Augmenting RRT*-Planner with Local Trees for Motion Planning in Complex Dynamic Environments

Ahmed Hussain Qureshi, Saba Mumtaz, Wajeeha Khan, Abdul Ahad Ashfaq Sheikh, Khawaja Fahad Iqbal, Yasar Ayaz, and Osman Hasan

***Abstract*— Collision free navigation in dynamic environments, where motion of moving obstacles is unknown, still presents a significant challenge. Sampling based algorithms are well known for their simplicity and are widely used in many real time motion planning problems. While many sampling based algorithms for dynamic environments exist, assumptions taken by these algorithms such as known trajectories of moving obstacles, make them unsuitable for motion planning in real-world problems. In this paper, we present RRT* based motion planning in unknown dynamic environments. Effectiveness of our idea is demonstrated in multiple simulations with more than 15 simultaneously moving obstacles placed in various environments.**

Index Terms - Motion Planning, Random Sampling, Dynamic Environment, RRT*.

I. Introduction

Efficient robotic motion planning in dynamic environments is a basic requirement of today's physical world. The Motion Planning problem is to find a set of control inputs that consider dynamics of a given robot and utilise them to drive the robot from one point to another while avoiding collisions with any obstacles, static or dynamic, occurring in its environment. Two classes of algorithms address this problem; one is complete, i.e., it succeeds in returning a solution, if one exists, in finite execution time and reports failure if the solution does not exist. While the other does not assure full completeness but does ensure probabilistic completeness or resolution completeness. Many complete motion planning algorithms exist but are computationally inefficient [1] for most practical applications[2]. Algorithms imparting resolution completeness include artificial potential fields (APF) [3]. However, APF based algorithms are effective only if the resolution parameter of the grid is finely tuned. Moreover, they also suffer from the problem of local minima [4].

To overcome the above mentioned problem of inefficiency, computationally efficient sampling based algorithms [5] were proposed. Arguably, the most effective of these algorithms are Probabilistic Road Maps (PRM) [6] and Rapidly exploring Random Trees [7], which ensure probabilistic completeness, i.e., as the number of iterations increases to infinity, the probability of finding path trajectory, if one exists, approaches one. Many successful attempts have been made to use PRM [8] and RRTs [9] [10] for motion planning in dynamic environments. The PRM algorithm is known to perform well in high dimensional space but PRM based motion planning proposed in [8] requires prior information of the trajectory of moving obstacles, which is usually unknown in real-world motion planning problems. PRMs therefore, tend to be unfeasible when the environment is unknown and computing a road map during run time is computationally expensive. Moreover, most online motion planning problems can be solved as single-query problems instead [2]. Also, RRTs algorithms are not only more efficient in terms of query time complexity as compared to PRM based algorithms, but the improved variant of RRTs, called Rapidly Exploring Random Tree Star (RRT*) [2], also ensures asymptotic optimality and memory efficient motion planning. RRT*, just like RRTs, finds the initial path very quickly and then iteratively provides an optimal/near optimal solution irrespective of obstacles' geometry. Since Karaman and Frazzoli [2] have proven that RRT* is more memory efficient than RRTs and RRT*'s running time is a constant multiple of RRT's running time, it can be logically concluded that RRT* will be the most suitable sampling based algorithm for dynamic environments as well.

This paper therefore introduces RRT* based motion planning in complex unknown dynamic environments. Taking into account only the static obstacles, RRT* is first used to find the global generalized path from the starting to goal points. The robot follows this path until it encounters moving obstacles. We propose a local tree generation technique, called the Triangular Geometerised RRT* (TG-RRT*), which is then used for re-planning of RRT*'s trajectory to avoid any prospective collisions. It will be shown in this paper that TG-RRT* has a faster convergence rate then RRT* in smaller sections of the configuration space due to its biased sampling technique. To speed up the process of re-planning, this part of the algorithm therefore applies TG-RRT* instead of RRT* itself. In our proposed algorithm Hybrid-RRT* (H-RRT*), no prior information of moving obstacles is required, but once the algorithm starts to execute, it keeps track of current and previous positions of every moving obstacle. This assumption is reasonable since in real-world problems, obstacles tend to have some distinct

A. H. Qureshi, S. Mumtaz, W.Khan, A.A.A.Sheikh, K.F.Iqbal, Y.Ayaz, and O. Hasan are with the RISE lab, School of Mechanical And Manufacturing Engineering (SMME), National University of Sciences and Technology (NUST),Islamabad, Pakistan. (10beeaqureshi, 10beesmumtaz, 11beewkhan, ahad.ashfaq)@seecs.nust.edu.pk, (fahadiqbal, yasar)@smme.nust.edu.pk, osman.hasan@seecs.nust.edu.pk

A. H. Qureshi, S. Mumtaz, W.Khan, A.A.A.Sheikh and O.Hasan are also with the School of Electrical Engineering and Computer Sciences (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan.

feature that can be used to track them. Information about the location of obstacles helps to identify any obstacles on a collision path with the robot and to take remedial measures. This paper comprises of a total of 8 sections. Section III to Section VI explain the proposed Hybrid-RRT* (H-RRT*) algorithm while Section VII presents its experimental results. Section VIII concludes the paper.

II. The RRT* Algorithm

This section formally presents the RRT* algorithm. Let the given state space be represented as $X \in \mathbb{R}^d$ where d represents the dimension. The state space is further classified into collision free region i.e., $X_{\text{free}} \subset X$ and collision region $X_{\text{obs}} = X \backslash X_{\text{free}}$. RRT* path finding algorithm is used to guide a robot from an initial start point, $x_{\text{init}} \in X_{\text{free}}$, to a goal region, $X_{\text{goal}} \subset X_{\text{free}}$. Following are some of RRT*'s main processes and Algorithm 1 represents the pseudocode of RRT* algorithm.

Random Sampling: The procedure SampleState randomly samples the collision free space X_{free} and assigns the random sample to the random state variable x_{rand}. The tree T comprises of these samples which are uniformly and independently distributed in the obstacle free space.

Distance: Since the Euclidean space is considered in this paper, therefore the distance procedure computes the Euclidean distance, defined as cost, between two input random configurations.

Nearest Node: The procedure $NearestNode$ locates the configuration x_{nearest} in the random tree, T, nearest to the random sample x_{rand}.

Near Nodes: This function NearNodes returns the set of nodes situated within the ball region centered at random state x_{rand} of volume $\gamma \times (\frac{\log n}{n})^d$. Any given tree has n nodes and d dimensions while γ is a constant.

Extension: The procedure ExtendTo returns x_{new} in the configuration space. The state x_{new} is a node located at a small incremental distance δ from x_{nearest}, on the path from x_{nearest} to x_{rand} .

Collision checking: Let the path connecting any two states $x_1, x_2 \in X$ be denoted as $\sigma : [0, 1]$ such $\sigma[0] = x_1$ and $\sigma[1] = x_2$. The procedure Collisionfree returns true if $\sigma \subseteq X_{\text{free}}$ otherwise if the path σ crosses any obstacles space i.e., $\sigma \nsubseteq X_{\text{free}}$ then the procedure should returns false.

Insert Node: The procedure Insertion inserts the new node x_{new} to the tree as a child of nearest node which is now the parent node x_{parent}. This function also maintain the cost of path through x_{parent}, connecting x_{new} to x_{init}.

III. Spherical Region and The Temporal Goal

Spherical regions and Temporal Goals is a method that we designed to ensure safe and collision-free online motion planning. A Spherical region is defined around the main robot as shown in Figure 1. Since all moving obstacles are being continuously tracked, once a moving obstacle approaching the main robot is seen to enter this spherical region, it will be considered a threat to the robot. The robot would then immediately respond by altering its path and moving from behind the approaching obstacle by setting up a temporal goal as shown in the Figure 1. This particular strategy we referred to as the *go from behind* strategy. A temporal goal is the intersection point of the spherical region's boundary and the line joining the robot and a previous node of the approaching obstacle. Any number of approaching obstacles can therefore be avoided by the robot by redirecting itself towards the obstacle's previous position. The radius of the spherical region is defined as $r := (\frac{M}{A} * u)^{\frac{1}{N}}$ where $M \in \mathbb{R}$ and $A \in \mathbb{R}$ represent the mass and surface area, of the main robot, respectively. The variable $u \in \mathbb{R}$ denotes speed of the main robot and N is the number of moving obstacles in the given environment. The radius formula is devised based on the following assumptions 1)Geometry and the mass of moving obstacles is exactly the same as that of the main robot (the one with H-RRT*). This implies that the ratio of mass over area is a constant for all robots present in the configuration space; 2) higher the speed u of the main robot, larger will be the radius, ensuring that the obstacles present further away are also considered since higher speeds enhance the possibility of a collision; and 3) the exponent $\frac{1}{N}$ bounds the radius of spherical region within a certain range. It should be noted that the number of moving obstacles is unknown beforehand and is calculated on the basis of data received from an overhead camera. Moreover, trajectories followed by the moving obstacles are not known to the main robot. Sampling performed by the overhead camera at current time t leads to the determination of current positions of moving obstacles MO_t while the sampling done previously at $(t-1)$ leads to the determination of the previous position of the moving obstacles MO_{t-1}. The process of obtaining positions of moving obstacles from overhead cameras is, however, not included in the scope of this paper. For the purpose of this paper, we are assuming this information is already provided and accurate. The motion of the robot towards the temporal goal depends upon whether the temporal goal and the robots previous position are directly connectable. If the path towards

Algorithm 1: RRT*$(x_{\text{init}}, X_{\text{goal}})$

```
V ← x_init;
E ← φ;
for i ← 0 to N do
    x_rand ← SampleState(i);
    x_nearest ← NearestNode(T, x_rand);
    (x_new, u_new, T_new) ← ExtendTo(x_nearest, x_rand);
    if Collisionfree(x_new) then
        X_near ← NearNodes(T, x_new, |V|);
        x_parent ← BestParent(X_near, x_nearest, x_new);
        T ← InsertNode(x_parent, x_new, T);
        T ← Rewiring(T, x_parent, X_near, x_new);
    i ← i + 1;
return T = (E, V);
```

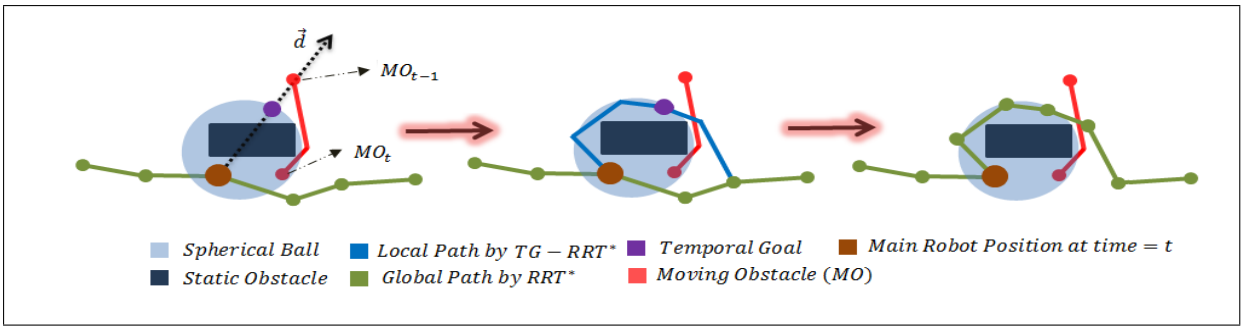


Fig. 1: Local Path through Temporal Goal

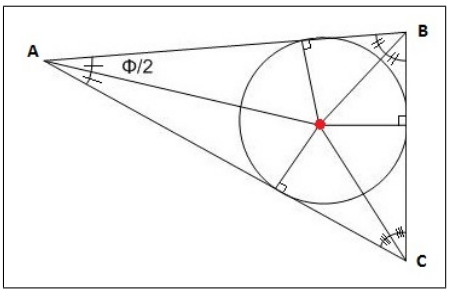


Fig. 2: Incentre

the temporal goal is clear of any static obstacles, the robot moves towards it directly. If, however, static obstacles lie in the path of the robot and the temporal goal, the part of the algorithm dealing with the generation of local trees comes into play.

IV. Local Trees

Local trees are used to provide collision-free local paths around static obstacles from a given node to a temporal goal and then back towards the next node on the initial path given by RRT*. We introduce TG-RRT*, a variation of the RRT* algorithm which utilizes triangular geometry to improve the performance of RRT* in small sections of the configuration space. The TG-RRT* algorithm modifies the original RRT* algorithm by proposing that every time the configuration space is randomly sampled, a further calculation be carried out to determine the Geometrical Incentre of the start point, goal point and the random sample. A Geometrical Incentre of three points forming a triangle is a point where all 3 angle bisectors intersect. Such a point is also the centre of the incircle, i.e., the largest possible circle that can be enclosed inside the confines of a triangle formed by the aforementioned three points. Figure 2 displays the location of a triangle's Geometrical Incentre. This calculated centre point is then assigned to a new variable called x_{nrand} which, from this point on, replaces the variable x_{rand} in the RRT* algorithm.

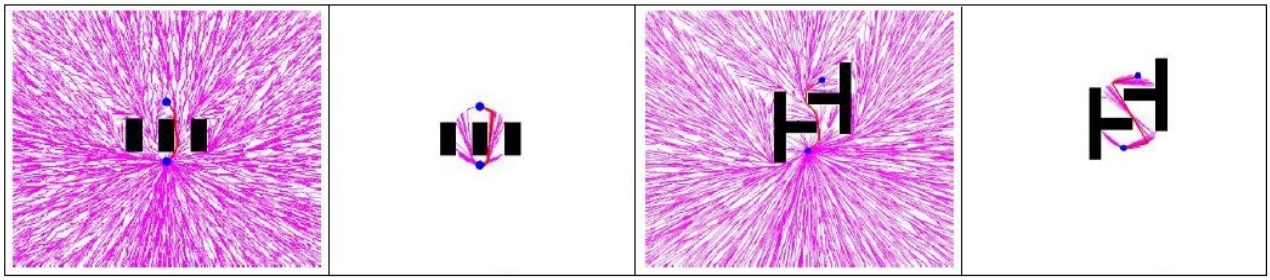
This variation on the RRT* algorithm presents an initial path towards the goal in significantly reduced number of iterations as compared to RRT* and consequently, requires much less amount of time. This is demonstrated in two sample environments presented in Figure 3. Table I provides further evidence of this improvement in performance by comparing the number of iterations and time required by RRT* and TG-RRT* to determine initial paths in 7 different types of environments.

Algorithm 2: H-RRT*$(x_{\text{init}}, X_{\text{goal}})$

$T = (V, E) \leftarrow \text{RRT}^*(x_{init})$;
$i \leftarrow 0$;
while $i \leq N$ **do**
  $P \leftarrow \text{GetPosition}()$;
  $R \leftarrow \text{ComputeSphericalBall}(P)$;
  $Y \leftarrow \text{NearestMovingObstacle}(P, \hat{P}, R)$;
  if Y **then**
    $x_{\text{new}} \leftarrow \text{GetTemporalGoal}(R, Y, \hat{P})$;
    if $\text{Obstaclefree}(x_{\text{new}})$ **then**
      $x_{\text{min}} \leftarrow \text{ChooseParent}(x_{\text{new}}, T)$;
      $T \leftarrow \text{InsertNode}(x_{\text{new}}, x_{\text{min}}, T)$;
      $T \leftarrow \text{Rewire}(x_{\text{new}}, x_{\text{min}}, T)$;
    else
      $\hat{T} = (V_{\text{new}}, E_{\text{new}}) \leftarrow \text{TG} - \text{RRT}^*(x_{\text{new}}, Y)$;
      $T \leftarrow \text{UpdateTree}(T, \hat{T})$;
      $x \leftarrow \text{NextNearestNode}(x_{\text{new}}, T)$;
      $\hat{T} = (V_{\text{new}}, E_{\text{new}}) \leftarrow TG - RRT^*(x_{\text{new}}, x)$;
      $T \leftarrow \text{UpdateTree}(T, \hat{T})$;
  $\hat{P} \leftarrow P$;
  $i \leftarrow i + 1$;
return;

V. Hybrid-RRT*

We call the overall algorithm Hybrid-RRT* (H-RRT*), comprising of both RRT* and Local Trees generated by TG-RRT*. Algorithm 2 represents the pseudo code of Hybrid-RRT* (H-RRT*). The given configuration space is represented as $X \in \mathbb{R}^d$, where d is the dimension. Region containing static obstacles is denoted by $X_{\text{obs}} \subset X$ while obstacle-free region is denoted by $X_{\text{free}} = X \backslash X_{\text{obs}}$. The trajectory returned by RRT* comprises of vertices $V \subset X_{\text{free}}$ and edges E. Both vertices and edges collectively form the tree T in the obstacle free configuration space i.e., $T \subseteq X_{\text{free}}$. Local trees generated by TG-RRT* are represented



(a) RRT* (b) TG-RRT* (c) RRT* (d) TG-RRT*

Fig. 3: Comparison of RRT* and TG-RRT* in small sections of configuration space

by $\hat{T}$ and contain edges E_{new} and vertices $V_{\text{new}} \in X_{\text{free}}$. $P \in X$ is the current Position of moving obstacles while $\hat{P}$ is their previous position. The main robot i.e., the one operating under the H-RRT* algorithm is represented by $\mathcal{A}$. A spherical ball region centered at $\mathcal{A}$ of radius r is represented as $\mathfrak{B}_{\mathcal{A},r} \in X$. The nearest moving obstacle approaching the main robot and also falling inside the spherical ball region is denoted by $Y \in P$. Following is a brief description of the set of procedures used in the second stage.

UpdateTree: It updates the tree T generated by RRT* and concatenates it with the local tree $\hat{T}$ generated by TG-RRT*, as shown in the Figures 4,(a) to (c). The procedure Rewire works similar to UpdateTree procedure, except it just redirects the path through the newly inserted temporal goal.

NextNearestNode: To aid the rewiring process, this function locates the node nearest to the temporal goal node on the trajectory initially returned by RRT*. This would be a node that has not yet been visited by the main robot.

NearestMovingObstacle: This process determines intersection of the spherical ball region $\mathfrak{B}_{\mathcal{A},r} \in X$ and moving obstacles P in the configuration space and assigns it to a new variable P_s. If P_s is empty, an indication of false is generated. Otherwise, this function iteratively checks the set of nearby moving obstacles and returns the moving obstacle which is nearest and approaching the main robot. Whether an obstacle is moving towards the main robot or away from it is determined using current position P and previous position $\hat{P}$ variables of the moving obstacles in $P_s \in P$.

TG-RRT:* The Triangular Geometrised RRT* (TG-RRT*) algorithm is used to generate local trees. The algorithm uses the set Y returned by the procedure *NearestMovingObstacle* to eliminate those edges of the generated local trees which intersect with the space $Y \in P$. This makes sure that the local path does not pass through the nearest moving obstacles.

Line 1 of Algorithm 2 corresponds to the first stage, i.e., RRT* returns the final optimal/near-optimal collision-free path solution considering only the static environment. Lines 9 to 12 are only executed if the path joining the main robot $\mathcal{A}$ and the temporal goal x_{new} lies in X_{free}, i.e, the path towards the temporal goal from the robot's current position $\mathcal{A}$ is unobstructed by any static obstacles. Otherwise, local trees

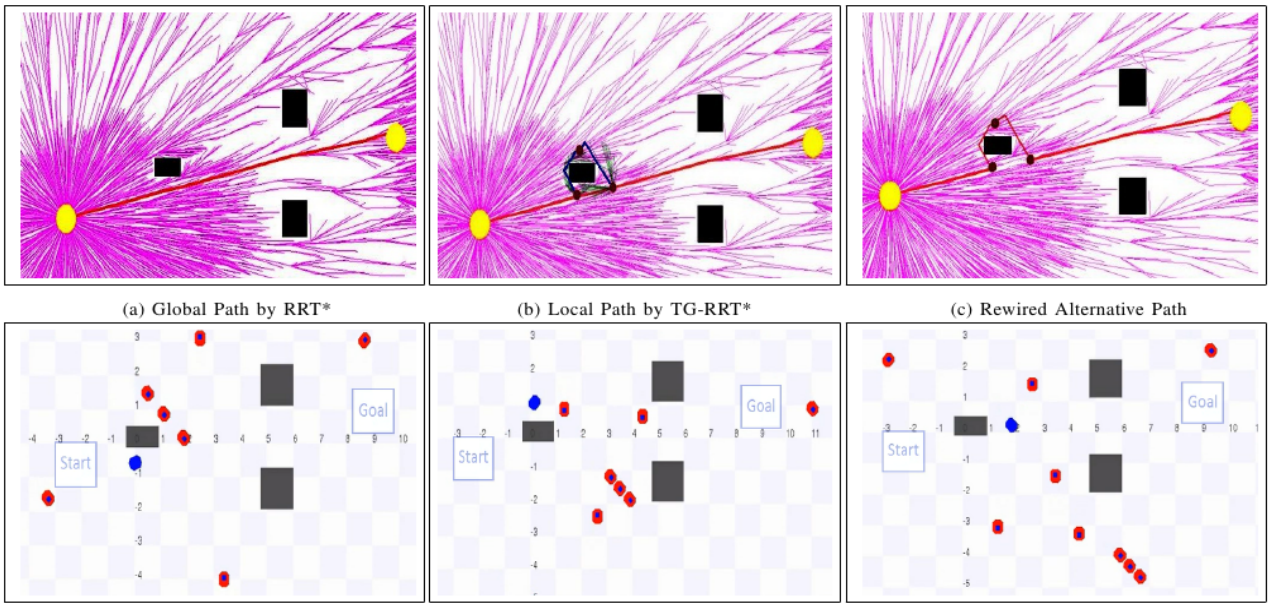
	RRT*		TG-RRT*	
Environments	Iterations	Time (sec)	Iterations	Time (sec)
A	074795	1.40	186	0.08
B	180314	3.31	072	0.00
C	058796	1.10	282	0.10
D	030472	0.56	106	0.00
E	116112	2.14	311	0.12
F	131504	2.42	128	0.061
G	097021	1.78	023	0.00

TABLE I: Iterations and Time Used By RRT* and TG-RR* for Finding Initial Path.

are used instead to devise a path to the temporal goal. In such a case, the global path given by RRT* is then concatenated with the local path towards the temporal goal, updating the tree T online and temporarily redirecting the robot. Lines 3 till 21 are repeated continuously until the desired goal position is achieved. It should be noted that if the temporal goal is directly connectable i.e., the path connecting $\mathcal{A}$ and x_{new} does not lie in X_{obs} then the procedure replan is executed. This procedure returns T that contains redirected path which directs the robot toward goal through the newly selected temporal goal. If the path connecting $\mathcal{A}$ and x_{new} does lie in X_{obs} the whole tree T is updated and local trees are inculcated into T by the procedure UpdateTree as presented in algorithm 2.

VI. Experimental Results

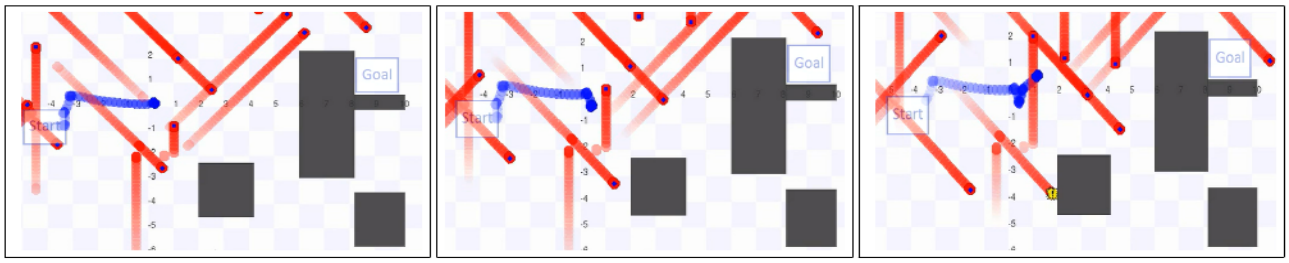
This section formally presents simulation results of our proposed algorithm that have been implemented in Player/Stage (a free software tool for robot and sensor applications, www.playerstage.sourceforge.net). All the results presented in this paper are simulated on 2.4 GHz Intel Corei5 processor with 4GB RAM. Speed of all mobile robots, including the mobile obstacles, was kept constant at 0.3 m/s. The Blue robot in the figures represents the main robot, i.e., the one using H-RRT* for motion planning. The red robots are moving obstacles while the black blocks represent static obstacles. For simplification purposes, collisions amongst the moving obstacles have been ignored. The pink and green edges in some figures denote the random trees, global and local, generated by RRT* and TG-RRT* respectively. Finally, the global path given by RRT* is shown as a red coloured trajectory while the local path by TG-RRT* is shown in navy blue. Complete simulation videos



(a) Global Path by RRT* (b) Local Path by TG-RRT* (c) Rewired Alternative Path

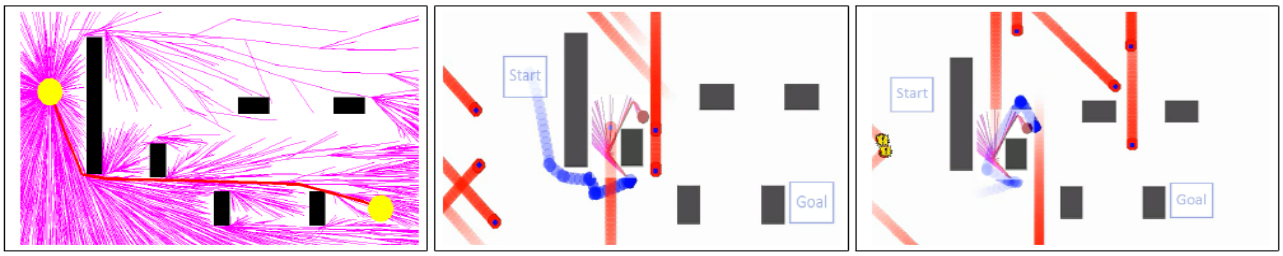
(d) E4: time=16s (e) E4: time=32s (f) E4: t=47s

Fig. 4: Path generated by RRT* and TG-RRT* to avoid expected collision



(a) E2: t=21s (b) E2: t=24s (c) E2 t=31s

Fig. 5: Robot Avoiding Real Time Collisions



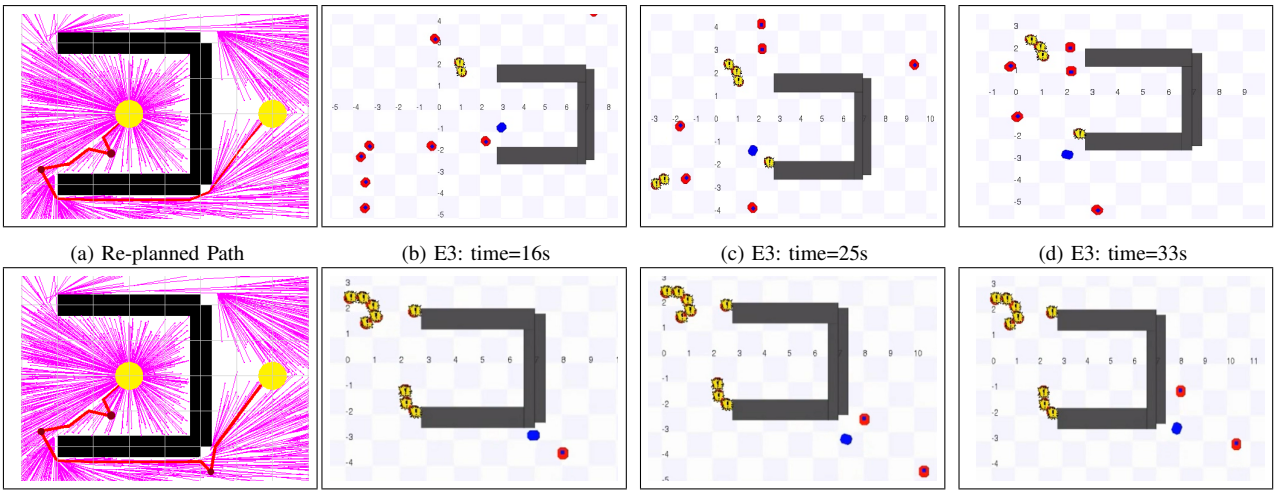
(a) Global Path By RRT* (b) E1: t=24s (c) E1: t=45s

Fig. 6: Robot Avoiding Real Time Collisions

are available at `http://www.4shared.com/folder/kg45QKC7/MMAR_2014.html`.

Figure 4a shows the optimal/near optimal trajectory obtained by RRT* in the given environment while Figure 4d shows the same environment implemented in Player/Stage. In Figure 4d, the robot detects the presence of moving obstacles in its Spherical Ball region. To avoid colliding with the three obstacles approaching the robot in a row, H-RRT* prepares to move from behind the incoming obstacles by generating a temporal goal. From Figure 4b, it can be seen that since the temporal goal could not be moved towards directly, the robot employs the use of TG-RRT* to generate a local tree around the static obstacle. Figure 4c shows the re-planned path generated by TG-RRT* which has been re-rewired with the original path given by RRT*. Furthermore, Figure 4e and Figure 4f clearly show the main robot following the new re-planned path.

Figure 5 represents the footprints of the robot at different times on the path generated by Hybrid-RRT* operating in the given dynamic environment. In Figure 5a, the robot



(a) Re-planned Path (b) E3: time=16s (c) E3: time=25s (d) E3: time=33s

(e) Re-planned Path (f) E3: time=46s (g) E3: time=49s (h) E3: time=54s

Fig. 7: H-RRT* in local minima environment

detects the presence of moving obstacles in its Spherical Ball region. To avoid collision, the robot follows the alternative path through the temporal goal. From Figure 5b, it can be seen that since the temporal goal was directly connectable, the robot moves directly towards it. There was no need to generate local trees. Figure 5c shows the robot returning to the global path initially given by RRT*.

Figure 6a shows the global path returned by RRT* among static obstacles in another given environment. Figure 6b, shows a situation where the robot found itself in danger and the temporal goal generated was not directly connectable. Therefore, in such a situation, local trees play their role and a local route is formed around the static obstacle using TG-RRT*. Figure 6b and Figure 6c, shows the local trees generated by TG-RRT* and the robot following this new alternative route to avoid prospective collision.

Figure 7 shows the local minima environment where each row represents a situation where the robot encountered dynamic obstacles, while performing motion planning in order to reach the goal point. As mentioned in the previous section, no local trees are generated if the two nodes are directly connectable, as shown in Figure 7a and Figure 7e. Since in this case, the temporal goal was directly connectable from the current robot position as well as from the next nearest node, the nodes were instead rewired directly. It should be noted that it is in fact the RRT* algorithm's proposed path which is being modified to avoid real time collisions.

VII. Conclusions

We have presented a simple sampling based approach for motion planning in complex dynamic environments. The proposed Hybrid-RRT* algorithm uses RRT* for finding a global trajectory from the start to the goal point. Working alongside is the TG-RRT* algorithm which generates local trees to redirect the RRT*'s path when in danger of colliding with moving obstacles. The results of the proposed algorithm were demonstrated using Player/Stage on the skid steering Poineer P3AT mobile robot. Moreover, in this paper it is assumed that all moving obstacles in the configuration space are of the same geometry as that of main robot (the one with H-RRT*) as our proposed Hybrid-RRT* is motivated from Robocup Standard Platform league where all robots present in the field are NAO humanoid robot i.e., all have same geometry and mass.

In future, we expect to perform hardware implementation of our proposed algorithm on the NAO humanoid robots. We also plan to analyze the H-RRT* in complex 3-D environments and on systems with higher degrees of freedom.

References

[1] J. Canny, *The complexity of robot motion planning*. The MIT press, 1988.

[2] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[3] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.

[4] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*. IEEE, 1991, pp. 1398–1404.

[5] F. Lamiraux and J.-P. Laumond, "On the expected complexity of random path planning," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 4. IEEE, 1996, pp. 3014–3019.

[6] L. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration for fast path planning," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE, 1994, pp. 2138–2145.

[7] S. M. LaValle and J. J. Kuffner Jr, "Rapidly-exploring random trees: Progress and prospects," 2000.

[8] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

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

[10] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 2210–2215.