Recreating the Informed RRT* Algorithm

Timothy Sweeney
Applied Graduate Engineering
University of Maryland
College Park, MD
tsweene1@umd.edu

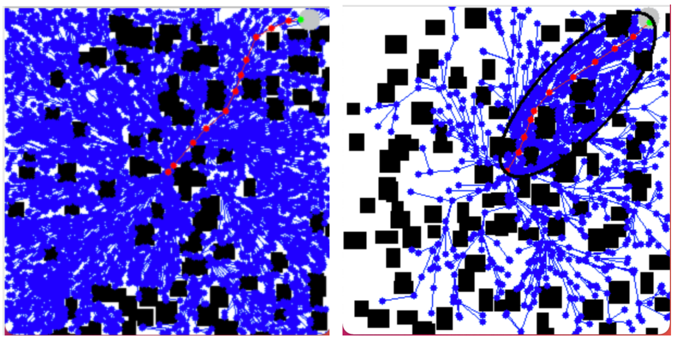
Amr Narmouq
Applied Graduate Engineering
University of Maryland
College Park, MD
amr.narmouq@gmail.com

***Abstract*—This paper attempts to recreate the results of an earlier paper titled *Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic* written by Jonathan Gammell, Siddhartha Srinivasa, and Timothy Barfoot from the Autonomous Space Robotics Lab at the University of Toronto Institute for Aerospace Studies.**

The paper introduces a novel method for improving the RRT* algorithm by confining the area that is randomly sampled to an ellipsoidal region. By introducing a heuristic search, the Informed RRT* algorithm converges to an optimal solution in significantly reduced time.

This paper reproduces the tests and results of the original paper, discuses the methods used to implement the algorithm, and examines a necessary step of the algorithm not mentioned by the original paper.

***Keywords*—RRT, RRT*, Informed RRT*, heuristic search**



RRT Algorithm*
C_{BEST}: 0.926
Nodes: 3421
Time: 31.0 seconds

Informed RRT Algorithm*
C_{BEST}: 0.929
Nodes: 924
Time: 31.0 seconds

Fig. 1. Example of RRT* and Informed RRT*

I. Introduction

Path planning is one of the fundamental problems of mobile robotics. Path planning can be described as the problem of finding a viable path from one position to another in a given map. For a robot to navigate autonomously, it must be able to do this. There are two general approaches to solving the path planning problem: through graph searches and through stochastic searches.

Graph searches, such as the breadth first search, create a graph of the environment by discretizing the space and connecting nodes to one another in parent child pairs. One of the most popular breadth first searches is Dijkstra's algorithm, which covers an entire map in nodes finding the optimal path through the environment with the lowest cost to traverse. This process is both *resolution complete* and *resolution optimal* as it searches every possible node of the sample space without biasing, up until the goal node is found. Because of the completeness of the search, this algorithm is much more computationally intensive than other more modern algorithms.

Dijkstra's Algorithm was improved upon by the invention of the A* algorithm, which uses assumed costs of nodes to prioritize searching nodes with a higher probability of finding the optimal path to a goal. This method of path planning is not as computationally intensive, is resolution complete, and will result in the optimal path from start to goal if a path exists.

The second method of path planning is through a stochastic search method, such as the Randomly-exploring Random Trees algorithm developed by Steven M. LaValle and James J. Kuffner Jr. These algorithms are less computationally intensive than graph searching when searching for a feasible path and scale better than graph searches in large maps. The downside to this approach is that the result you get is almost guaranteed to be a sub optimal path because of the random nature of the path creation.

To get the benefit of optimality you get from graph searching while maintaining the efficiency of stochastic search, researchers have been investigating Optimal RRTs, such as the RRT* algorithm. This algorithm introduces a process of map rewiring to improve the optimality of the resultant solution path. This algorithm continues to run after a first solution has been found, in an attempt to produce a more optimal solution.

The RRT algorithm has been further improved by the introduction of the Informed-RRT* algorithm, put forth by Gammell, et al. [1] This algorithm uses a heuristic-based sampling technique to increase the probability of randomly selecting a node that will decrease the solution cost. The median time to find a first solution for RRT* and informed

RRT* will be the same, but Informed RRT* will converge to an optimal solution in a greatly reduced time.

II. Background

The core idea behind Informed RRT* is that the straight-line path between the start position and end position will be the lowest cost, assuming cost is equivalent to Euclidean distance. This is a fact of basic geometry. From this axiom, we conclude that if we were to find a solution-path with a lower cost, that path must be closer to a straight line. We then select a random point from within this ellipse, and check to see if that random point can be used to find a solution with a lower cost. As the costs decrease, the area of the ellipse decreases and we converge to an optimal path.

To bound the area we are searching, we use ellipses. The ellipse is a convenient geometry for this task because it uses simple equations and does not have the unneeded area of the corners of rectangles. To find the measurements of the major and minor axis of the ellipse, we first determine the C_{min} and C_{max}.

C represents the cost of a given solution. C_{min} is the theoretical minimum distance between the two points. C_{max} is the current solution with the lowest cost. This is the C_{max} because any solution with a greater cost is suboptimal.

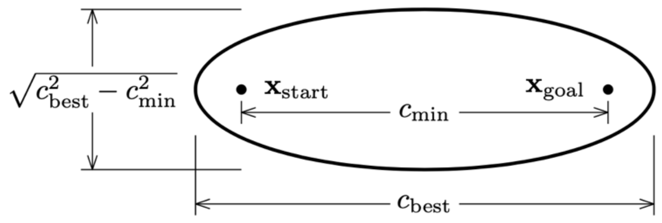

$\sqrt{c_{\text{best}}^2 - c_{\text{min}}^2}$
$\mathbf{x}_{\text{start}}$
$\mathbf{x}_{\text{goal}}$
c_{min}
c_{best}

Fig. 2. Measurements of the hueristic ellipse. Image taken from [1]

Using the measurements of the major and minor axis, we can easily get the equation of an ellipse centered at the origin of the XY axis. To get a point within the axis that describes our search environment we must follow a three-step process.

1) Get a point in the ellipse centered at the origin. Using a pseudo random number generating technique, we get a random radius from [0,1) and a random theta from [0, 2π). X and Y are then calculated as such

$$x = Semi\ major\ axis * radius * \cos(theta)$$
$$y = Semi\ minor\ axis * radius * \sin(theta)$$

2) Rotate the point. Use matrix multiplication to multiply the rotation matrix (Fig. 3) by the X, Y coordinates. In the matrix multiplication equation, Theta is the angle between the start and goal nodes, which is found using the atan2 function.

3) Translate the point. Find the center of the ellipse using the midpoint formula. Use the X value and Y values as offsets for the new random point.

$$M = (\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2})$$

Fig. 3. Midpoint formula

$$\mathsf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix},$$

Fig. 4. Rotation Matrix

III. Algorithm

The algorithm for Informed RRT*, as presented by Gammell et al. in [1] is the same as the RRT* algorithm presented by Karaman and Frazzoli in [2], with a few changes. The biggest and most obvious change is that the sampling is not done uniformly but instead uses the ellipsoidal heuristic. The other two changes in the pseudocode are that before finding the random sample we must first determine the C_{best}, and at the end of the for loop if a random point is selected and it is in the goal region, it is added to a set of points in the goal region, from which the point with the lowest cost is determined.

In simple terms, the algorithm is as follows. Determine the C_{best}. Select a random point. If a solution has already been found, use the ellipse to find the point. Find the closest node to the new node that can be connected to without collision with an obstacle. Find all of the nodes within a given radius to the new node. This is the set X_{near}. See if connecting the new node to any of the nodes in X_{near} produces a lower cost. Connect to whichever node results in the lowest cost. See if any of the nodes in the search radius could have a lower cost if they connected to the new node instead of their current parent node. If so, update the old node. If a random node is selected that is in the goal region, add it to the goal nodes set.

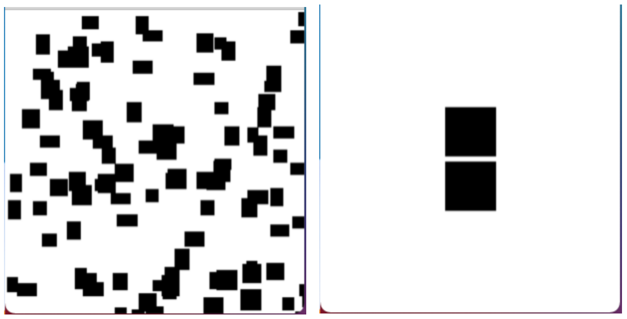
One thing that is different in our code than the code in [1] is that if an existing node is affected by the creation of the new node (an existing node would have a lower cost by going through the new node), the old node is added to a set $X_{affected}$. The $X_{affected}$ set contains all the nodes that are updated. Each node in the $X_{affected}$ set is examined to see if any of its neighbors would benefit from being routed through the new $X_{affected}$ node. Once the neighbors of a node in $X_{affected}$ have been examined, the node is removed from the set. This process continues until the $X_{affected}$ set is empty.

```
Algorithm 1: Informed RRT*($\mathbf{x}_{start}, \mathbf{x}_{goal}$)
$V \leftarrow \{\mathbf{x}_{start}\}$;
$E \leftarrow \emptyset$;
$X_{soln} \leftarrow \emptyset$;
$\mathcal{T} = (V, E)$;
for iteration $= 1 \dots N$ do
    $c_{best} \leftarrow \min_{\mathbf{x}_{soln} \in X_{soln}} \{\texttt{Cost}(\mathbf{x}_{soln})\}$;
    $\mathbf{x}_{rand} \leftarrow \texttt{Sample}(\mathbf{x}_{start}, \mathbf{x}_{goal}, c_{best})$;
    $\mathbf{x}_{nearest} \leftarrow \texttt{Nearest}(\mathcal{T}, \mathbf{x}_{rand})$;
    $\mathbf{x}_{new} \leftarrow \texttt{Steer}(\mathbf{x}_{nearest}, \mathbf{x}_{rand})$;
    if $\texttt{CollisionFree}(\mathbf{x}_{nearest}, \mathbf{x}_{new})$ then
        $V \leftarrow \cup \{\mathbf{x}_{new}\}$;
        $X_{near} \leftarrow \texttt{Near}(\mathcal{T}, \mathbf{x}_{new}, r_{RRT^*})$;
        $\mathbf{x}_{min} \leftarrow \mathbf{x}_{nearest}$;
        $c_{min} \leftarrow \texttt{Cost}(\mathbf{x}_{min}) + c \cdot \texttt{Line}(\mathbf{x}_{nearest}, \mathbf{x}_{new})$;
        for $\forall \mathbf{x}_{near} \in X_{near}$ do
            $c_{new} \leftarrow \texttt{Cost}(\mathbf{x}_{near}) + c \cdot \texttt{Line}(\mathbf{x}_{near}, \mathbf{x}_{new})$;
            if $c_{new} < c_{min}$ then
                if $\texttt{CollisionFree}(\mathbf{x}_{near}, \mathbf{x}_{new})$ then
                    $\mathbf{x}_{min} \leftarrow \mathbf{x}_{near}$;
                    $c_{min} \leftarrow c_{new}$;
        $E \leftarrow E \cup \{(\mathbf{x}_{min}, \mathbf{x}_{new})\}$;
        for $\forall \mathbf{x}_{near} \in X_{near}$ do
            $c_{near} \leftarrow \texttt{Cost}(\mathbf{x}_{near})$;
            $c_{new} \leftarrow \texttt{Cost}(\mathbf{x}_{new}) + c \cdot \texttt{Line}(\mathbf{x}_{new}, \mathbf{x}_{near})$;
            if $c_{new} < c_{near}$ then
                if $\texttt{CollisionFree}(\mathbf{x}_{new}, \mathbf{x}_{near})$ then
                    $\mathbf{x}_{parent} \leftarrow \texttt{Parent}(\mathbf{x}_{near})$;
                    $E \leftarrow E \setminus \{(\mathbf{x}_{parent}, \mathbf{x}_{near})\}$;
                    $E \leftarrow E \cup \{(\mathbf{x}_{new}, \mathbf{x}_{near})\}$;
        if $\texttt{InGoalRegion}(\mathbf{x}_{new})$ then
            $X_{soln} \leftarrow X_{soln} \cup \{\mathbf{x}_{new}\}$;
return $\mathcal{T}$;
```

```
Algorithm 2: Sample ($\mathbf{x}_{start}, \mathbf{x}_{goal}, c_{max}$)
if $c_{max} < \infty$ then
    $c_{min} \leftarrow ||\mathbf{x}_{goal} - \mathbf{x}_{start}||_2$;
    $\mathbf{x}_{centre} \leftarrow (\mathbf{x}_{start} + \mathbf{x}_{goal})/2$;
    $\mathbf{C} \leftarrow \texttt{RotationToWorldFrame}(\mathbf{x}_{start}, \mathbf{x}_{goal})$;
    $r_1 \leftarrow c_{max}/2$;
    $\{r_i\}_{i=2,\dots,n} \leftarrow \left(\sqrt{c_{max}^2 - c_{min}^2}\right)/2$;
    $\mathbf{L} \leftarrow \mathrm{diag}\{r_1, r_2, \dots, r_n\}$;
    $\mathbf{x}_{ball} \leftarrow \texttt{SampleUnitNBall}$;
    $\mathbf{x}_{rand} \leftarrow (\mathbf{CLx}_{ball} + \mathbf{x}_{centre}) \cap X$;
else
    $\mathbf{x}_{rand} \sim \mathcal{U}(X)$;
return $\mathbf{x}_{rand}$;
```

Fig. 5. Pseudocode

IV. Simulation



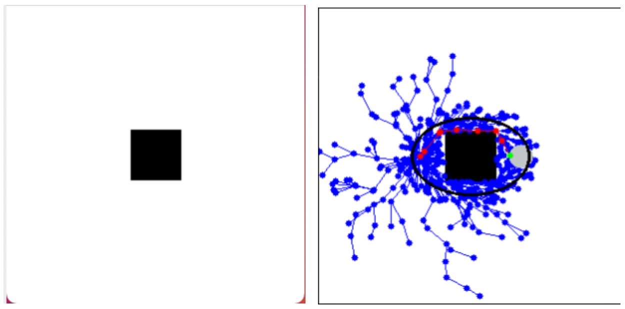


Fig. 6. Different map configurations that were used in the simulation to test both algorithms' performance

To test the difference between the performance of Informed RRT* and RRT* we ran a couple of different simulations. Each simulation target a specific point of comparison between the two algorithms.

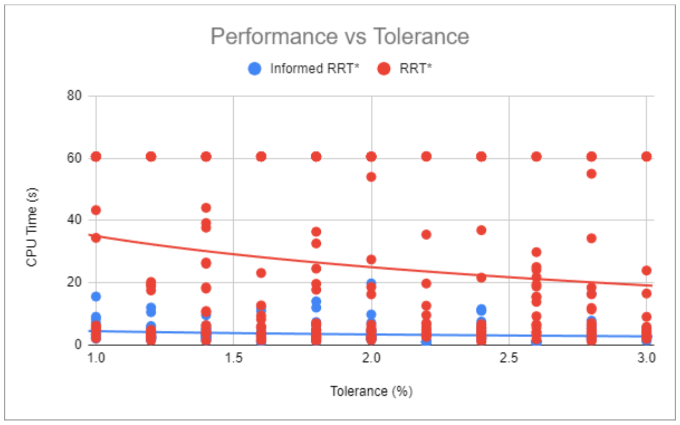


Fig. 7. The mediam computational time needed by both algorithms RRT* and Informed RRT* to find a path within the specified tolerance. The algorithms ran in an empty map with the same start and end locations and constants. Each trial was ran 20 independently.

The first simulation (performace vs tolerance) targeted the comparasion of CPU Time (s) the algorithm takes to reach the goal point at the desired tolerance. This simulation used a an empty map with a consistant map size, goal radius, start and goal locations and re-wiring radius for all the runs. The one componant that was changed between the different simulation runs was the tolerance allowed to reach the goal within. The tolerance ranged from 1% (cbest = 0.99) to 3% (cbest = .97).

In figure 7, we can observe the performance of both algorithms with the change of tolerance. Informed RRT* performance was seen to be much better and consistant that RRT*, the computational time was observed to almost maintain a minimul linear growth as the tolerance got tighter and tigher. On the other hand, RRT* showed an exponential growth pattern as the tolerance allowed got tighter.

The results of both algorithms were consistent with the expected outcome as Infomred RRT* focuses on the region that is know to contain the goal, while RRT* continues to search randomly through the entire region of the map in attempt to optimize the paths as more random points are generated.

The plot in figure 7 shows several simulation outcomes of the RRT* to be capped off at 60 seconds of run time, and the tigheter the tolerance the tighter of a spread towards the 60 seconds mark it gets. This is due to the result of only allowing the simulation to run for 60 seconds to try and reach the optimal solution. The simulation was capped off at 60 seconds for both algorithms, but due to the nature of the Informed RRT* algorithm it was able to find the optimal path that meets the tolerance in every case prior to hitting the time limit. In the case of RRT*, because of the nature of the algorithm and how it randomly grows it was not able to randomly locate the desired path within the time limit.

This simulation allowed us to confirm that the Informed RRT* algorithm allows you to locate the optimal path much quicker than RRT* and it is not affected by the tolerance. The reason being the nature of Informed RRT*'s algorithm of only searching within the goal region rather than across the whole map.

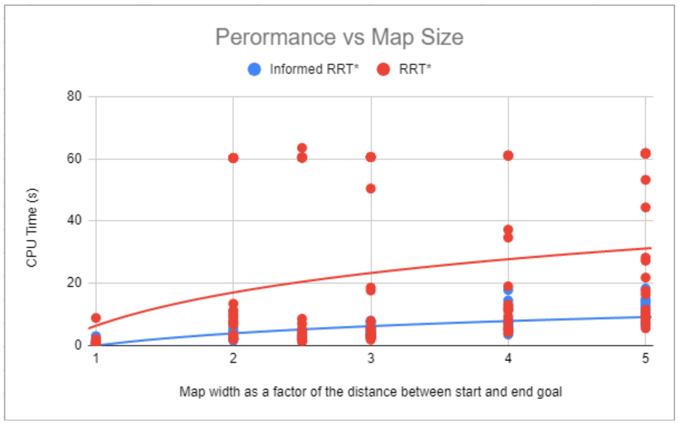


Fig. 8. The mediam computational time needed by both algorithms RRT* and Informed RRT* to find a path within the specified tolerance of 2%. The algorithms ran in an empty map with the same start and end locations and constants. The x-axis shows the ratio of the overall map width to the direct distance between the start and goal locations.

The second Simulation (performace vs Map Size) targeted the comparasion of CPU Time (s) the algorithm takes to reach the goal point within the set tolerance in different map sizes. This simulation used a an empty map with a consistant tolerance, goal radius, start and goal locations and re-wiring radius for all the runs. The one component that as changed between the different simulation runs was the map size. The map size ratio to the distance between the start and goal points increased from a 1:1 ratio to 5:1 ratio (map:goal distance). In this simulation we also set a time limit similar to the time limit in the first simulation to limit the computational time for each run.

In figure 8, we can observe the performance of both algorithms with the change of the map size. Informed RRT* performance was seen to be much better and consistant that RRT*, the computational time was observed to almost maintain a minimul linear growth as the tolerance got tighter and tigher. On the other hand, RRT* showed a logrithmic growth pattern as the map size increased.

Like in the first simulation the results were consistent with the expected outcome. Since RRT* is a randomly growing tree algorithm, the bigger the map size the bigger the possiblities for it to grow and optimize in random directions that are not towards the goal node. Due to our set time limitation for this simulation, the RRT* plot is constrained within 60 seconds. If we had not limited the time, a bigger growth would have observed similar to the first simulation.

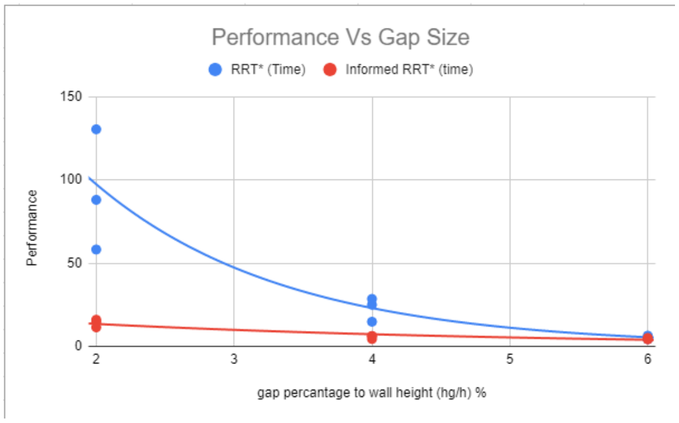


Fig. 9. The mediam computational time needed by both algorithms RRT* and Informed RRT* to find a path within the specified tolerance of 5%. The algorithms ran in a map with with a narrow passage between the start and goal points to demonstarete Informed RRT*'s ability to quickly manuever through tight passages. The same start and end locations and constants. The x-axis shows the ratio between the overall wall height and the percantage of the passage that passes through the wall.

The third simulation (Performace vs Gap Size) targeted the comparasion of CPU Time (s) the algorithm takes to reach the goal point within the set tolerance in different map configurations and to see how both algorithms perform around tight manuvers in the map. This simulation used the map shown in figure 9 with a consistant map size, tolerance, goal radius, start and goal locations and re-wiring radius for all the runs. The one component that as changed between the different simulation runs was the gap size between the two blocks.

In figure 9, we can observe the performance of both algorithms with the change of the gap to wall height (overall length of the two boxes) ratio. Informed RRT* performance was seen to be much better and consistant that RRT*, the computational time was observed to almost maintain a minimul linear growth as the tolerance ratio got smaller. On the other hand, RRT* showed an exponantial growth pattern as the gap ratio got smaller.

Like the first and second simulations, the results were consistent with the expected outcomes. Since RRT* is a randomly growing tree that does not take the goal location under consideration as it grows, the tighter the passage became the less of a chance it had to generate points of the path it had, and as a result the time taken to manuver through the passage grew as the passage became tigher. On the other hand Informed RRT* was unnoticably affected by the change of the passage size since it took into account the goal location and only searched within the goal region.

V. Conclusion

The Informed RRT* Algorithm outperforms the RRT* algorithm in time-to-convergence, number of computations, and ability to find difficult but optimal paths.

Future work that needs to be examined is the effect of a variable rewiring radius on convergence time. One of the factors slowing down convergence is the constant size of the rewiring radius. When the number of points in the map is small compared to the size of the map, it is best to have a large search radius. This allows for a greater likelihood of connecting a new node to an existing node. However, as the density of points in the map increases, a smaller radius is beneficial because it limits the number of nodes needed to be checked for updates.

VI. References

[1] 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," 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, 2014, pp. 2997-3004, doi: 10.1109/IROS.2014.6942976.

[2] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” IJRR, 30(7): 846–894, 2011.