Modified RRT* for Dynamic Path Planning

Nagarakshith Makam Sreenivasulu and Venkat Varun Velpula

Abstract

Sampling based path planning algorithms have become a very powerful motion planning framework for manipulator and mobile robotics to solve complex path planning problems. But, the drawback of standard sampling based algorithms is that they assume that the environment is static. In our work for the final project for ESE 650, we want to extend current sampling based algorithms like RRT* for dynamic environments. The replanning is done by taking the existing nodes in consideration or growing the tree and then reconnecting them, thus ensuring optimality and speed of replanning. We have compared and benchmarked the performance of this algorithm with the standard RRT* expansion based replan. We found that this method outperforms the other in both speed and optimality.

I. Introduction & Motivation

PATH planning for robots is a very important part of an autonomous robot software stack. As robots are becoming more ubiquitous in industries and in consumer electronics space, robots are being used in highly dynamic enironments. This led to a rise of importance of dynamic path planning.

Sampling based algorithms is a class of path planning algorithms which are probabilistic in nature. The advantages of sampling based algorithms is that they are computationally less expensive because explicit representation of the environment constraints is not required. For the purpose of collision checking, they only need to check for collision for a particular trajectory of the path. Also, another advantage of sampling based algorithms are that they are easily expandable to higher dimensions.

In our work we want to focus on the class of RRT based algorithms, from which RRT* is also derived. RRT aims to find the solution as soon as possible. Thus, there is a very high emphasis on exploration over exploitation (or optimization). But this methodology leads to paths which are highly sub-optimal. RRT* on the other hand is asymptotically optimal. In the algorithm we perform optimization through the process of rewiring nodes where we rewire parent node and child node relationships to optimize for global cost.

However, in the real world, planning once before robot motion and the assumption that the environment does not change is highly utopian. The environments in which robots perform their tasks is always constantly changing. A very straightforward approach is to clear all the nodes expanded and path computed and replan again from the current position with the new obstacles. But this has two main disadvantages. One, we would again need to spend a lot more compute resources to obtain an optimal path. Two, the time required for this replanning procedure would not be acceptable in fast paced environments like a road for an autonomous vehicle.

So we want to perform replan with the existing tree expanded with as minimal changes as required so as to reduce computation load on the robot processors and enable swift responses to dynamical obstacles. In our project we want to implement an algorithm which replans fast and also guarantees to a certain degree, an optimal path.

II. Related Work

Path planning using cell decomposition [1] divides the map into cells and determine free and occupied cells for traversal. Artificial potential fields [2] use attractive and repulsive forces for the goal position and obstacles respectively. Visibility graphs is another prior research done in non-sampling based path planning [3]. But using these techniques in higher dimension space and with large number of obstacles is challenging. This is because, they need an explicit representation of the obstacles.

Sampling based algorithms include PRM [4], RRT [5] and RRT* [6]. The D* [7] and D* Lite [8] algorithms are examples of dynamic are deterministic algorithms for dynamic environments. They also face the issue of high complexity at higher dimensions.

RRT based motion planning algorithms for dynamic environment include DRRT [9] where invalidated nodes are removed and and the tree is regrown till a path is found. A Dynamic grafting based RRT dynamic planner is introduced in [10], where grafts are used to connect between two points in the tree by avoiding obstacles. In [11], RRT is used to determine the path to the goal, and RRT tree is expanded after every step the robot takes. Dynamic path planning is achieved in [12] by performing RRT* tree expansion again to determine more trajectories to avoid obstacles. Another paper [13] based on RRT* deals with an obstacle which moves only once using similar concepts of regrow and reconnect which we have presented in our work.

III. Method and Implementation

A. Methodology

The method of dynamic probabilistic path planning expands over the implementation of standard RRT*, so that we can account for a moving obstacle which obstructs the computed path. The entire algorithm is presented in Algorithm 1. Initially a regular tree is grown with a suitable number of iterations considering the static obstacles using the RRT* algorithm. In our experiments, we have assumed that during the tree growth process, there are no moving obstacles present on the map. The optimal path to the goal is computed and the robot starts executing it. At every step of the robot along the path, the information of the dynamic obstacle position is updated.

Algorithm 1 RRT* Dynamic

$\tau = RRT^*(p_{init})$
$\sigma = findPath(p_{init}, p_{goal})$
$prev\_obstacle\_pose = obstacle\_pose, i = -1, recompute\_optimal\_path = False$
while $i \neq len(\sigma)$ **do**
  $i = i + 1$
  $p\_current = \sigma(i)$
  $obstacle\_pose = updateObstacle()$
  if $prev\_obstacle\_pose \neq obstacle\_pose$ **then**
    $\tau = pruneTillPcurrent(\tau)$
    $recompute\_optimal\_path = True$
  end if
  if $detectCollision(\sigma, obstacle\_pose)$ **then**
    $\sigma_{current}, \sigma_{separate}, obstructed\_nodes = detectOccupiedNodes()$
    $\tau, \sigma_{current}, \sigma_{separate} = modifyTreeAndPath(\sigma_{current}, \sigma_{separate}, obstructed\_nodes)$
    $\sigma_{new} = Reconnect(\sigma_{separate})$
    if $len(\sigma_{new}) \neq 0$ **then**
      $\sigma = \sigma_{new}$
      $i = 0$
    else
      $\tau = Regrow(\tau)$
      $\sigma_{new} = Reconnect(\sigma_{separate})$
      $i = 0$
    end if
    $recompute\_optimal\_path = False$
  end if
  if $recompute\_optimal\_flag$ **then**
    $\sigma = findPath(p\_current, p\_goal)$
    $i = 0$
    $recompute\_optimal\_flag = False$
  end if
end while

If the obstacle is found to block the path, the robot stops and tries to replan. The current position of the robot in terms of the index (node in the tree) of this position on the tree is called $p_{current}$. The function $detectOccupiedNodes()$ returns a list of obstructed nodes in the tree and the broken paths $\sigma_{current}$ and $\sigma_{separate}$. $\sigma_{current}$ is the path between the robot and the obstacle while $\sigma_{separate}$ is the path between obstacle and the goal. The first node in $\sigma_{separate}$ is called $p_{separate}$. The tree is split into two sections, one is the main tree and the other with the root as $p_{separate}$ is the orphan tree as seen in Figure 2(b) and in Figure 3(b).

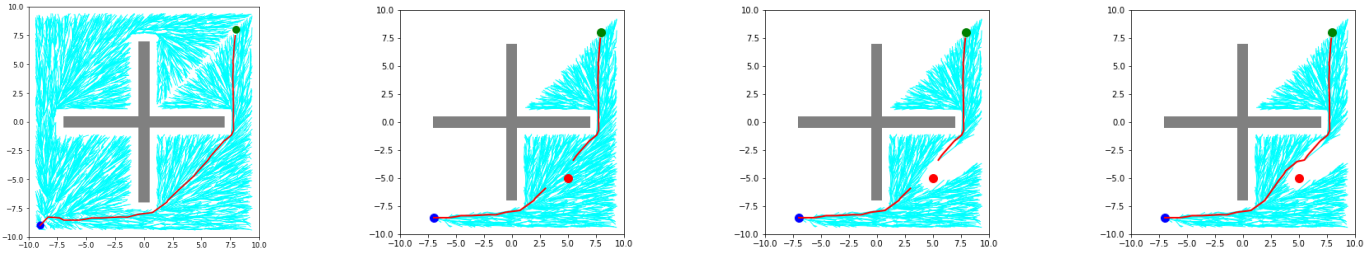
An assumption, we made is that if an obstacle is blocking the path anywhere on the map, the replanning routine is invoked. In the first step of the replanning routine, all the nodes in the previous path up to $p_{current}$ along with their children are deleted from the tree using the function $prune\_till\_pcurr()$. This is done because it is not possible to reach $p_{current}$ from those nodes. Then, all the nodes and their children obstructed by the obstacles are invalidated temporarily in the tree. The robot tries to reconnect $\sigma_{separate}$ and $p_{current}$ through other nodes of the main tree. For every node on the $\sigma_{separate}$ a suitable radius is considered and all the nodes of the main tree in that radius are obtained. For every node of the main tree found, the sub routine $findPath()$ returns the path and the cost associated with that path if present. If many suitable paths are obtained, the one with the minimum cost is chosen for guaranteeing optimality. The robot now starts executing the new path obtained. The reconnect algorithm is presented in Algorithm 2. The reconnect process is illustrated in Figure 1.

Algorithm 2 Reconnect

for all $nodes$ in $\sigma_{separate}$ **do**
  $distances = getDistances(\tau, nodes)$
  $potential\_parents = distances < radius$
  $min\_cost = Inf$
  for all $potential\_parent\_nodes$ in $potential\_parents$ **do**
    $path, cost = findPath(p_{current}, potential\_parent\_nodes, \sigma_{separate})$
    if $cost < min\_cost$ **then**
      $best\_path = path$
      $min\_cost = cost$
    end if
  end for
end for
return $best\_path$

Algorithm 3 Regrow

while True **do**
  $x = randomly\ sampled\ point$
  $node_{nearest} = Nearest(x, p_{separate})$
  $x_{new}, steer_{cost} = Steer(x, node_{nearest})$
  $check = is\_in\_collision(x_{new}, node_{nearest})$
  if check **then**
    continue
  else
    break
  end if
end while
$nodes_{near} = GetNearList(x_{new})$
$Connect(x_{new}, node_{nearest}, steer_{cost})$
if $indices_{nearest}$ is not empty **then**
  $ModifyNearIndices(indices_{near}, p_{separate})$
  $node_{best}, cost = NearestFromList(x_{new}, nodes_{near})$
  if $node_{best}$ **then**
    $steer_{cost} = getDistance(node_{new}, node_{best})$
    $Connect(x_{new}, node_{best}, steer_{cost})$
  end if
  $Rewire(node_{new}, nodes_{near})$
end if

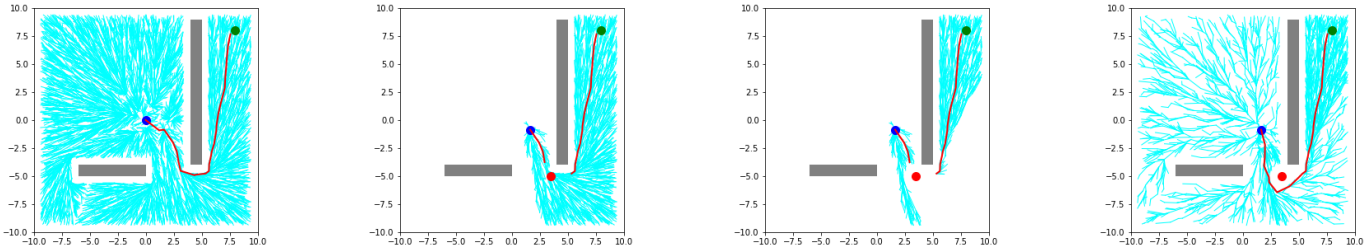


(a) Original Path (b) Obstacle Blocking the Path (c) Nodes Temporarily Blocked (d) Reconnected Path

Fig. 1: Reconnect Routine

Sometimes if the tree is pruned to a large extent after detecting the obstacle, reconnect may fail due to the absence of the nodes in the main tree. Hence the tree has to be regrown. The function $Regrow()$ is analogous to the tree building process as in RRT* but here only the main tree is grown. After regrowing the tree for desired number of iterations, the function $Reconnect()$ is called which connects the $\sigma_{separate}$ and $p_{current}$ with the newly generated nodes of the main tree. The regrow

algorithm is presented in Algorithm 3. The reconnect is illustrated in Figure 2.



(a) Original Path

(b) Obstacle Blocking the Path

(c) Nodes Temporarily Blocked

(d) Path Obtained after Regrow

Fig. 2: Regrow Routine

So, in addition to previous work done in this algorithm, we have in addition also accounted for the motion of the obstacle with respect to the motion of the robot as time passes, till the robot reaches the goal. If the obstacle changes its position from the previous step, then there is a possibility of finding a more optimal path than the one being executed currently even if the current path is not obstructed. This is because the nodes of the parent tree obstructed by the obstacle previously are now active and these nodes can contribute to a lower cost path. But before finding the new path, we prune the tree till $p_{current}$ node so that the children of the traversed path nodes are deleted. Now, an optimal path is found from the goal to the current position of the robot. If the obstacle moved in the map in such a manner that the path is obstructed, then the general Reconnect or Regrow routines are performed and the previously mentioned steps are not executed.

B. Algorithm Implementation Details

The code is written in Python. The entire tree is stored as a Numpy array with six columns and sufficient number of rows where the rows correspond to the nodes of the tree. The reason we opted to use an Numpy array in place of linked list is due to the computation advantages of Numpy array as operations on them can be vectorized. The first two columns store the coordinates of the tree node, the third column stores the cost between the current node and its parent, the fourth column stores the unique ID of its parent, the fifth column is a flag variable which is 1 if the node is in the tree, 0 if it is temporarily invalidated and 2 if the node has to be deleted immediately. Finally the sixth column stores the unique ID of every node. This is done to keep track of the hierarchy among the nodes on deletion of the nodes in the pruning step.

The function $pruneTillPcurrent()$ marks the fifth column of all the previous nodes on the travelled path and their children with 2. To determine all the children of a given node in the tree the Breadth First Search (BFS) algorithm is implemented. We have also tried implementing the Depth First Algorithm (DFS) algorithm. But it cannot be implemented with this data structure as it is crossing the limit of 1000 recursion calls allowed in Python.

The actual deletion of the nodes is done in the function $modifyTreeAndPath()$ where all nodes marked 2 along the fifth column are deleted. Also the fifth column of obstructed nodes is marked with 0 for invalidation. After deleting the nodes in the tree, the parent indices stored in the fifth column of the nodes in the array are no longer linked to the correct indices as nodes have been deleted. To tackle this problem the idea of unique ID is introduced. Every node in the tree stores the unique ID of its parent instead of the array indices. Initially before any deletion, the array indices and the unique ID of the nodes are same. After deletion, the unique ID's of parent of every node is updated according to the new array indices and once again the unique ID is made equal to the array indices. The node IDs in the path are also updated according to their position in the new pruned tree.

IV. Results

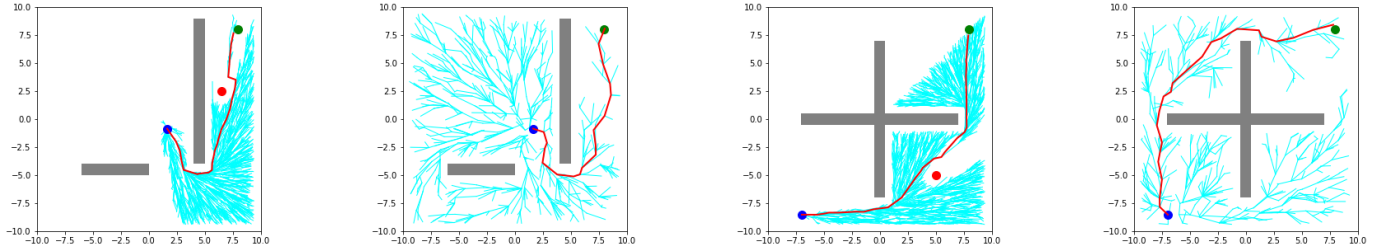
The dynamic path planning algorithm implemented is successful for all the trials we have tested it for on the maps tested for. When an obstacle obstructs the path, first the reconnect routine is run where we try to reconnect the two separated path segments using the nodes already present in the tree. If this step fails to return a reconnected path, then the regrow routine is run. Here the tree is expanded for N number of iteration and then the path is tried to be reconnected again.

Our method of Dynamic-RRT* is better than simply replanning from scratch using RRT*. In Figure 3, we have compared these two. The results of the runtime for both the methods on both the maps are as follows:

Map	Reconnect	RRT* (until a path is found)	RRT* (for 6000 iterations)
Map 1	21.74s	14.13s	224.49s
Map 2	36.59s	19.38s	239.19s

TABLE I: Runtime for replanning (done on laptop with i7 dual core processor with 12 GB RAM)

The RRT* does obtain a goal faster as seen in results from I, but as seen in Figure 3, the paths obtained are highly sub-optimal. The paths obtained through the reconnect routine ensure the asymptotic optimailty of RRT* tree expanded earlier. To re-obtain this level of optimailty, the RRT* needs to be run again for large number of iterations which increases the replanning time.



(a) Reconnection done on Map 1

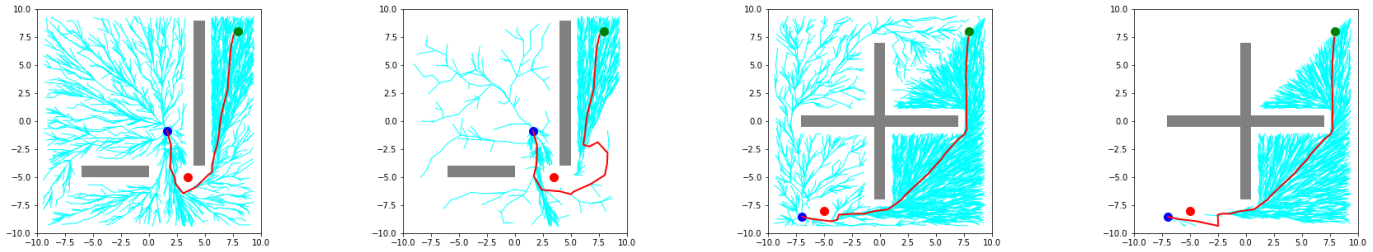
(b) RRT* run after deleting old tree until a suitable path is found

(c) Reconnection done on Map 2

(d) RRT* run after deleting old tree until a suitable path is found

Fig. 3: Comparison between Reconnect and RRT* (run until any path to goal is found)

We have conducted the analysis of performance of the Regrow subroutine. We have tested two ways in performing this routine. One method is to regrow the tree for N number of iterations and then check for a reconnection. The other method is to check for the possibility of reconnection after every time the tree is regrown by one node. When we ran the routine for about 1500, we obtained better paths, but sacrificed some speed on replanning. In the other case, we found the path very quickly but the path is almost never good (optimality-wise). So, there is trade-off between speed of replanning and optimaility of replanned path. During the process of replanning, the robot needs to take quick decisions to avoid obstacles. So, the number of iterations required is highly dependent on the circumstances. Hence, in our experiments we decided to set the number of iterations to about 1000 to 1500 nodes as this ensures both the speed of response and some degree of optimaility.



(a) Path Obtained from Regrow Routine which is run for 1500 iterations

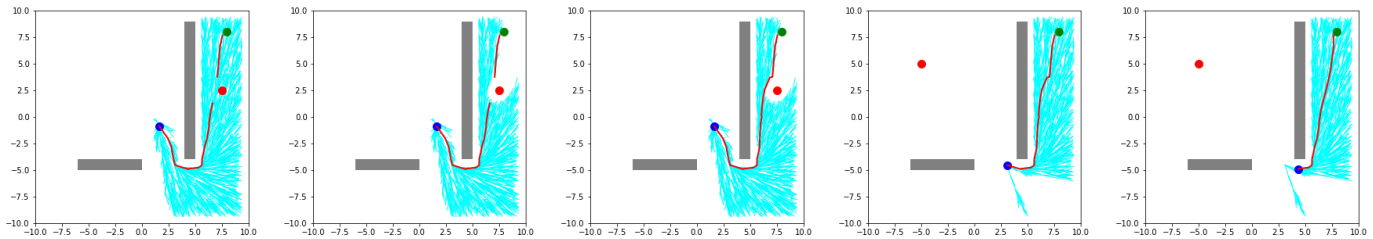
(b) Path Obtained from Regrow Routine, run until reconnection is possible

(c) Path Obtained from Regrow Routine which is run for 1500 iterations

(d) Path Obtained from Regrow Routine, run until reconnection is possible

Fig. 4: Comparison of two implementation methods of Regrow

As explained in Section III our implementation, we have tried to invoke concept of receding horizon control like concept for path planning. So, whenever the obstacle moves in the map, and frees any nodes, a check for best path is done. This is because, sometimes, previously occupied nodes can give rise to a better path as shown in Figure 5.



(a) Location of first obstacle

(b) Blocking nodes and their children obstructed by obstacle

(c) Reconnected path

(d) Movement of obstacle to (-5,5)

(e) Re-determination of the best optimal path by re-activating those nodes

Fig. 5: Figures Illustrating the Dynamic Nature of Replanning

V. Future Work

In this work the results are shown for a single moving obstacle. With a few modifications to the code it can be scaled to multiple moving obstacles. But some of the challenges might include as to how to deal when more than one obstacle is blocking the path at different points simultaneously. Few metrics have to be defined to split the path effectively. One approach might involve to replan the path around the closest obstacle and hope for other obstacles to move as the time progresses. Other might be to invalidate all the obstructed nodes on the path and also the nodes on the path between the obstacles if they are close enough. Another possible option is to plan for the obstacles sequentially, one at a time.

References

[1] Tomás Lozano-Pérez. "Spatial Planning: A Configuration Space Approach". In: *IEEE Transactions on Computers* C-32.2 (1983), pp. 108–120. ISSN: 00189340. DOI: 10.1109/TC.1983.1676196.

[2] Oussama Khatib. "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots". In: *Autonomous Robot Vehicles*. Springer New York, 1986, pp. 396–404. DOI: 10.1007/978-1-4613-8997-2_29.

[3] Tomás Lozano-Pérez and Michael A Wesley. "An algorithm for planning collision-free paths among polyhedral obstacles". In: *Communications of the ACM* 22.10 (1979), pp. 560–570.

[4] Lydia E. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580. ISSN: 1042296X. DOI: 10.1109/70.508439.

[5] Steven M. Lavalle and Steven M. Lavalle. "Rapidly-Exploring Random Trees: A New Tool for Path Planning". In: (1998). URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.1853.

[6] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research* 30.7 (June 2011), pp. 846–894. ISSN: 0278-3649. DOI: 10.1177/0278364911406761. URL: http://journals.sagepub.com/doi/10.1177/0278364911406761.

[7] Anthony Stentz. *Optimal and efficient path planning for unknown and dynamic environments*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1993.

[8] Sven Koenig and Maxim Likhachev. "Fast replanning for navigation in unknown terrain". In: *IEEE Transactions on Robotics* 21.3 (June 2005), pp. 354–363. ISSN: 15523098. DOI: 10.1109/TRO.2004.838026.

[9] Dave Ferguson, Nidhi Kalra, and Anthony Stentz. "Replanning with RRTs". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2006. 2006, pp. 1243–1248. ISBN: 0780395069. DOI: 10.1109/ROBOT.2006.1641879.

[10] Enzhong Shan et al. "An dynamic RRT path planning algorithm based on B-spline". In: *ISCID 2009 - 2009 International Symposium on Computational Intelligence and Design*. Vol. 2. 2009, pp. 25–29. ISBN: 9780769538655. DOI: 10.1109/ISCID.2009.155.

[11] Mikael Svenstrup, Thomas Bak, and Hans Jørgen Andersen. "Trajectory planning for robots in dynamic human environments". In: *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*. 2010, pp. 4293–4298. ISBN: 9781424466757. DOI: 10.1109/IROS.2010.5651531.

[12] Devin Connell and Hung Manh La. "Dynamic path planning and replanning for mobile robots using RRT". In: *2017 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2017*. Vol. 2017-January. Institute of Electrical and Electronics Engineers Inc., Nov. 2017, pp. 1429–1434. ISBN: 9781538616451. DOI: 10.1109/SMC.2017.8122814. arXiv: 1704.04585.

[13] O. Adiyatov and H. A. Varol. "A novel RRT*-based algorithm for motion planning in Dynamic environments". In: *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*. 2017, pp. 1416–1421.