Dynamic Path Planning and Replanning for Mobile Robots using RRT*

Devin Connell
Advanced Robotics and Automation Lab
Department of Computer Science and Engineering
University of Nevada, Reno NV, 89519
Email: devin.connell@gmail.com

Hung Manh La
Advanced Robotics and Automation Lab
Department of Computer Science and Engineering
University of Nevada, Reno NV, 89519
Email: hla@unr.edu

***Abstract*—It is necessary for a mobile robot to be able to efficiently plan a path from its starting, or current, location to a desired goal location. This is a trivial task when the environment is static. However, the operational environment of the robot is rarely static, and it often has many moving obstacles. The robot may encounter one, or many, of these unknown and unpredictable moving obstacles. The robot will need to decide how to proceed when one of these obstacles is obstructing it's path. A method of dynamic replanning using RRT* is presented. The robot will modify it's current plan when an unknown random moving obstacle obstructs the path. Various experimental results show the effectiveness of the proposed method.**

I. Introduction

Path planning has been one of the most researched problems in the area of robotics. The primary goal of any path planning algorithm is to provide a collision free path from a start state to an end state within the configuration space of the robot. Probabilistic planning algorithms, such as the Probabilistic Roadmap Method (PRM) [1] and the Rapidly-exploring Random Tree (RRT) [2], provide a quick solution at the expense of optimality. Since its introduction the RRT algorithm has been one of the most popular probabilistic planning algorithms. The RRT is a fast, simple algorithm that incrementally generates a tree in the configuration space until the goal is found.

The RRT has a significant limitation in finding an asymptotically optimal path, and has been shown to never converge to an asymptotically optimal solution [3] [4]. There is extensive research on the subject of improving the performance of the RRT. Simple improvements such as the Bi-Directional RRT and the Rapidly-exploring Random Forest (RRF) improve the search coverage and speed at which a single-query solution is found. The Anytime RRT [5] provides a significant improvement in cost-based planning. The RRT* algorithm provides a significant improvement in the optimality of the RRT and has been shown to provide an asymptotically sub-optimal solution [3].

Since the introduction of the RRT* algorithm, research has expanded to discover new ways to improve upon the algorithm. Research includes adding heuristics [6] [7] or bounds [8] to the algorithm in order to maintain the convergence of the algorithm but reduce the execution time. Additional research attempts to guide the algorithm through intelligent sampling [9], or guided sampling through an artificial potential field [10].

In many scenarios the operational environment is rarely static. The path from a single query will often be obstructed during execution. For that reason the topic of replanning is very important to robotic path planning. It is not feasible to discard an entire search tree and start over. One method is to store waypoints and regrow trees called the ERRT [11]. Another method (DRRT) is to place the root of the tree at the goal location, so that only a small number of branches may be lost or invalidated when replanning [12]. The Multipartite RRT maintains a set of subtrees that may be pruned and reconnected, along with previous states to guide regrowth. It is essentially a combination of DRRT and ERRT [13]. More recently the RRT* algorithm has been incorporated into replanning. RRT^X is an algorithm that uses RRT* to continuously update the path during execution [14]. The RRT^X is able to compensate for instantaneous changes in the static environment which is outside the scope of this work.

The contribution of this paper is the method using the RRT* algorithm for replanning in a dynamic environment with random, unpredictable moving obstacles. Also included is the comparison of RRT and RRT* algorithms in a complex 2-D environment.

The remainder of this paper is organized as follows: Section II provides an overview of the RRT* algorithm. Section III will present the replanning method using RRT* in a dynamic environment. Section IV contains the results from all simulations. Section V presents the conclusions and future work.

II. Robot Path Planning using the RRT*

The RRT* algorithm provides a significant improvement in the quality of the paths discovered in the configuration space over it's predecessor the RRT. The quality of the path is determined by the cost associated with moving from the start location to the end location. While RRT* does produce higher quality paths, the algorithm does have a longer execution time. The longer execution time of RRT* is due to the algorithm making many additional calls to the local planner in order to continuously improve the discovered paths. RRT* operates in a very similar way as RRT. The algorithm builds a tree using random samples from the configuration space of the robot and

Algorithm 1: $T = (V, E) \leftarrow$RRT*(q_{init})

```
1  T ←InitializeTree()
2  T ←InsertNode(∅,q_init,T)
3  for k ← 1 to N do
4      q_rand ←RandomSample(k)
5      q_nearest ←NearestNeighbor(q_rand,Q_near,T)
6      q_min ←ChooseParent(q_rand,Q_near,q_nearest, Δq)
7      T ←InsertNode(q_min, q_rand, T)
8      T ←Rewire(T, Q_near, q_min, q_rand)
9  end
```

Algorithm 2: $q_{min} \leftarrow$ChooseParent(q_{rand},Q_{near},$q_{nearest}$,Δq)

```
1  q_min ← q_nearest
2  c_min ←Cost(q_nearest) + c(q_rand)
3  for q_near ∈ Q_near do
4      q_path ←Steer(q_near, q_rand, Δq)
5      if ObstacleFree(q_path) then
6          c_new ←Cost(q_near) + c(q_path)
7          if c_new < c_min then
8              c_min ← c_new
9              q_min ← q_near
10         end
11     end
12 end
13 return q_min
```

connects new samples to the tree as they are discovered. There are two primary differences between RRT and RRT*. The first difference is in the method that new edges are added to the tree. The second difference is an added step to change the tree in order to reduce path cost using the newly added vertex. Each of these differences contributes to the improvement of discovered paths over time and is reason RRT* will converge to an asymptotically sub-optimal solution.

When a random vertex is added to the tree, the RRT will select the nearest neighbor as the parent for this new vertex and edge. RRT* will select the best neighbor as the parent for the new vertex. While finding the nearest neighbor, RRT* considers all the nodes within a neighborhood of the random sample. RRT* will then examine the cost associated with connecting to each of these nodes. The node yielding the lowest cost to reach the random sample will be selected as the parent, and the vertex and edge are added accordingly.

The RRT* algorithm begins in the same way as the RRT. However, when selecting the nearest neighbor the algorithm also selects the set of nodes, Q_{near}, in the tree that are in the neighborhood of the random sample q_{rand}. Line 6 of Algorithm 3 is the first major difference between RRT* and the RRT. Instead of selecting the nearest neighbor to the random sample, the $ChooseParent()$ function will select the best parent from the neighborhood of nodes.

Algorithm 4 describes the $ChooseParent()$ function. This function maintains the node with the lowest total cost for reaching q_{rand}. At line 1 of Algorithm 4 the nearest neighbor,

Algorithm 3: $T \leftarrow$Rewire(T, Q_{near}, q_{min}, q_{rand})

```
1  for q_near ∈ Q_near do
2      q_path ←Steer(q_rand, q_near)
3      if ObstacleFree(q_path) and Cost(q_rand) + c(q_path) <
         Cost(q_near) then
4          T ←ReConnect(q_rand, q_near, T)
5      end
6  end
7  return T
```

$q_{nearest}$, is considered the minimum cost neighbor, or q_{min}. On line 2 the cost associated with reaching the new random sample q_{rand} by using $q_{nearest}$ as the parent is stored as the current best cost, or c_{min}. The algorithm then searches the set of nodes in the neighborhood of q_{rand}. The $Steer()$ function on line 4 of Algorithm 4 will return a path from the nearby node, q_{near} to q_{rand}. If this path is obstacle free and has a lower cost than the current minimum cost, then the nearby node becomes the best neighbor, q_{min} and this cost becomes the best cost c_{min} (lines 7-9 of Algorithm 4). When all nearby nodes have been examined the function returns the best neighbor. The new random node is inserted into the tree using q_{min} as the parent. The next step is the second major difference between the RRT* and the RRT algorithms. Line 8 of Algorithm 3 calls the $Rewire()$ function.

The $Rewire()$ function, described in Algorithm 5, changes the tree structure based on the newly inserted node q_{rand}. This function again uses the nearby neighborhood of nodes, Q_{near}, as candidates for rewiring. The $Rewire()$ function uses the $Steer()$ function to get the path, except this time the path will start from the new node, q_{rand} and go to the nearby node q_{near}. If this path is obstacle free and the total cost of this path is lower than the current cost to reach q_{near} (line 3 of Algorithm 5). Then the new node q_{rand} is a better parent than the current parent of q_{near}. The tree is then rewired to remove the edge to the current parent of q_{near}, and add an edge to make q_{rand} the parent of q_{near}. This is done using the $ReConnect()$ function on line 4 of Algorithm 5.

The functions $ChooseParent()$ and $Rewire()$ change the structure of the search tree when compared to the RRT algorithm. The tree generated by the RRT has branches that move in all directions. The tree generated by the RRT* algorithm rarely has branches that move back in the direction of the parent. The $ChooseParent()$ function ensures edges are created and always moving away from the start location. The $Rewire()$ function changes the internal structure of the tree to ensure internal vertices do not add unnecessary steps on any discovered path. The $ChooseParent()$ and $Rewire()$ functions guarantee the paths discovered will be asymptotically sub-optimal because these functions are always minimizing the costs to reach each node within the tree.

III. Dynamic Replanning

A. Overview

A real world environment is not static, and it is full of moving obstacles. These obstacles are often moving in unpre-

Algorithm 4: ExecutePath()

```
1  SetObsDestination(numObs)
2  SetObsVelocities(numObs)
3  SetRobotDestination()
4  SetRobotVelocity()
5  while robotLocation! = GOAL do
6      UpdateObsLocation(numObs)
7      UpdateRobotLocation()
8      if Replan then
9          DoReplan()
10     end
11 end
```

Algorithm 5: UpdateRobotLocation()

```
1  robotLocation ← robotLocation + robotVelocity
2  if robotLocation == robotDestination then
3      robotDestination ←GetNextPathLocation()
4      SetRobotVelocity()
5  end
6  while obsIndex < numObs do
7      obsDistance ←GetDistance(robotLocation,...
8              ObsLocation(obsIndex))
9      if obsDistance < robotRange then
10         obs_path ←Steer(robotLocation, obsLocation)
11         if ObstacleFree(obs_path) then
12             if IsPathBlocked(obsIndex) then
13                 Replan ←TRUE
14             end
15             SetObsVisible(obsIndex)
16         end
17     end
18 end
```

dictable directions, which makes planning tasks to avoid them difficult. When a moving obstacle is known and is following a known trajectory, the configuration space can be modified to account for this trajectory. When the obstacle is unknown, the robot will need to be able to dynamically determine a course of action in order to avoid a collision. In this section a method of dynamic replanning is proposed in order to avoid a random obstacle when it is detected by the robot.

B. Simulation Environment

For the following simulations the environment remains very similar. The robot is given a configuration space from which to build a tree using RRT* and determine the best path to reach the goal configuration from the start configuration. In all the experiments below, the robot was allowed a tree of varying sizes to evaluate the performance with different node densities. During the simulation a few random moving obstacles are added to the environment, described in the next section. These obstacles represent a region of the configuration space that would be a collision if the robot were to enter.

1) Path Execution: After the initial planning process, the robot begins to execute the optimal path found by the search tree. The robot traverses the optimal path by selecting the next node and required velocity vector to reach it, see lines 3 and 4 of Algorithm 6. This process is described in Algorithm 7 below. When the vertex is reached the robot changes the velocity vector to move toward the next node. This process continues until the robot reaches the goal node. If the robot encounters a random moving obstacle that is obstructing the path a replan event occurs.

C. Random Moving Obstacles

The random moving obstacles force the robot to dynamically plan around the obstacle using RRT*. In order for the obstacles to move about the environment, a graph is created to provide the paths between the static obstacles, and the vertices are the intersections of these paths. Upon initialization of the simulation the obstacles are placed at random vertices. The vertices are chosen such that the robot will have a chance to move before encountering a random obstacle. When the simulation begins the moving obstacles choose a random adjacent vertex and begins moving toward that vertex, see lines 1 and 2 of Algorithm 6. When the vertex is reached a new random vertex is chosen and the obstacle moves in the new direction, line 6 of Algorithm 6.

1) Random Obstacle Detection: Robots operating in a real world scenario will have sensors, such as a LIDAR, to detect both static and dynamic obstacles. Sensors are not included in this simulation. Instead a detection range is placed on the robot. The simulation controls whether or not a moving obstacle is within the detection range of the robot (lines 7 and 8 of Algorithm 7). If a moving obstacle is within range the $Steer()$ function is used, by the simulation, to determine if any static obstacles are blocking the robot's line of sight to the moving obstacle.

The obstacle must be observed for a minimum of two time steps in order to determine the direction that the obstacle is moving. Once the direction is observed the robot can determine if the moving obstacle is blocking the path or not, line 11 of Algorithm 7. If the robot decides that the path is blocked, the replanning event begins.

D. Path Replanning

Path replanning begins with the determination of whether or not the moving obstacle is blocking the path, described in the next section. Algorithm 8, below, lists all the steps executed during the replanning process. The next step is to find the location along the optimal path that is beyond the obstacle. Next, the tree generated by RRT* is modified and expanded in order to find a path around the obstacle. Finally, the best path around the obstacle is chosen and the execution of this sub-path begins. Each of these steps is described in the following sub-sections.

1) Path Obstruction: The method for determining if the moving obstacle is blocking the path is a series of trigonometric functions using a direction vector from the robot to the moving obstacle and a comparison between the robot

Algorithm 6: $T \leftarrow$DoReplan()

1 InvalidateNodes()
2 GetReplanGoalLocation()
3 SetReplanSamplingLimits()
4 Rewire(T, Q_{all}, $NULL$, q_{robot})
5 RRT*(q_{robot})
6 SetReplanPath()

velocity vector and the moving obstacle velocity vector. Since the configuration space is 2-Dimensional, the inverse tangent can be used to find the angles of the vectors. To obtain the direction vector to the moving obstacle the equation is:

$$angle_{direction} = atan2((Y_{obs} - Y_{robot}), (X_{obs} - X_{robot})). \quad (1)$$

Where (X_{robot}, Y_{robot}) is the position of the robot and (X_{obs}, Y_{obs}) is the position of the obstacle. This will return an angle in degrees over the range $(-180, 180)$. Similarly the angle of the robot's velocity vector can be obtained using the following equation:

$$angle_{V_{robot}} = atan2(V_j, V_i). \quad (2)$$

Where V_i and V_j are the X and Y components of the robot velocity vector. Using the angles from (1) and (2) the difference can be taken to see if they are similar. If the absolute value of the difference between the two angles is less than some threshold, then the robot is moving toward the moving obstacle. Note, the angle difference is normalized to be in the range$(-180, 180)$ before the absolute value is taken. This is done for all angle comparisons:

$$|angle_{direction} - angle_{V_{robot}}| < angle_{thresh}. \quad (3)$$

If the robot is moving in the direction of the random obstacle the velocity vectors are examined. Substituting the obstacle velocity into (2) above the angle of the obstacle velocity can be obtained. Next, the differences between the velocity vectors is found:

$$angle_{V_{diff}} = |angle_{V_{robot}} - angle_{V_{obs}}|. \quad (4)$$

There are three possibilities from this point. If the angle difference between the velocity vectors is less than the angle threshold, then the robot and the obstacle are moving in a similar direction. Second, if the angle difference between the velocity vectors is greather than $180 - angle_{thresh}$, then the robot and the obstacle are approximately moving toward each other. Last, if the angle difference falls outside of these ranges the moving obstacle and the robot are moving in different directions.

For the first case: The robot will simply follow the obstacle, until the obstacle changes direction, or the path takes the robot away from the obstacle. The robot will then choose from one of the other conditions. For the second case: The robot quickly activates a replan event to get out of the way. The random obstacle may move out of the way on its own, but there is no way of predicting that will occur. Finally, if the robot and the obstacle are moving in different directions, the robot ignores the obstacle unless it gets too close. This third condition catches the event that the robot moves out from a corner and a random obstacle is detected very close by. This event is best summed up with the following example: When two people approach a hallway intersection they will run into each other if they continue on their current course, it is only when they see each other that they can adjust to avoid a collision.

2) Select Replan Goal Location: The second step in the replan event, line 2 in Algorithm 8, is to find a location that will navigate the robot around the random obstacle. First, any nodes that are in collision with a random moving obstacle are invalidated, not deleted. The only exception is the goal location of the optimal path. After this step an assumption had to be made to simplify and speed up the rest of the replanning process. The assumption is that the robot is currently following the best path in order to reach the goal location and should return to this path after the moving obstacle is avoided. Using this assumption only the nodes along the optimal path are examined. Nodes that are farther from the robot than the random obstacle are candidate nodes. The node on the optimal path that is immediately following the node that is closest to the obstacle, without colliding, will be the replan goal location.

3) Modify Search Tree: The third step is to modify the original search tree in order to find a way around the moving obstacle. First a node is added to the tree at the robot's current location. Using the distance to the replan goal node as a metric, a sampling area is established, line 3 in Algorithm 8. Then, using $Rewire()$, every node within the sampling area is rewired such that the robot's current location becomes the parent of that node. New nodes are then sampled within this area and added to the tree using RRT*. Since there are already many nodes in the tree only a small number will need to be added. However, the number of nearest neighbors used during the $ChooseParent()$ function and the $Rewire()$ function is increased. This increase allows each new node to direct the existing tree toward the replan goal location.

4) Sub-path Selection and Execution: When the search tree modification is complete the best path to the replan goal location is found. Path execution will begin again as it did at the beginning of the simulation. When the robot reaches the replan goal location, the execution of the original optimal path resumes. If the robot encounters another moving obstacle and determines the path is obstructed again. The replanning is repeated, however the replan goal location will always be a node on the original optimal path.

IV. Results

A. The Simulation Environment

The environment for all of the experiments is a complex 2-Dimensional environment that will also serve as the configuration space for the robot. The environment is complex due to the number of obstacles and several narrow passages. There are also several sub-optimal paths where the algorithm may get stuck. For each experiment the path cost is measured

in Euclidean distance. The environment is also intended to mimic a potential real world situation where there would be streets or sidewalks and open areas such as parks and plazas. In the RRT* results presented below, the algorithm is allowed a maximum of 5000 nodes. The optimal path length is 98.48 units.

B. RRT* Results

The RRT* evaluation was conducted in the following way: There is not a growth factor for extending the tree and the tree is goal oriented. The expecation as the tree grows will be long branches. These branches are often inefficient, the RRT* $Rewire()$ function will remove these long inefficient branches as the algorithm executes in favor of shorter, lower cost branches. The algorithm also has a maximum number of nearest neighbors, or neighborhood size that is configurable to the algorithm. This implementation of RRT* has a maximum number of nearest neighbors equal to 1% of the total number of nodes.

Fig. 2c shows the result of the search tree using the RRT* algorithm. The best path length found is 103.96 units.

C. Dynamic Replanning Results

The simulation results shown below demonstrate the robot's ability to plan a path around the moving obstacle and reach the goal. Using RRT* during the replanning step allows an efficient path to be found to avoid the moving obstacle and continue on the original optimal path. Since the obstacles move randomly, it is possible for the robot to execute the optimal path and never be obstucted by a moving obstacle. Only examples where the robot did encounter these obstacles are shown.

The first set of results use a search tree containing 2000 nodes. Upon completion of the search tree the moving obstacles are placed randomly within the configuration space, and the simulation begins. Fig. 1a shows the search tree found by the robot upon reaching 2000 nodes.

When executing this path the robot encounters two random obstacles near the center of the configuration space. One obstacle moves across the path and obstructs the robot. The robot triggers a replanning event at this time. Fig. 1b shows when the robot encountered the moving obstacles and replanned to avoid them. A second obstacle is nearby and can be seen by the robot and must be considered when replanning. Following the replanning steps in Algorithm 8, the robot will select a goal location, then modify the search tree to avoid the obstruction.

Fig. 1c shows the full modified search tree. There is an obvious empty region of the configuration space where the moving obstacles are located. The abscence of branches within this area shows the robot has considered this space as obstacle space, rather than free space. Note, the nodes in this region are not removed, they are considered invalid. If the robot should need to replan again, and this area is free of any moving obstacles, these nodes would be available.

When the robot reaches the replanning goal location the original path can resume. Fig. 1d shows the completed path, along with the final positions of the random moving obstacles. The magenta line shows the path followed by the robot. The sections in blue are unexecuted portions of the original path.

The second set of results is similar to the first, with one exception. The random moving obstacle initial positions were selected in order to increase the probability the robot would encounter one, or many, while executing the path. Fig. 2a shows the starting locations of the moving obstacles. The robot was obstructed three times during the execution of the path and successfully planned around the moving obstacle each time. Fig. 2b shows the final positions of moving obstacles and the path followed by the robot.

The final set of results is a simulation with a search tree containing 5000 nodes and 3 obstacles moving at random in the configuration space. The three moving obstacles in this simulation were placed similarly to those in the second simulation. In this simulation the robot encounters a moving obstacle very early in the execution of the path. The robot finds a path around the moving obstacle and completes the path. Fig. 2d shows the final positions of the obstacles and the executed path.

V. Conclusion and Future Work

The replanning method presented performs well and is a good first step toward a more robust method of replanning when unknown randomly moving obstacles obstruct the robot's path. Future work will research a floating replan goal location to minimize the total remaining cost to the query goal. Minimizing the modifications to the original search tree is another area of improvement. This method has only been implemented in a 2-Dimensional configuration space. The algorithm must be expanded and modified to operate in higher dimension configuration spaces. The simulations up to this point have been with small numbers of moving obstacles, further research is needed to determine how the algorithm performs when there are many random moving obstacles. Additional research is pursuing multi-robot systems. Efficient path planning and replanning will benefit: cooperative sensing systems [15], [16], formation control systems [17]–[21] , and target tracking and observation [22].

References

[1] 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*, vol. 12, no. 4, pp. 566–580, Aug 1996.

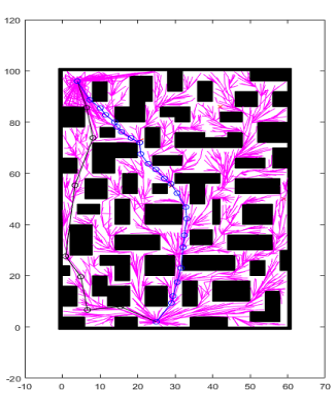
[2] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[3] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE Conference on Decision and Control (CDC)*, Dec 2010, pp. 7681–7687.

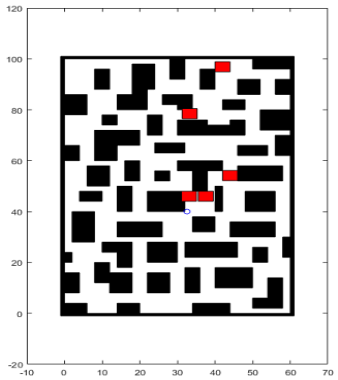
[4] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt*," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1478–1483.

[5] D. Ferguson and A. Stentz, "Anytime rrts," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2006, pp. 5369–5375.

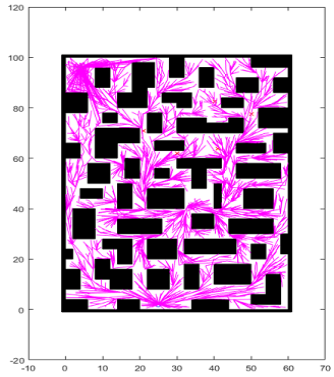
[6] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 2537–2542.



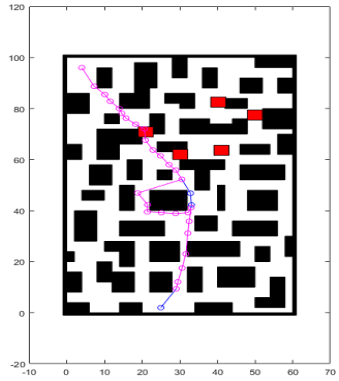
(a) Search tree and robot path.



(b) Moving obstacles block the path.

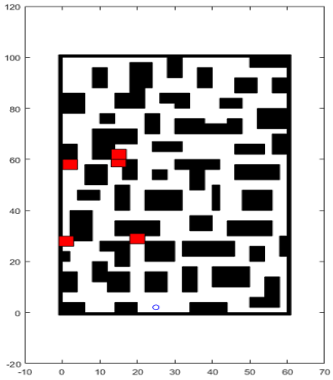


(c) Search tree after modification during replanning.

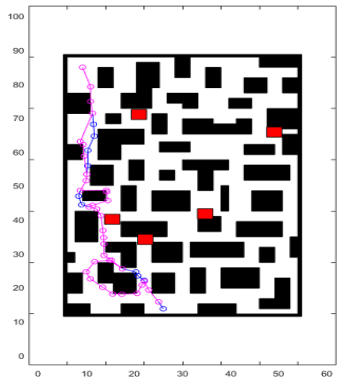


(d) Final obstacle positions and the executed path.

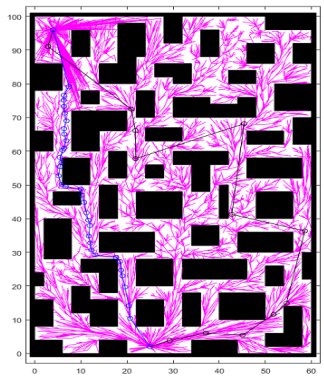
Fig. 1: Replanning results from the first set



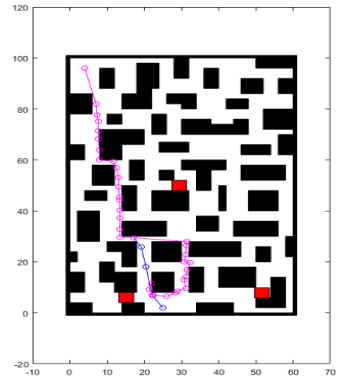
(a) Initial positions of the moving obstacles.



(b) Final obstacle positions and the executed path.



(c) RRT* with 5000 nodes.



(d) Final obstacle positions and the executed path.

Fig. 2: A sample RRT* search tree containing 5000 nodes and replanning results from the second and third sets.

[7] 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*, Sept 2014, pp. 2997–3004.

[8] O. Salzman and D. Halperin, "Asymptotically near-optimal rrt for fast, high-quality, motion planning," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 4680–4685.

[9] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "Rrt*-smart: Rapid convergence implementation of rrt*; towards optimal solution," in *2012 IEEE International Conference on Mechatronics and Automation*, Aug 2012, pp. 1651–1656.

[10] A. H. Qureshi, K. F. Iqbal, S. M. Qamar, F. Islam, Y. Ayaz, and N. Muhammad, "Potential guided directional-rrt* for accelerated motion planning in cluttered environments," in *2013 IEEE International Conference on Mechatronics and Automation*, Aug 2013, pp. 519–524.

[11] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2002, pp. 2383–2388 vol.3.

[12] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, May 2006, pp. 1243–1248.

[13] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite rrts for rapid replanning in dynamic environments," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, April 2007, pp. 1603–1609.

[14] M. Otte and E. Frazzoli, "Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 797–822, 2016.

[15] H. M. La, W. Sheng, and J. Chen, "Cooperative and active sensing in mobile sensor networks for scalar field mapping," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 1, pp. 1–12, Jan 2015.

[16] H. M. La and W. Sheng, "Distributed sensor fusion for scalar field mapping using mobile sensor networks," *IEEE Transactions on Cybernetics*, vol. 43, no. 2, pp. 766–778, April 2013.

[17] H. La, T. Nguyen, T. D. Le, and M. Jafari, "Formation control and obstacle avoidance of multiple rectangular agents with limited communication ranges," *IEEE Transactions on Control of Network Systems*, vol. PP, no. 99, pp. 1–1, 2016.

[18] H. M. La, R. S. Lim, W. Sheng, and J. Chen, "Cooperative flocking and learning in multi-robot systems for predator avoidance," in *2013 IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems*, May 2013, pp. 337–342.

[19] H. M. La and W. Sheng, "Multi-agent motion control in cluttered and noisy environments," *Journal of Communications*, vol. 8, no. 1, pp. 32–46, 2013.

[20] A. D. Dang, H. M. La, and J. Horn, "Distributed formation control for autonomous robots following desired shapes in noisy environment," in *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, Sept 2016, pp. 285–290.

[21] H. M. La and W. Sheng, "Flocking control of multiple agents in noisy environments," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 4964–4969.

[22] ——, "Dynamic targets tracking and observing in a mobile sensor network," *Elsevier Journal on Robotics and Autonomous Systems*, vol. 60, no. 7, pp. 996–1009, 2012.