1. INTRODUCTION

Different approaches have been proposed to realise planning in dynamic environments\(^1\). Sampling\(^2\) based motion planning algorithms have made real time planning of motion for unmanned vehicles possible\(^3\). A bias towards the goal\(^4\) is usually added by giving an increased probability to the goal states which tends to increase the rate of convergence of the planner. Shaping the probability distribution function\(^5\) which is used to generate random samples to grow a tree has shown improvements in the rate of convergence of the path planner\(^6\). The negative side of increasing the bias is that like randomised potential field it becomes too greedy also leading to local minima problem\(^7\). The goal directed approach presented in this paper is intended to provide an indirect goal bias without the probability based goal bias included in sampling algorithms. With the known advantage of increase in the rate of convergence, this approach tends to provide a solution with improved solution length. Study shows the improvement of goal directed approach over the probabilistically goal biased RRT algorithm for different unmanned vehicle platforms\(^8\).

2. RAPIDLY EXPLORING RANDOM TREES

Path planning involves defining a state space, \(X\) which constitutes both free region, \(X_{\text{free}}\) where the path can proceed and obstacle region, \(X_{\text{obs}}\) where the path cannot proceed through.

The state space enables representation of both configuration and velocity in a space. A collision detection algorithm enables to determine if a state, \(x\) causes collision in the modeled world. Thus selecting a number of collision-free states and computing a path from the start, \(x_{\text{init}}\) to the goal state, \(x_{\text{goal}}\).

The RRT algorithm\(^9\) considers dynamic constraints for path planning. Algorithm 1: BUILD RRT(\(x_{\text{init}}\))

\[
\text{a. } T.\text{init}(x_{\text{init}});
\]

\[
\text{b. for } k = 1 \text{ to } K \text{ do}
\]

\[
\text{c. } x_{\text{rand}} \leftarrow \text{RANDOM_STATE}();
\]

\[
\text{d. } \text{EXTEND}(T, x_{\text{rand}});
\]

\[
\text{e. } \text{Return } T
\]

Given an initial and goal state, a tree is constructed with the root of the tree in the initial state and branching till the goal state is reached. Adding nodes to the tree involves generation of random state \(x_{\text{rand}}\) within the state space using the \(\text{RANDOM_STATE}\) function. The selection of random nodes can be controlled by introducing a bias towards goal using probabilistic shaping in \(\text{RANDOM_STATE}\) function. Followed by the generation of a state \(x_{\text{rand}}\) extension of the tree is realised by the \(\text{EXTEND}\) function.

The node in the tree which is nearest to the random state is identified using \(\text{NEAREST_NEIGHBOR}\) function. The function uses a certain metric to compute the distances of each node of the tree with the random state. The node with the shortest distance to the random node, called nearest neighbour, \(x_{\text{near}}\) is
selected to connect the random node to the tree.

**Algorithm 2:** RRT EXTEND(T, x)

- **a.** \( x_{\text{near}} \leftarrow \text{NEAREST_NEIGHBOUR}(x, T) \)
- **b.** if NEW_STATE\((x, x_{\text{near}}, x_{\text{new}}, u_{\text{new}})\) then
- **c.** \( T.\text{add vertex}(x_{\text{new}}) \)
- **d.** \( T.\text{add edge}(x_{\text{near}}, x_{\text{new}}, u_{\text{new}}) \)
- **e.** if \( x_{\text{new}} = x \) then
- **f.** Return Reached;
- **g.** else
- **h.** Return Advanced;
- **i.** Return Trapped;

**NEW STATE** function is to make a motion towards \( x_{\text{rand}} \) from \( x_{\text{near}} \) using an input \( u_{\text{new}} \) or an incremental distance, \( \varepsilon \). The function checks if the random state, \( x_{\text{rand}} \), and also the intermediate states from \( x_{\text{near}} \) are reachable. If the random sample state \( x_{\text{rand}} \) was reached then the function returns Reached. If the tree only progresses towards \( x_{\text{rand}} \) then it returns Advanced and if the motion is trapped by an obstacle it returns Trapped. The new state is represented as \( x_{\text{new}} \).

3. **GOAL DIRECTED RRT PATH PLANNER**

A different variations\(^7\) has been proposed in this paper to the basic RRT algorithm. The current existing variations to the algorithm are classified under three categories as methods for generating random state, methods to extend towards a new node and methods to add bias to the goal. Every type of variation has its own advantages. The goal directed RRT planner introduces a new method of adding bias towards the goal primarily by using a \( \text{NEAREST}_\text{TO}_\text{GOAL} \) function resulting in faster rate of convergence of the planner. And for certain reasons the experiments also show a reduction in solution length.

**Algorithm 3:** BUILD DIRECTED_RRT\((x_{\text{init}})\)

- **a.** \( T.\text{init}(x_{\text{init}}) \)
- **b.** for \( k = 1 \) to \( K \) do
- **c.** \( x_{\text{rand}1} \leftarrow \text{RANDOM_STATE}() \)
- **d.** \( x_{\text{rand2}} \leftarrow \text{RANDOM_STATE}() \)
- **e.** \( x_{\text{rand}} \leftarrow \text{NEAREST}_\text{TO}_\text{GOAL}(x_{\text{rand}1}, x_{\text{rand}2}) \)
- **f.** EXTEND\((T, x_{\text{rand}})\)
- **g.** Return \( T \)

With the knowledge of initial and goal state, the planner starts by generating two random states \( x_{\text{rand}1} \) and \( x_{\text{rand}2} \) instead of one random state generated by the basic RRT planner in the free region using the RANDOM_STATE function. The difference in the RANDOM_STATE function introduced in GRRT is that it does not include probabilistic shaping towards goal as in RRT and hence avoiding the local minima problem.

The GRRT achieves the search towards the goal by the introduction of \( \text{NEAREST}_\text{TO}_\text{GOAL} \) function which selects the random state closer to the goal out of the two random states, \( x_{\text{rand}1} \) and \( x_{\text{rand}2} \). The random state out of \( x_{\text{rand}1} \) and \( x_{\text{rand}2} \) which is closest to the goal becomes the chosen random state \( x_{\text{rand}} \) for further growth of the tree.

The number of random states to choose from was set to two after simulation results showed best performance when the number of random states was limited to two. So, GRRT algorithm is proposed to include a preferential selection out of just two random states.

The **EXTEND** function in GRRT carries out the same functionality as in RRT by extending an edge from the tree towards the chosen \( x_{\text{rand}} \). The NEAREST_NEIGHBOUR function then computes the neighbour in the tree \( x_{\text{near}} \) closest to the chosen random state, \( x_{\text{rand}} \). Then the tree extends towards the random state. If the tree reaches \( x_{\text{rand}} \), the function returns Reached or if it advances but does not reach \( x_{\text{rand}} \) it returns Advanced and if the motion gets trapped by an obstacle, it returns Trapped.

The RRT tree structure exhibits the advantage of exploration of the entire search space. The tree structure of Goal directed RRT has an inclination towards the goal. This initiates a faster rate of convergence of the Goal directed RRT algorithm.

A possible variation to the proposed approach is if \( x_{\text{rand}1} \) or \( x_{\text{rand}2} \) lies in \( X_{\text{near}} \), whichever lies in \( X_{\text{free}} \) can be selected within the \( \text{NEAREST}_\text{TO}_\text{GOAL} \) function.

4. **EXPERIMENTAL SETUP**

The experiments were carried out on Intel Core i3 CPU with 2.53 GHz processor with 4GB RAM. The OMPL\(^{13}\) motion planning library was used for benchmarking the proposed algorithm. OMPL has provision for carrying out benchmarking tests by including new motion planning algorithms. The GRRT was included in OMPL for benchmarking the algorithm. OMPL includes different environment wherein the benchmarking can be carried out. Scripts available with OMPL to visualise the benchmarking results can be used for data visualisation.

On an average 25 runs were made for each planner in every scenario with a minimum planning time of 10 seconds to a maximum of 100 seconds. Whenever goal bias i.e., adding goal state to the distribution with a higher probability was considered, it was set to 0.05.

**Algorithm 4:** DIRECTED_RRT EXTEND\((T, x)\)

- **a.** \( x_{\text{near}} \leftarrow \text{NEAREST}_\text{NEIGHBOUR}(x, T) \)
- **b.** if NEW_STATE\((x, x_{\text{near}}, x_{\text{new}}, u_{\text{new}})\) then
- **c.** \( T.\text{add vertex}(x_{\text{new}}) \)
- **d.** \( T.\text{add edge}(x_{\text{near}}, x_{\text{new}}, u_{\text{new}}) \)
- **e.** if \( x_{\text{new}} = x \) then
- **f.** Return Reached
- **g.** else
- **h.** Return Advanced
- **i.** Return Trapped

4.1 **Kinematic Car Planning**

The configuration of a car can be expressed as,

\[
q = (x, y, \theta) \in R \times S^1
\]  
(1)

where the position is,

\[
(x, y) \in R^2
\]  
(2)

and the orientation is,

\[
\theta \in S^1
\]  
(3)

The definition of the benchmarking parameters is as follows:

**Graph Motion:** The number of edges in the constructed graph.

**Solution Length:** The length of the found solution.

**Solution Segments:** The number of segments on the solution path.

**Time:** The amount of time spent planning, in seconds.
**Solution Difference**: If the solution is approximate, this refers to the distance of goal from the end-point of the found approximate solution.

**Solved**: Indicating whether the planner found a solution (even approximate solution is considered)

The benchmarking was carried out on maze environment as shown in Fig. 1. As seen from Table 1 and Table 2, GRRT shows a decrease in graph motion, which represents a reduced graph tree data structure resulting in a faster search. Any graph search algorithm would be benefited by the reduced graph data structure.

![Image of maze environment](image)

Figure 1. Maze environment for kinematic car planning.

<table>
<thead>
<tr>
<th>Performance measure</th>
<th>RRT</th>
<th>GRRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph motions</td>
<td>42057.32</td>
<td>32661.72</td>
</tr>
<tr>
<td>Solution length</td>
<td>24.278</td>
<td>24.410</td>
</tr>
<tr>
<td>Solution segments</td>
<td>18.16</td>
<td>16.32</td>
</tr>
<tr>
<td>Time</td>
<td>7.93</td>
<td>6.85</td>
</tr>
<tr>
<td>Solved</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 1. Kinematic car benchmarking results

<table>
<thead>
<tr>
<th>Performance measure</th>
<th>Improvement - GRRT (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph motions</td>
<td>decrease of 22.34</td>
</tr>
<tr>
<td>Solution segments</td>
<td>decrease of 10.13</td>
</tr>
<tr>
<td>Time</td>
<td>decrease of 13.62</td>
</tr>
<tr>
<td>Solved</td>
<td>increase of 50</td>
</tr>
</tbody>
</table>

Table 2. Percentage change for kinematic car

4.2 Dynamic Car Planning

Dynamic car planning involves considering the acceleration \( \ddot{q} \) in addition to velocity \( \dot{q} \) and configuration \( q \).

For dynamic car planning, the state is,

\[
s = (x, y, \theta, s, \varphi) \tag{4}
\]

where the position,

\[(x, y) \in R^2 \tag{5}\]

the orientation,

\[\theta \in S^1 \tag{6}\]

the translational velocity,

\[s \in R \tag{7}\]

the steering angle,

\[\varphi \in R \tag{8}\]

The benchmarking of algorithms for a dynamic car was performed in ‘Random Polygon’ environment shown in Fig. 2, with a maximum runtime of 10 s for 25 runs for each planner. Similar to kinematic car benchmarking, different benchmarking parameters like solution length, time, solution segments, graph motions were considered for benchmarking. The results for a dynamic car are shown in Table 3 and the percentage change in Table 4.

![Image of random polygon environment](image)

Figure 2. Random polygon environment for dynamic car planning.

<table>
<thead>
<tr>
<th>Performance measure</th>
<th>RRT</th>
<th>GRRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph motions</td>
<td>17841.4</td>
<td>17591</td>
</tr>
<tr>
<td>Solution length</td>
<td>83.915</td>
<td>76.8071</td>
</tr>
<tr>
<td>Solution segments</td>
<td>56.4</td>
<td>50.24</td>
</tr>
<tr>
<td>Solution difference</td>
<td>0.96536</td>
<td>0.770209</td>
</tr>
<tr>
<td>Time</td>
<td>9.86</td>
<td>9.33</td>
</tr>
<tr>
<td>Solved</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3. Dynamic car benchmarking results

<table>
<thead>
<tr>
<th>Performance measure</th>
<th>Improvement - GRRT (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph motions</td>
<td>decrease of 8.47</td>
</tr>
<tr>
<td>Solution length</td>
<td>decrease of 10.92</td>
</tr>
<tr>
<td>Solution segments</td>
<td>decrease 20.21</td>
</tr>
<tr>
<td>Time</td>
<td>decrease of 5.38</td>
</tr>
<tr>
<td>Solved</td>
<td>increase of 100</td>
</tr>
</tbody>
</table>

Table 4. Percentage change for dynamic car

4.3 Quadrotor Planning

Quadrotor motion planning\(^2\) is a very hard motion planning problem\(^3\) even with a simplified dynamics model. Quadrotor motion using predefined waypoints has become a basic capability but generating motion plans autonomously is still a challenging research area. The dynamics of the quadrotor are defined as follows:

\[
m\ddot{p} = u_0 n - \beta \dot{p} - mg \tag{9}
\]

\[
\alpha \in (u, u_z, u_q)^T \tag{10}
\]

where \(p\) is the position, \(n\) is the Z-axis of the body frame in world coordinates, \(\alpha\) is the angular acceleration, \(m\) is the mass and \(\beta\) is the damping.
The system is controlled through,
\[ u = (u_0, u_1, u_2, u_3) \]

The control inputs include the mass normalised collective thrust, the roll rate about the body axes \( u_1 \), the pitch rate \( u_2 \) and the yaw rate \( u_3 \).

The quadrotor benchmark was carried out in ‘Cubicles’ environment as shown in Fig. 3, with a maximum time limit for every planner execution set to 10 s for one trial and to 100 s for another trial.

As seen in Table 5, the goal direction approach defined in this paper improves on the solution length. Even in a high dimensional environment like quadrotor planning, the directed approach presents itself as a valuable addition to the RRT algorithm. The improvement in solution becomes significant as time of planning increases and so would be profited by higher computing power.

5. WORK IN PROGRESS

Currently, work is being carried out to implement GRRT algorithm in a real world environment with quadrotor as an unmanned vehicle platform. This serves the cause of furthering the research on autonomous quadrotor planning as well as to analyse the proposed algorithm.

6. CONCLUSION

In this paper we have presented the first results for an algorithm which is proposed to overcome the local collisions caused by goal bias in unmanned vehicles along with improving the solution length and computing time which is required to perform real-time planning. The GRRT algorithm presents a faster solution with shorter solution length and would be useful in the case of large search spaces and in places where planning is to be accomplished in previously unknown search spaces. The goal bias is introduced by a modified approach in the selection of sampled random state. The test results of Goal directed RRT algorithm shows significant improvements in time and solution lengths. The algorithm was formulated in order to overcome the disadvantages of heuristic probability shaping. This goal directed approach can be considered for different types of sampling based planners. The effectiveness of this method will be determined with experiments on unmanned vehicles in unknown environment.

REFERENCES

CONTRIBUTORS

**Mr David Boon Moses E.** obtained his ME (Avionics) from Madras Institute of Technology (MIT), Anna University. Currently pursuing his PhD (Avionics) at MIT, Anna University. He is working in the area of autonomous UAV motion planning. His areas of interest include robotic simulation and motion planning. In the current study, he conceived the algorithm, performed simulation and benchmarked the results.

**Dr G. Anitha** obtained her MTech (Control & Instrumentation) from IIT, Madras and PhD from Madras Institute of Technology, Anna University. Presently, she is working as Assistant Professor in the Department of Aerospace Engineering, MIT campus, Anna University, Chennai. Her areas of interest include avionics system, navigation guidance and control, and image processing. In the current study, she was involved in conception of the algorithm along with defining the framework for simulation.