless of the robot having passed by the waypoints that preceded the nearest waypoint in the path. Another could be to use a random walk method to disturb the robot enough to have it travel out of the local minima.

As for the artificial potential field method, the coordination force used in the approach could be improved to increase the success rate of that approach. But, due to the jittering behavior of the agents using the approach, it may not be worth pursuing it since jittering would not be suitable for physical robots.

Finally, the ORCA-based approached was shown to be the most successful approach used. This could speak to the overall viability of using a sample-based planning method as a global planner and a collision avoidance algorithm as a way of resolving near-collision conditions. Therefore, it could be interesting to see how other collision avoidance and other sample-based planners can work in place of ORCA and BIT* respectively. Additionally, it could be worthwhile to implement the approach on physical robots or at least robots with more complex dynamics to see how the approach handles dynamic constraints. Implementation on physical robots will also enable to see how the approach works on a distributed system.

BIBLIOGRAPHY

[1] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

[2] S. Chen, Z. Yang, Z. Liu, and H. Jin. An improved artificial potential field based path planning algorithm for unmanned aerial vehicle in dynamic environments. In *2017 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, pages 591–596. IEEE, 2017.

[3] H.-T. Chiang, N. Malone, K. Lesser, M. Oishi, and L. Tapia. Path-guided artificial potential fields with stochastic reachable sets for motion planning in highly dynamic environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2347–2354. IEEE, 2015.

[4] V. R. Desaraju and J. P. How. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots*, 32(4):385–403, 2012.

[5] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004. IEEE, 2014.

[6] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3067–3074. IEEE, 2015.

[7] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[8] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; pspace-hardness of the" warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[9] P. Janovskỳ, M. Cáp, and J. Vokrínek. Finding coordinated paths for multiple holonomic agents in 2-d polygonal environment. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1117–1124. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[10] R. Kala. Rapidly exploring random graphs: motion planning of multiple mobile robots. *Advanced Robotics*, 27(14):1113–1122, 2013.

[11] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[12] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[13] S. Koenig, M. Likhachev, and D. Furcy. Lifelong planning a. *Artificial Intelligence*, 155(1-2):93–146, 2004.

[14] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium*

Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), volume 2, pages 995–1001. IEEE, 2000.

[15] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[16] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[17] PyQT. Pyqt reference guide. 2012.

[18] D. Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.

[19] C. Urmson and R. Simmons. Approaches for heuristically biasing rrt growth. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 2, pages 1178–1183. IEEE, 2003.

[20] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[21] G. Wagner and H. Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, pages 3260–3267. IEEE, 2011.

[22] G. Wagner, M. Kang, and H. Choset. Probabilistic path planning for multiple robots with subdimensional expansion. In *2012 IEEE International Conference on Robotics and Automation*, pages 2886–2892. IEEE, 2012.

[23] A. Yershova and S. M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1):151–157, 2007.

.1 Comparison to Single Robot Planning APF

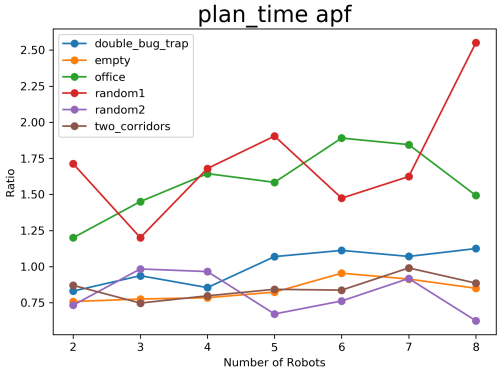
In the more complex environments, it appears that the number of explored nodes was about the same as for the single robot case. In the simpler environments, it seems like more nodes were explored since more robots in the environments lead to creating a more complex environment. As a result, plan time increased accordingly.

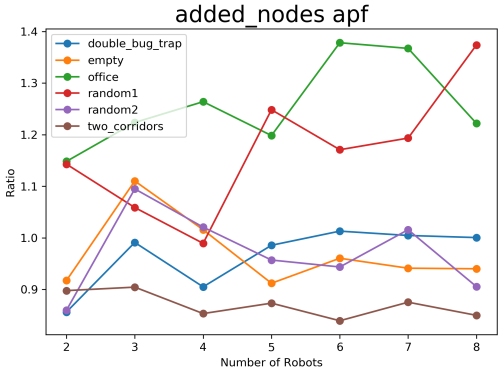
.2 Comparison to Single Robot Planning ORCA

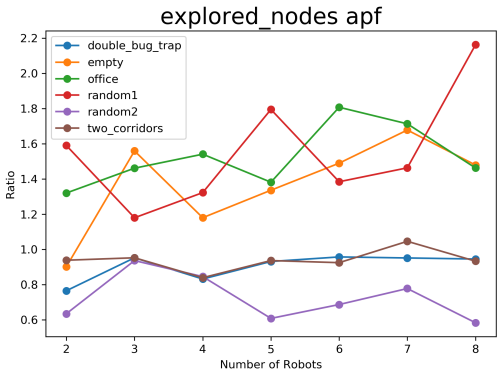
In comparisons with the single robot case, the average runtime was significantly greater than the single robot runtime for all environments. The trends for average runtime matched the trends seen in the costs section. There were more nodes added and explored by the approach than in the single robot case. But, as seen previously this difference was more pronounced in "easier" environments such as the empty and random environments than in the more complex environments. As before, a possible reason for this difference is that the addition of robots in the simpler environments increases the magnitude of complexity of those environments more than the other more complex environments.

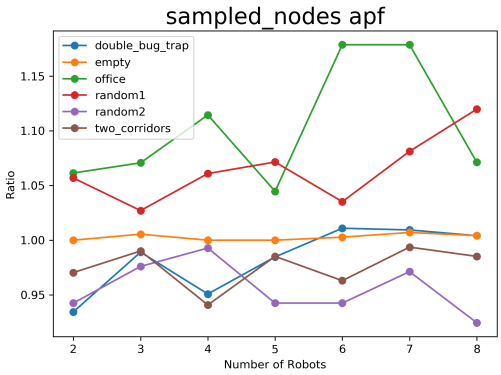
.3 Comparison to Single Robot Planning

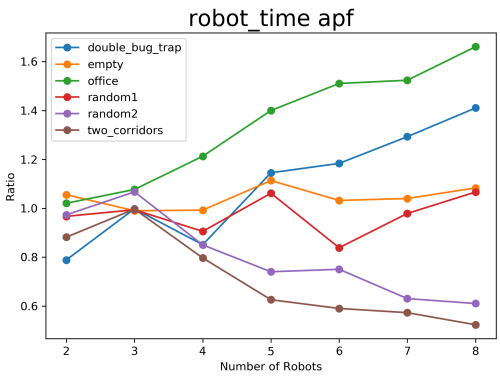
For the single robot comparison, metrics like average runtime and average distance traveled mirrored the results explained the cost and efficiency sections. The strangest outcome is that more nodes were explored and added in the randomized environments compared to all other environments. This result is unexpected since the planning

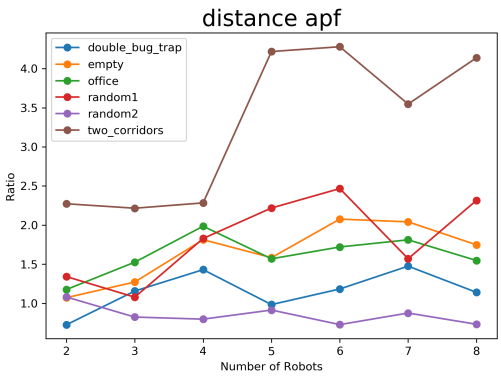

plan_time apf
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

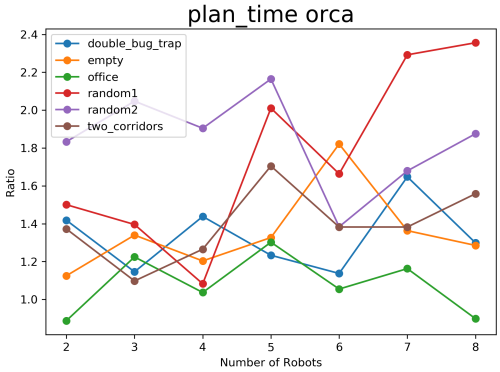

added_nodes apf
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

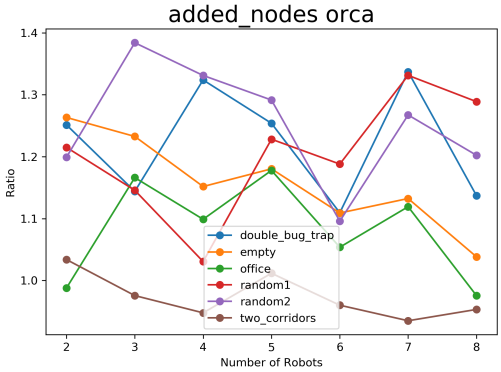

explored_nodes apf
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

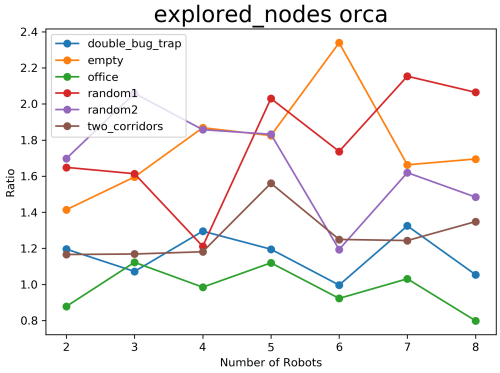

sampled_nodes apf
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

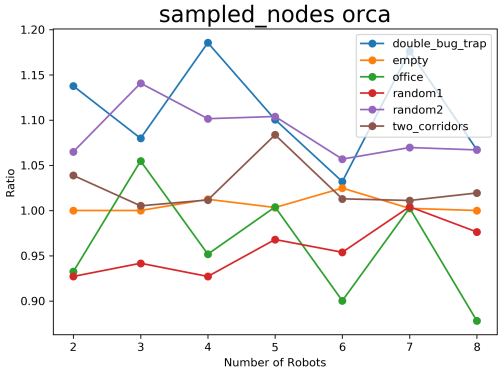

robot_time apf
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

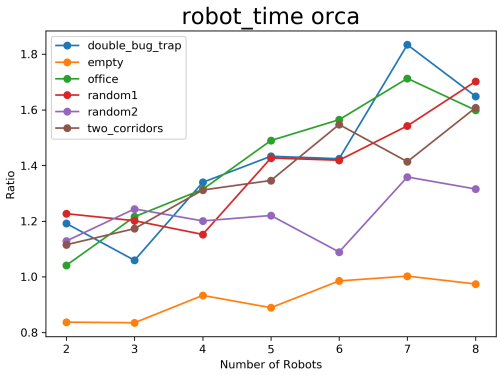

distance apf
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

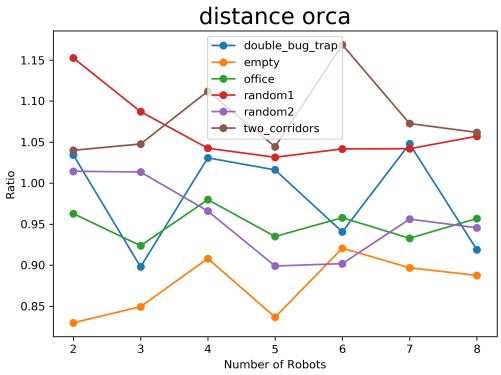

plan_time orca
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots


added_nodes orca
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots


explored_nodes orca
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots


sampled_nodes orca
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

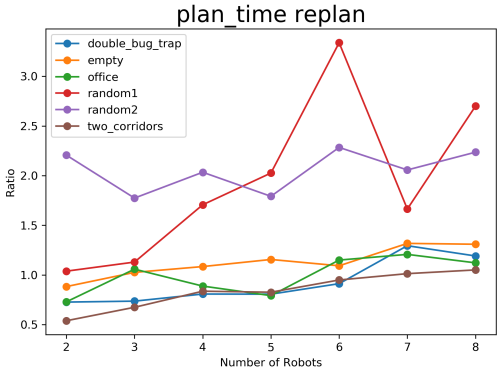

robot_time orca
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

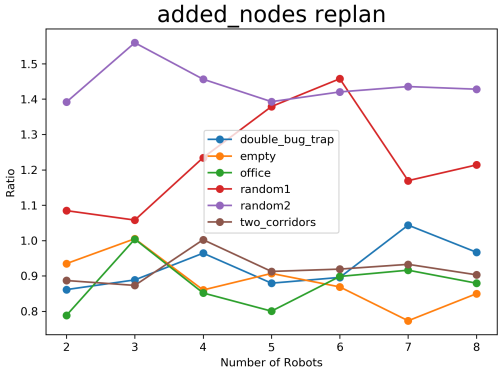

distance orca
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

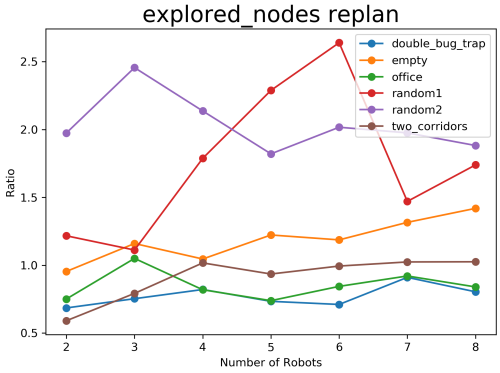
technique used for the APF and ORCA approaches is the exact same as the technique used for the replanning approach.

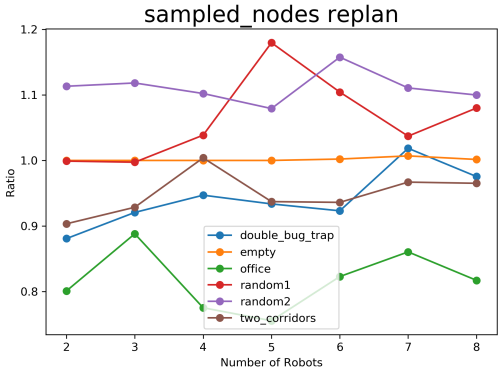
Due to the random nature of the planning algorithm, this could be due to random chance, but since so many trials were run, it would seem unlikely. Perhaps, there could have been an error in recording and these results are also dependant on the success of the robots as well. This could be a possibility since the number of nodes explored for the most difficult environments is around half of that of the single robot case.

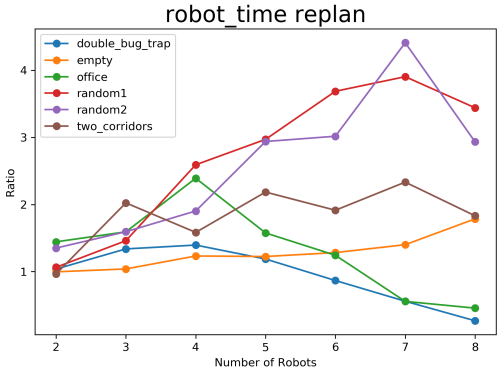
.4 Replanning Metrics

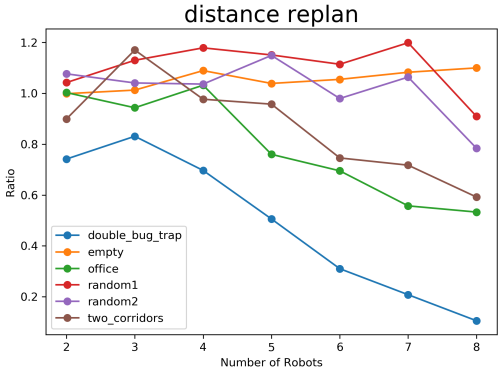

plan_time replan
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

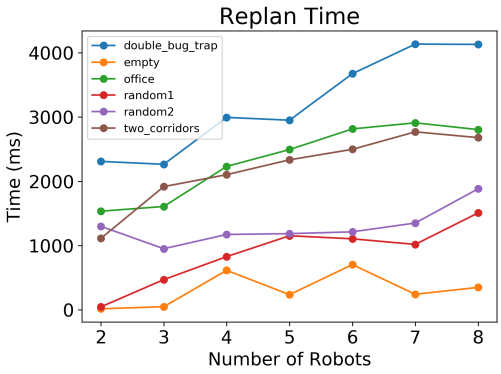

added_nodes replan
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

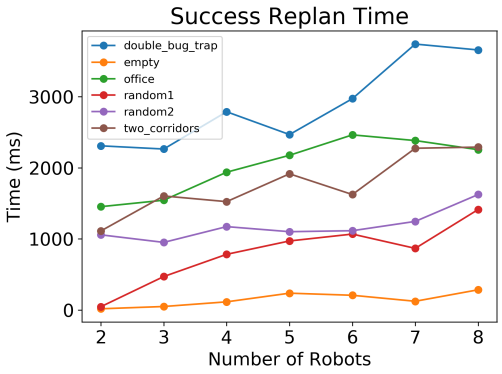

explored_nodes replan
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

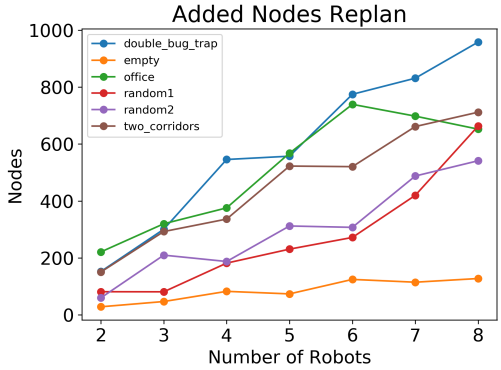

sampled_nodes replan
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

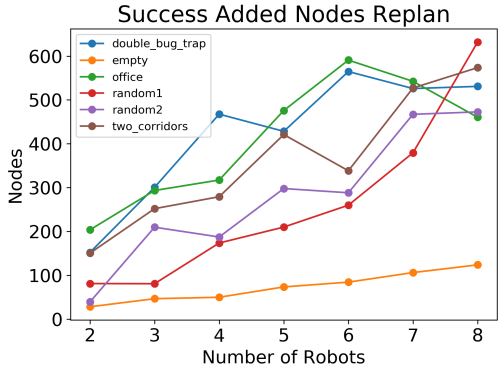

robot_time replan
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

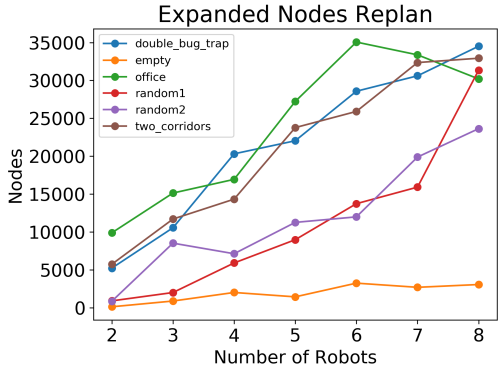

distance replan
double_bug_trap
empty
office
random1
random2
two_corridors
Ratio
Number of Robots

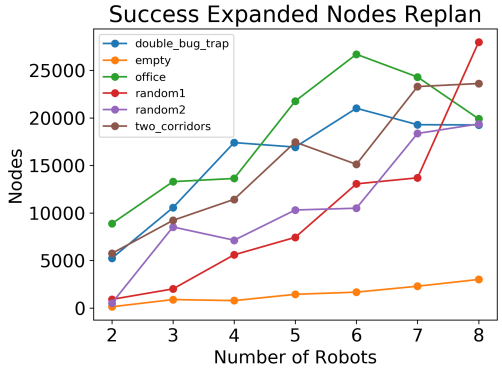

Replan Time
Time (ms)
Number of Robots
double_bug_trap
empty
office
random1
random2
two_corridors

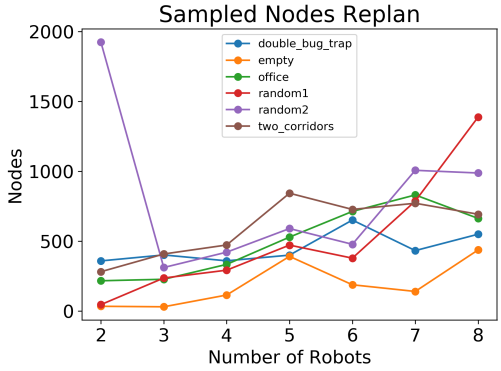

Success Replan Time
Time (ms)
Number of Robots
double_bug_trap
empty
office
random1
random2
two_corridors

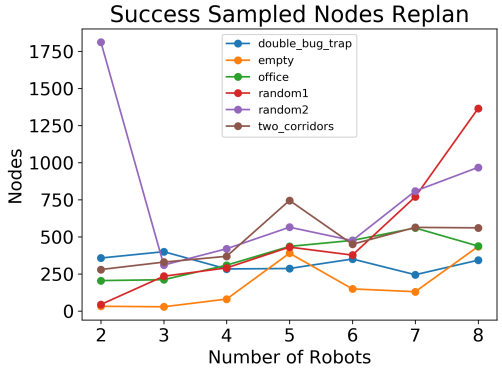

Added Nodes Replan
Nodes
Number of Robots
double_bug_trap
empty
office
random1
random2
two_corridors


Success Added Nodes Replan
Nodes
Number of Robots
double_bug_trap
empty
office
random1
random2
two_corridors


Expanded Nodes Replan
Nodes
Number of Robots
double_bug_trap
empty
office
random1
random2
two_corridors


Success Expanded Nodes Replan
Nodes
Number of Robots
double_bug_trap
empty
office
random1
random2
two_corridors


Sampled Nodes Replan
Nodes
Number of Robots
double_bug_trap
empty
office
random1
random2
two_corridors


Success Sampled Nodes Replan
Nodes
Number of Robots
double_bug_trap
empty
office
random1
random2
two_corridors

```
A*(start, goal)
begin
    Initialize priority queue for nodes;
    Add start node to priority queue;
    while Queue is not empty do
        Pop queue;
        if Popped node is not the goal then
            Calculate the costs for all neighbors to popped node;
            if Neighbor is not in queue or has improved cost then
                Add neighbor to queue and update its parent node;
            end
        end
    end
    if Last node is goal then
        Backtrace from goal node to start node using the parent nodes;
        Return path;
    end
    Return infeasible;
end
```

Algorithm 3: A*(start, goal)

```
RRT(start, goal)
begin
    Initialize tree with starting configuration;
    while Goal has not been reached do
        Sample random point in space;
        Find closest node in tree to sampled point;
        Extend new node in tree from closest node towards sampled point;
        if New node and edge between closest node and new node are
          collision-free then
            Add new node to tree;
        end
    end
    Return tree;
end
```

Algorithm 4: RRT(start, goal)

```
BIT*-Replanning(start, goal)
begin
    Generate initial plan with BIT*;
    while Robot has not reached goal do
        if Condition for replan triggered then
            Rerun BIT* using only one sample per iteration;
            Move robot in direction of its next waypoint;
            if Close to next waypoint then
                Update current waypoint to next waypoint;
            end
        end
    end
end
```

Algorithm 5: BIT*-Replanning(start, goal)

```
BIT*-APF(start, goal)
begin
    Generate initial plan with BIT*;
    while Robot has not reached goal do
        Set next waypoint as goal for APF;
        Calculate next velocity with APF method;
        Move robot using calculated velocity;
        if Close to next waypoint then
            Update current waypoint to next waypoint;
        end
    end
end
```

Algorithm 6: BIT*-APF(start, goal)

```
BIT*-ORCA(start, goal)
begin
    Generate initial plan with BIT*;
    while Robot has not reached goal do
        Calculate desired velocity using robot's current position and next
         waypoint;
        Run ORCA with desired velocity as input;
        Move robot at velocity of velocity output from ORCA;
        if Close to next waypoint then
            Update current waypoint to next waypoint;
        end
    end
end
```

Algorithm 7: BIT*-ORCA(start, goal)

```
BIT*(start, goal)
begin
    Initialize tree with only the start in the tree;
    Initialize the edge and vertex queues as empty queues;
    Set the informed space to be the whole environment;
    while the termination condition has not been reached do
        if the queues are empty then
            Update the informed space;
            Remove all samples that are outside of the informed subproblem;
            Add samples to the space from the informed subproblem;
            Add all vertices in the tree to the vertex queue;
        end
        while there are vertices worth exploring do
            Add edges from the best vertex to new samples that can improve
             the current solution to the edge queue;
            Add edges from the best vertex to vertices in the tree that can
             improve their current cost;
            Get the best edge from the edge queue;
            if the best edge is within the informed subproblem then
                if after collision checking the edge, the edge still could improve
                 the solution then
                    if the edge improves the cost to the edge's endpoint then
                        Update the tree with the new edge;
                        Prune edge queue based on new edge;
                    end
                end
            end
            else
                Flush both queues
            end
        end
    end
    Return the tree
end
```

Algorithm 8: BIT*(start, goal)

		Number of Robots						
Environment	**Metric**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Double Bug Trap	Distance	1094.75	1744.67	2156.5	1483.3	1784.17	2224.21	1719.38
	Runtime	4666.92	5905.19	5036.07	6777.31	7003.83	7651.03	8345.86
	Success	1.0	0.944	0.833	0.94	0.867	0.862	0.921
Empty	Distance	767.75	910.83	1298.25	1134.1	1486.67	1461.57	1251.44
	Runtime	2528.88	2371.57	2377.88	2667.75	2472.82	2491.63	2595.64
	Success	1.0	0.978	0.933	0.96	0.911	0.914	0.942
Office	Distance	1292.25	1676.17	2180.88	1723.6	1889.58	1990.0	1699.75
	Runtime	4290.92	4528.13	5099.74	5885.67	6350.99	6406.85	6983.0
	Success	0.967	0.822	0.792	0.867	0.8	0.724	0.817
Random 1	Distance	963.0	774.5	1313.0	1592.7	1771.08	1127.14	1662.75
	Runtime	2331.28	2396.07	2183.06	2558.97	2021.46	2359.73	2570.72
	Success	0.933	0.967	0.85	0.86	0.739	0.738	0.658
Random 2	Distance	1721.0	1313.0	1270.5	1455.8	1159.75	1395.0	1166.56
	Runtime	2493.08	2734.13	2177.93	1896.71	1922.57	1615.35	1563.62
	Success	0.85	0.8	0.7	0.607	0.572	0.471	0.483
Two Corridors	Distance	1897.5	1850.0	1906.88	3522.4	3573.58	2961.5	3455.81
	Runtime	2688.28	3037.77	2428.71	1907.66	1798.99	1746.22	1595.61
	Success	0.85	0.878	0.708	0.587	0.556	0.476	0.508

Table .1: Metrics for All Robots using BIT*-APF

		Number of Robots						
Environment	**Metric**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Double Bug Trap	Distance	1094.75	1377.35	1266.6	1312.55	1260.29	1482.85	1374.37
	Runtime	4666.92	6252.55	6043.28	7209.91	8081.34	8876.88	9063.38
Empty	Distance	767.75	726.31	755.09	811.56	771.95	772.19	777.94
	Runtime	2528.88	2425.47	2547.72	2778.9	2714.07	2725.22	2756.43
Office	Distance	1031.12	1145.68	1101.32	1108.96	1183.13	1191.41	1215.46
	Runtime	4438.88	5507.19	6441.78	6791.15	7938.74	8851.57	8550.61
Random 1	Distance	689.2	682.41	652.35	701.86	673.2	720.19	754.27
	Runtime	2497.8	2478.69	2568.3	2975.55	2735.81	3197.05	3904.89
Random 2	Distance	784.41	827.92	753.04	760.71	720.87	751.06	708.23
	Runtime	2933.04	3417.67	3111.32	3126.45	3359.83	3426.49	3235.07
Two Corridors	Distance	830.88	893.92	850.76	765.51	723.3	726.3	615.12
	Runtime	3162.69	3460.75	3428.76	3251.69	3238.19	3667.06	3138.9

Table .2: Metrics for Successful Robots using BIT*-APF

		Number of Robots						
Environment	**Metric**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Double Bug Trap	Distance	1289.2	1119.4	1284.83	1266.55	1172.47	1306.11	1145.54
	Runtime	5805.43	5160.59	6524.62	6980.68	6937.43	8931.31	8030.43
	Success	1.0	1.0	0.95	0.953	0.994	0.952	0.929
Empty	Distance	674.75	690.65	738.38	680.26	748.59	729.21	721.6
	Runtime	2267.9	2261.96	2527.93	2408.33	2668.63	2715.54	2638.98
	Success	1.0	0.956	0.983	1.0	0.989	1.0	0.983
Office	Distance	1039.96	997.79	1058.37	1009.91	1034.62	1007.57	1033.53
	Runtime	4618.32	5391.89	5827.68	6605.58	6936.56	7594.51	7088.28
	Success	1.0	0.967	0.975	0.993	0.989	0.981	0.988
Random 1	Distance	684.15	645.35	618.9	612.38	618.41	618.55	627.56
	Runtime	2517.58	2465.87	2364.76	2927.71	2911.52	3164.53	3492.0
	Success	0.983	0.989	1.0	0.993	1.0	1.0	1.0
Random 2	Distance	777.96	777.26	740.79	689.47	691.66	733.24	725.22
	Runtime	3001.17	3307.24	3192.93	3244.43	2896.08	3611.95	3497.58
	Success	0.983	1.0	1.0	0.987	0.978	1.0	0.992
Two Corridors	Distance	758.52	764.13	811.01	761.69	852.3	782.32	774.43
	Runtime	2888.12	3037.77	3396.93	3485.59	4005.92	3660.36	4161.89
	Success	1.0	1.0	1.0	0.987	0.994	0.99	1.0

Table .3: Metrics for All Robots Using BIT*-ORCA

		Number of Robots						
Environment	**Metric**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Double Bug Trap	Distance	1289.2	1119.4	1251.11	1311.41	1167.86	1346.73	1227.15
	Runtime	5805.43	5160.59	6868.02	7322.39	6976.19	9377.88	8642.62
Empty	Distance	674.75	687.7	739.56	680.26	747.57	729.21	721.05
	Runtime	2267.9	2367.16	2570.77	2408.33	2698.61	2715.54	2683.71
Office	Distance	1039.96	1006.88	1066.81	1012.22	1040.96	1019.9	1037.2
	Runtime	4618.32	5577.82	5977.1	6649.91	7014.5	7741.98	7178.01
Random 1	Distance	684.29	647.48	618.9	613.47	618.41	618.55	627.56
	Runtime	2560.25	2493.57	2364.76	2947.36	2911.52	3164.53	3492.0
Random 2	Distance	781.44	777.26	740.79	690.72	694.93	733.24	725.7
	Runtime	3052.03	3307.24	3192.93	3288.27	2961.9	3611.95	3526.97
Two Corridors	Distance	758.52	764.13	811.01	761.77	851.09	779.23	774.43
	Runtime	2888.12	3037.77	3396.93	3532.7	4028.3	3695.56	4161.89

Table .4: Metrics for Successful Robots Using BIT*-ORCA

		Number of Robots						
Environment	**Metric**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Double Bug Trap	Distance	1027.75	1151.83	965.63	701.1	429.92	288.24	146.25
	Runtime	5938.65	7649.93	7977.74	6788.37	4959.99	3194.62	1534.63
	Success	0.967	0.911	0.75	0.644	0.356	0.23	0.108
Empty	Distance	721.5	731.67	787.0	750.6	762.17	782.36	794.69
	Runtime	2401.98	2505.42	2969.78	2951.27	3091.92	3375.32	4301.64
	Success	1.0	1.0	0.967	0.973	0.889	0.905	0.979
Office	Distance	1019.0	958.33	1048.38	772.1	706.25	566.29	541.5
	Runtime	6226.07	6872.14	10312.39	6799.93	5364.43	2402.43	1968.81
	Success	0.967	0.867	0.842	0.487	0.289	0.129	0.108
Random 1	Distance	604.0	654.83	683.13	666.93	645.75	694.79	526.94
	Runtime	2117.43	2905.41	5154.56	5906.18	7331.31	7763.42	6842.1
	Success	0.983	0.954	0.992	0.917	0.844	0.824	0.463
Random 2	Distance	797.25	770.67	767.38	850.9	725.42	787.71	580.6
	Runtime	3358.12	3963.17	4730.92	7304.26	7499.11	10960.9	7292.07
	Success	0.983	0.944	0.983	0.887	0.794	0.781	0.591
Two Corridors	Distance	702.25	914.0	762.88	747.72	582.5	560.29	462.42
	Runtime	2827.1	5922.32	4636.65	6393.43	5601.73	6826.69	5359.95
	Success	0.95	0.878	0.692	0.655	0.528	0.476	0.391

Table .5: Metrics for All Robots Using BIT*-Replanning

		Number of Robots						
Environment	**Metric**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Double Bug Trap	Distance	1054.91	1152.99	1073.17	844.63	593.91	476.81	283.85
	Runtime	6143.43	8396.27	10636.99	10543.64	13949.97	13866.0	14165.81
Empty	Distance	721.5	731.67	797.46	759.76	803.34	812.84	800.04
	Runtime	2401.98	2505.42	3072.19	3032.13	3478.41	3730.62	4393.17
Office	Distance	1025.69	1021.92	1097.67	997.81	1009.62	925.0	920.19
	Runtime	6440.76	7929.4	12252.35	13972.45	18569.19	18685.56	18173.65
Random 1	Distance	609.15	663.61	687.98	693.38	683.68	715.66	634.73
	Runtime	2153.32	3045.43	5197.87	6439.07	8681.82	9423.8	14793.73
Random 2	Distance	803.14	772.94	758.77	879.59	810.94	858.57	681.68
	Runtime	3415.03	4196.29	4811.1	8237.89	9439.44	14035.29	12348.62
Two Corridors	Distance	710.79	921.84	820.3	790.42	708.16	650.4	586.48
	Runtime	2975.89	6746.95	6703.59	9758.39	10613.8	14336.04	13723.82

Table .6: Metrics for Successful Robots Using BIT*-Replanning