Efficiency Enhancement in Path Planning - Expanding Path RRT*

ENPM 661 Project 5

1st Prathinav Karnala Venkata
M.Eng Robotics
University of Maryland
College Park, Maryland, USA
pratkv@umd.edu

2nd Pranav ANV
M.Eng Robotics
University of Maryland
College Park, Maryland, USA
anvpran@umd.edu

3rd Sarang Shibu
Robotics Program
University of Maryland
College Park, Maryland, USA
sarang@umd.edu

***Abstract*—This paper presents a comprehensive review of efficiency in path planning algorithms, with a particular focus on Rapidly-exploring Random Trees (RRT) and its variants. Path planning is a fundamental problem in robotics and artificial intelligence, with applications ranging from autonomous navigation to industrial automation. The efficiency of path planning algorithms directly impacts the performance and practicality of robotic systems. In this paper, we analyze several RRT variants, including RRT*, RRT Connect, Informed RRT*, and EP RRT*, discussing their key features, advantages, and limitations. Furthermore, we compare their efficiency in terms of computation time, solution quality, and scalability, providing insights into their applicability in different real-world scenarios.**

Keywords: Path planning, Rapidly-exploring Random Trees (RRT), RRT*, RRT Connect, Informed RRT*, EP RRT*

I. Introduction

A basic component of robotics, path planning is essential to allowing autonomous systems to travel successfully in challenging settings. For applications ranging from industrial automation to autonomous cars and robotics in agriculture, it entails creating a collision-free path from a starting point to a destination. Path planning is still difficult despite significant advancements since routes must be optimized for effectiveness, safety, and real-time responsiveness in constantly changing surroundings.

RRT, RRT-Connect, Informed RRT*, and EP-RRT* are some of the variants of the Rapidly Exploring Random Tree (RRT) algorithm that are examined in this research. The benefits of each algorithm vary with respect to speed, effectiveness, and path optimality. This study compares several approaches with the goal of determining which algorithm or combination of algorithms works best in different environmental conditions.

II. Literature Review

A. Rapidly-exploring Random Tree (RRT)

LaValle introduced the RRT algorithm in 1998, and because of its success in high-dimensional spaces, it has become a standard in path planning (LaValle, 1998) [4]. The algorithm builds a branching path from the start point towards the objective by randomly sampling the space. RRT is commended for its speed and ease of use, although it does not always provide the best route, which has led to advancements in improving its fundamental design.

B. RRT-Connect

In order to overcome some of the drawbacks of the original RRT, Kuffner and LaValle (2000) [2] developed RRT-Connect, which starts two trees—one from the target and one from the start point—and allows them to meet in the middle. This bidirectional method drastically cuts down on the amount of time it takes to discover a workable path, especially in congested areas where obstructions may be in the way. RRT-Connect, like RRT, does not, however, guarantee that the final path is optimal.

C. Informed RRT*

Based on the fundamentals of RRT, Karaman and Frazzoli's (2011) RRT* method [5] improves on the RRT by guaranteeing asymptotic optimality. As the number of iterations rises, RRT*, in contrast to its predecessor, revisits and improves the tree structure to identify the least expensive route to the objective. It accomplishes this by using two crucial processes: "Rewire," which modifies the tree's branches to lower the total path cost, and "ChooseParent," which determines the optimal parent for each new node while taking the path cost into account. This optimality comes at the expense of more computing work, especially in contexts where a thorough exploration is necessary.

To improve search efficiency, informed RRT* narrows the sampling space after an initial path is determined, concentrating on an ellipsoidal subset created around the shortest path. This technique, which was first presented by Gammell et al. (2014) [3], effectively reduces the search space, hastening the convergence of the best path. When previous algorithms, such as RRT*, may perform pointless explorations and the approach to the goal requires constant refining, informed RRT* performs especially well.

D. EP-RRT*

An advancement in the RRT algorithmic family, EP-RRT* combines the targeted sampling approach of Informed RRT* with the quick exploration capabilities of RRT-Connect. According to Ding et al. (2023) [1], EP-RRT* first uses methods similar to RRT-Connect to construct an initial feasible path, and then it expands this path to form a heuristic sample region. This method optimizes both speed and path quality in complex situations by maintaining the advantages of its predecessors while additionally concentrating computational resources on fine-tuning paths within a strategically selected area.

The advantages of its predecessors' work are combined in EP-RRT*, which also adds a specific technique for path expansion that greatly increases the efficacy of path planning, particularly in challenging situations like urban search and rescue, autonomous vehicle navigation, and robotic agriculture. This highlights the ongoing progress and specialization of route planning algorithms in robotics and makes EP-RRT* extremely helpful where efficient and reliable path planning is essential.

III. Rapidly-exploring Random Tree (RRT)

A. Introduction

RRT is a fundamental algorithm designed to efficiently explore high-dimensional spaces. It constructs a tree by iteratively adding nodes towards randomly sampled points, ensuring broad coverage of the search space.

B. Explanation

The Rapidly-exploring Random Tree (RRT) algorithm is a fundamental path planning technique widely used for its efficiency in exploring large and high-dimensional spaces. It works by incrementally building a tree from a start node by randomly sampling the search space and extending the tree towards these samples. Each new node is added by steering from the nearest existing node in the tree towards the sampled point, ensuring broad and rapid coverage of the space. The algorithm's simplicity and effectiveness make it a robust choice for solving various pathfinding problems, though it can sometimes produce suboptimal paths due to its random sampling nature. The primary strength of RRT lies in its ability to quickly explore complex environments, making it suitable for applications where finding a feasible path is more critical than finding the optimal path. The pseudo-code for RRT is shown in Fig. 1.

IV. RRT-Connect

A. Introduction

RRT-Connect extends RRT by growing two trees simultaneously, one from the start and one from the goal. These trees expand towards each other, aiming to connect and form a complete path.

```
RRT(q_start, q_goal)
1   T.add(q_start)
2   q_new ← q_start
3   while(DISTANCE(q_new, q_goal) > d_threshold)
4     q_target = RANDOM_NODE()
5     q_nearest = T.NEAREST_NEIGHBOR(q_target)
6     q_new = EXTEND(q_nearest, q_target, expansion_time)
7     if(q_new != NULL)
8       q_new.setParent(q_nearest)
9       T.add(q_new)
10  ResultingPath ← T.TraceBack(q_new)
11  return ResultingPath
```

Fig. 1. RRT psuedocode.

B. Explanation

RRT-Connect enhances the basic RRT algorithm by implementing a bidirectional growth strategy, where two trees are grown simultaneously from the start and goal positions. These trees expand towards each other, aiming to connect and form a continuous path between the start and goal. This approach significantly improves the efficiency of the search process, as the trees are more likely to meet in fewer iterations compared to the unidirectional growth of standard RRT. RRT-Connect's ability to reduce computational time while maintaining robust exploration makes it an effective solution for many real-time path planning scenarios. The bidirectional strategy also helps in navigating through narrow passages and complex environments more effectively than the standard RRT. The pseudo-code for RRT is shown in Fig. 2.

```
BUILD_RRT(q_init)
1  T.init(q_init);
2  for k = 1 to K do
3      q_rand ← RANDOM_CONFIG();
4      EXTEND(T, q_rand);
5  Return T
```

```
EXTEND(T, q)
1  q_near ← NEAREST_NEIGHBOR(q, T);
2  if NEW_CONFIG(q, q_near, q_new) then
3      T.add_vertex(q_new);
4      T.add_edge(q_near, q_new);
5      if q_new = q then
6          Return Reached;
7      else
8          Return Advanced;
9  Return Trapped;
```

Fig. 2. RRT Connect psuedocode.

V. Informed RRT*

A. Introduction

An enhanced variant of the Rapidly-exploring Random Trees (RRT) algorithm, namely RRT*, is called Informed RRT*. By concentrating the search in the most promising region of the configuration space, it was intended to increase the effectiveness of determining the best course of action. The capacity of Informed RRT* to restrict the search space to an ellipsoidal subset that includes only those regions that might potentially offer a better solution than the best so far discovered is the main advantage over its predecessor, RRT*.

B. Explanation

Informed RRT* starts by finding an initial feasible path using the RRT* algorithm. Once a path is found, Informed RRT* defines an ellipsoidal region based on the cost of this path. This region is determined by the current best path cost and spans an ellipsoid with foci at the start and goal points. The size and shape of the ellipsoid are dynamically adjusted based on improvements in the path cost found during the search process. By focusing subsequent sampling within this ellipsoid, Informed RRT* significantly reduces the search area, which in turn reduces the number of samples needed to potentially find a more optimal path. This targeted sampling continues until a termination condition is met, such as a maximum number of iterations or a satisfactory path cost threshold.

The pseudo-code for RRT is shown in Fig. 4 and Fig. 5, algorithm 1 of the Informed RRT* establishes the foundational framework for constructing a path planning tree, aiming to progressively refine the path by integrating nodes that potentially lower the overall path cost. This algorithm initiates with a tree rooted at the starting position and iteratively expands it by incorporating new nodes. These nodes are selected through a sampling process delineated in Algorithm 2, which conducts selective sampling within a hyper-ellipsoid region. This region is activated once an initial feasible path is established, and it is strategically calculated based on the cost of the best path found up to that point, c_{best}. The focus of the search is thus oriented towards areas more likely to yield path improvements. Algorithm 2 provides critical support to Algorithm 1 by offering a method to generate these samples from an ellipsoid that is dynamically shaped according to the current best path's cost metric, c_{best}. This targeted approach allows for a more efficient exploration of the search space, significantly enhancing the probability of finding more optimal paths as the algorithm progresses.

c_{best} is the 'long axis' or the major radius of the ellipsoid, representing the cost of the best path currently known. This cost must be at least c_{min} and reduces as the algorithm finds more efficient paths. c_{min} is the minimal distance or cost between the start and goal. The representation of c_{min} and c_{best} is shown in Fig. 3 and the equation is shown below

$$\sqrt{\|x_{\text{rand}} - x_{\text{start}}\|^2 + \|x_{\text{rand}} - x_{\text{goal}}\|^2} \leq c_{\text{best}} \qquad (1)$$

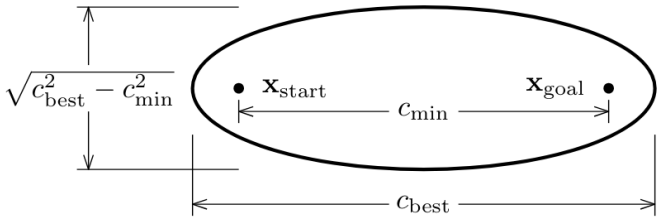

Fig. 3. c_{min} and c_{best}

Algorithm 1: Informed RRT*$(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}})$

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

Fig. 4. Informed RRT* pseudocode algorithm 1.

VI. EP-RRT*

A. Introduction

The Expanding Path RRT* (EP-RRT*) is an advanced path-planning algorithm developed to address the inefficiencies of the traditional RRT* algorithm, particularly in complex environments such as narrow corridors and mazes. Standard RRT* algorithms, while providing probabilistically complete solutions, often suffer from low sampling efficiency, which necessitates a high number of iterations, especially in environments characterized by long corridors. The EP-RRT* enhances the RRT* by integrating a path expansion heuristic, which

Algorithm 2: $\texttt{Sample}\left(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}, c_{\max}\right)$

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

Fig. 5. Informed RRT* pseudocode algorithm 2.

significantly improves both the efficiency and the quality of the path planning process.

B. Explanation

EP-RRT* starts its procedure akin to RRT-Connect, leveraging its efficient searching strategy to swiftly establish an initial feasible path. After this path is identified, EP-RRT* implements a novel heuristic sampling strategy within a dynamically determined path expansion area. This strategy is intentionally designed to iteratively refine and optimize the path.

Operational Mechanism: The operational mechanisms of EP-RRT* include:

1) **Initial Path Generation:** Utilizing the rapid exploration capabilities of RRT-Connect, EP-RRT* quickly identifies a feasible path from the start to the goal point.
2) **Path Expansion:** After establishing the initial path, the algorithm expands this path to form what is termed an "expansion area." This area is calculated based on the trajectory of the initial path and is adapted as the path evolves.
3) **Heuristic Sampling:** Within this expansion area, EP-RRT* performs heuristic sampling. This targeted sampling focuses on the most promising regions of the configuration space, improving the algorithm's efficiency by enhancing node utilization and accelerating convergence.
4) **Continuous Optimization:** As the path is iteratively optimized, the parameters defining the expansion area are dynamically adjusted, allowing the algorithm to focus more narrowly on the most beneficial parts of the configuration space.

C. Pseudocode

The pseudocode for the EP-RRT* (Expanding Path RRT*) algorithm is shown in Fig. 6 designed to efficiently find an optimal or suboptimal path in environments that are particularly challenging due to narrow corridors or complex obstacles.

$T_{init} \leftarrow RRT - Connect; T \leftarrow T_{init}$;
$\sigma_{init} \leftarrow GetPath(T)$;
$X_{expand} \leftarrow Expand(\sigma_{init})$;
for $i = 1$ to N **do**
  $x_{rand} \leftarrow HeuristicSample(X_{expand})$;
  $x_{nearest} \leftarrow Nearest(V, x_{rand})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{nearest})$;
  if $CollisionFree(x_{new}, x_{nearest})$ **then**
    $X_{near} \leftarrow Near(T, x_{new}, \eta)$;
    $x_{parent} \leftarrow ChooseParent(X_{near}, x_{nearest}, x_{new})$;
    $V \leftarrow V \cup \{x_{new}\}$; $E \leftarrow E \cup \{(x_{parent}, x_{new})\}$;
    $T \leftarrow Rewire(T, x_{new}, X_{near})$;
    $\sigma \leftarrow GetPath(T)$;
    $X_{expand} \leftarrow Expand(\sigma)$;
  end if
end for
return T;

Fig. 6. EP RRT* pseudocode.

D. Our Methodology of Implementation

We were not able to achieve the EP-RRT* algorithm as described in the research paper properly, but we took inspiration from the way it is being executed with the help of the greediness of RRT connect and the sampling efficiency of Informed RRT*. The method we chose to implement our code is by sampling between greedy exploitation of sampling the goal node or exploration. With the help of this technique, we were able to generate a path that was quick to reach the goal node and we performed rewiring to get an optimal path. The pseudocode for our modified EP-RRT* is shown in "Algorithm 1: Modified EP-RRT* Algorithm".

VII. Simulations

A. Time and Space Complexities of our code

Here we detail the time and space complexities of various RRT-based algorithms according to the code we have written:

1) **RRT Algorithm**
   - **Time Complexity:** $O(\text{max\_iterations}^2)$
     The time complexity arises from checking all vertices to find the closest one in each iteration, leading to a quadratic complexity relative to the number of iterations.
   - **Space Complexity:** $O(\text{max\_iterations})$
     The space complexity is linear with respect to the maximum number of iterations, as it primarily depends on the number of vertices stored.
2) **RRT-Connect Algorithm**
   - **Time Complexity:** $O(\text{max\_iterations}^2)$
     Similar to RRT, this complexity results from iterating through vertices in two trees, still leading to a quadratic relationship in the worst-case scenario.

Algorithm 1 Modified EP-RRT* Algorithm

Initialize tree with start node.
for each iteration **do**
  Decide between exploration or exploitation:
  if exploration **then**
    Sample a random point.
  else
    Directly sample the goal point.
  Find the nearest node in the tree.
  Steer from the nearest node towards the sampled point.
  if the new node is valid **then**
    Add the new node to the tree.
    Rewire the tree if necessary:
    Check if any nearby nodes should change their parent to the new node to reduce path cost.
    for each nearby node **do**
      if changing parent reduces cost **then**
        Change parent of the node to the new node.
      end if
    end for
    if the new node is near the goal **then**
      Stop the algorithm.
    end if
  end if
end for
Extract and return the path. =0

- **Space Complexity:** $O(\text{max\_iterations})$
  Stores two trees but only counts towards the total vertices up to the maximum iterations, maintaining a linear space complexity.

3) **Informed RRT* Algorithm**
   - **Time Complexity:** $O(\text{max\_iterations}^2)$
     Involves complex operations like choosing the best parent and rewiring, which depend on the number of vertices in each iteration, contributing to quadratic complexity.
   - **Space Complexity:** $O(\text{max\_iterations})$
     Similar to RRT and RRT-Connect, the primary space usage is from the vertices added to the tree, linear with respect to max_iterations.
4) **EP-RRT Algorithm**
   - **Time Complexity:** $O(\text{max\_iterations}^2)$
     Includes both exploratory and exploitative phases in each iteration and determines the closest vertex, which can lead to quadratic time complexity as the number of vertices grows.
   - **Space Complexity:** $O(\text{max\_iterations})$
     Primarily based on the storage of vertices generated during the iterations, leading to a linear space complexity in terms of the maximum iterations allowed.

These complexities highlight the scalability challenges and efficiency considerations inherent to each algorithm, especially as the size of the input (i.e., the number of iterations and complexity of the environment) increases.

B. Experimentation using Map 1

Here we are executing the code file corresponding to MAP1. The results will vary due to randomness but in most cases will work.

Here we have provided two Testcases for this MAP.

Testcase 1: Parameters:

- Enter start x-coordinate (0-600): 10
- Enter start y-coordinate (0-200): 10
- Enter goal x-coordinate (0-600): 590
- Enter goal y-coordinate (0-200): 10
- Enter step size: 10
- Enter number of iterations: 5000

1) RRT: RRT completed in 1.40 seconds and found route of length 73. The path has been visualized using Matplotlib and is shown in Fig. 7.
2) RRT-Connect: RRT-Connect completed in 0.05 seconds and found route of length 76 The path has been visualized using Matplotlib and is shown in Fig. 8.
3) Informed RRT*: Informed RRT* completed in 3.42 seconds and found route of length 88 The path has been visualized using Matplotlib and is shown in Fig. 9.
4) EP RRT*: EP-RRT* completed in 0.11 seconds and found route of length 74. The path has been visualized using Matplotlib and is shown in Fig. 10.

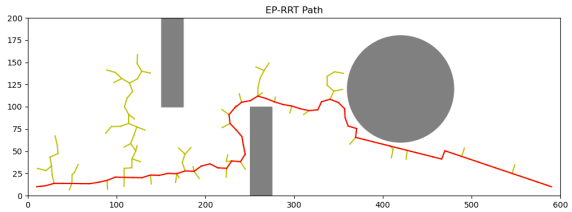

Fig. 7. Testcase 1 RRT simulation for MAP1.

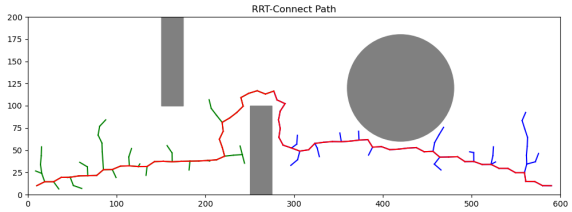

Fig. 8. Testcase 1 RRT-Connect simulation for MAP1.

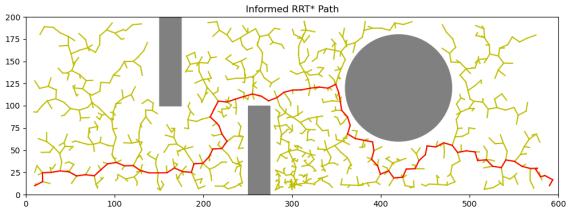

Fig. 9. Testcase 1 Informed RRT* simulation for MAP1.

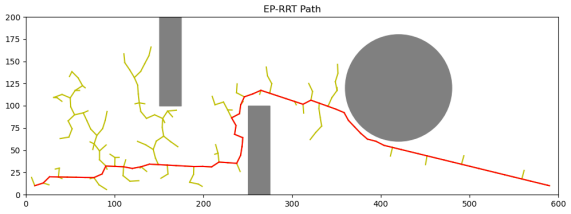

Fig. 10. Testcase 1 EP RRT* simulation for MAP1.

Testcase 2: Parameters:

- Enter start x-coordinate (0-600): 50
- Enter start y-coordinate (0-200): 100
- Enter goal x-coordinate (0-600): 550
- Enter goal y-coordinate (0-200): 150
- Enter step size: 10
- Enter number of iterations: 1000

1) RRT: RRT completed in 1.40 seconds and found route of length 73. The path has been visualized using Matplotlib and is shown in Fig. 11.
2) RRT-Connect: RRT-Connect completed in 0.05 seconds and found route of length 76 The path has been visualized using Matplotlib and is shown in Fig. 12.
3) Informed RRT*: Informed RRT* completed in 3.42 seconds and found route of length 88 The path has been visualized using Matplotlib and is shown in Fig. 13.
4) EP RRT*: EP-RRT* completed in 0.11 seconds and found route of length 74. The path has been visualized using Matplotlib and is shown in Fig. 14.

C. Experimentation using Map 2

Here we are executing the code file corresponding to MAP1. The results will vary due to randomness but in most cases will work.

Here we have provided two Testcases for this MAP.

Testcase 1: Parameters: Enter start x-coordinate (0-450): 10 Enter start y-coordinate (0-400): 10 Enter goal x-coordinate

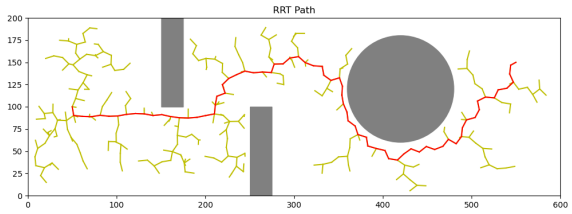

Fig. 11. Testcase 2 RRT simulation for MAP1.

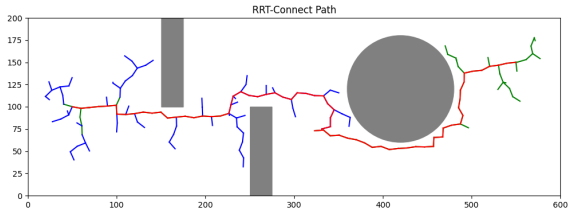

Fig. 12. Testcase 2 RRT-Connect simulation for MAP1.

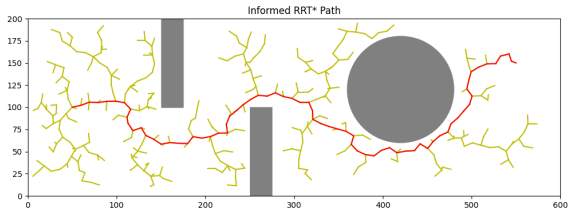

Fig. 13. Testcase 2 Informed RRT* simulation for MAP1.

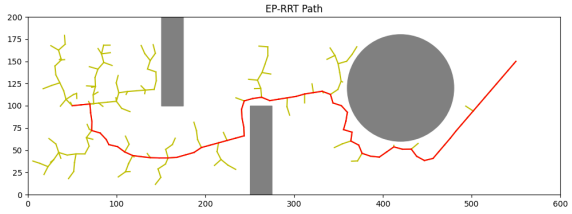

Fig. 14. Testcase 2 EP RRT* simulation for MAP1.

(0-450): 440 Enter goal y-coordinate (0-400): 10 Enter step size: 10 Enter number of iterations: 8000

- Enter start x-coordinate (0-600): 10
- Enter start y-coordinate (0-200): 10
- Enter goal x-coordinate (0-600): 440
- Enter goal y-coordinate (0-200): 10
- Enter step size: 10
- Enter number of iterations: 8000

1) RRT: RRT took 4.78 seconds and found path of length 91. The path has been visualized using Matplotlib and is shown in Fig. 15.
2) RRT-Connect: RRT-Connect took 0.07 seconds and found path of length 93. The path has been visualized using Matplotlib and is shown in Fig. 16.
3) Informed RRT*: Informed RRT* took 5.00 seconds and found path of length 92. The path has been visualized using Matplotlib and is shown in Fig. 17.
4) EP RRT*: EP-RRT took 0.12 seconds and found path of length 80. The path has been visualized using Matplotlib and is shown in Fig. 18.

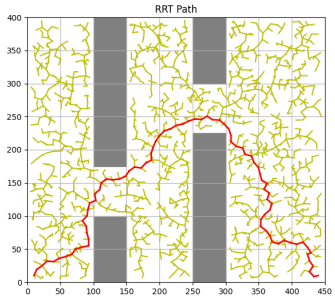

Fig. 15. Testcase 1 RRT simulation for MAP2.

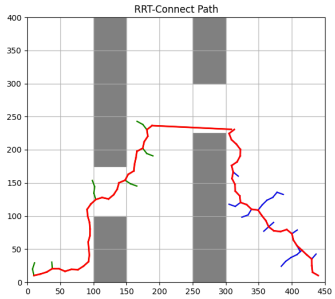

Fig. 16. Testcase 1 RRT-Connect simulation for MAP2.

Testcase 2: Parameters: Enter start x-coordinate (0-450): 50 Enter start y-coordinate (0-400): 350 Enter goal x-coordinate (0-450): 400 Enter goal y-coordinate (0-400): 50 Enter step size: 10 Enter number of iterations: 2000

- Enter start x-coordinate (0-600): 50
- Enter start y-coordinate (0-200): 350
- Enter goal x-coordinate (0-600): 400

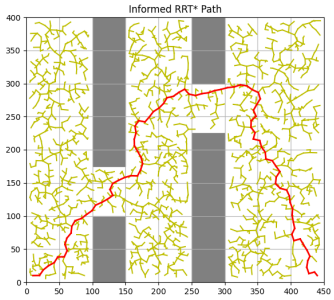

Fig. 17. Testcase 1 Informed RRT* simulation for MAP2.

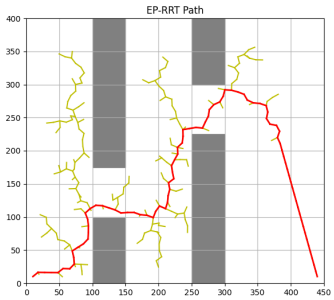

Fig. 18. Testcase 1 EP RRT* simulation for MAP2.

- Enter goal y-coordinate (0-200): 50
- Enter step size: 10
- Enter number of iterations: 2000

1) RRT: RRT took 1.31 seconds and found path of length 88. The path has been visualized using Matplotlib and is shown in Fig. 19.
2) RRT-Connect: RRT-Connect took 0.14 seconds and found path of length 88. The path has been visualized using Matplotlib and is shown in Fig. 20.
3) Informed RRT*: Informed RRT* took 3.18 seconds and found path of length 88. The path has been visualized using Matplotlib and is shown in Fig. 21.
4) EP RRT*: EP-RRT took 0.61 seconds and found path of length 84. The path has been visualized using Matplotlib and is shown in Fig. 22.

Here we can observe that our modified EP-RRT* is performing quite well, and is giving us an output that is under a second in most cases. We are able to generate the trees for each of the RRT algorithms but there can be cases where a path won't be generated, in such cases increasing the iterations proved to solve this issue.

RRT Connect has some bugs while displaying the path and Informed RRT* is taking more than one second to give us the result. We have tried to reduce the errors and bring out a better result, as there is scope for improvement and we will continue our work to fix this issue.

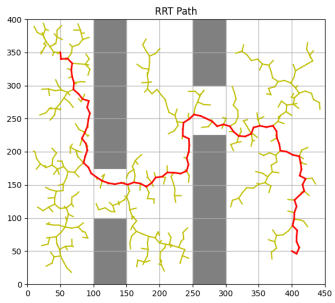

Fig. 19. Testcase 2 RRT simulation for MAP2.

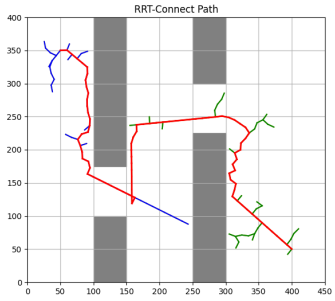

Fig. 20. Testcase 2 RRT-Connect simulation for MAP2.

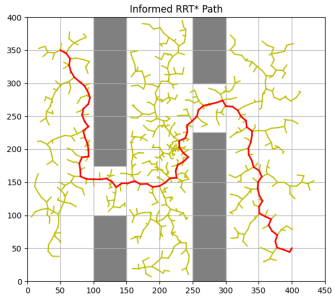

Fig. 21. Testcase 2 Informed RRT* simulation for MAP2.

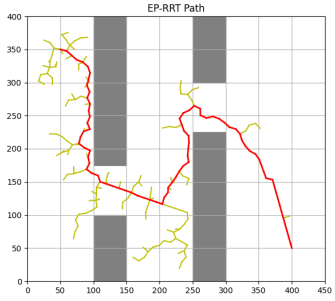

Fig. 22. Testcase 2 EP RRT* simulation for MAP2.

VIII. Conclusion

In summary, this exploration of RRT algorithms such as RRT, RRT-Connect, Informed RRT*, and EP-RRT* has highlighted their potential in robotic path planning. Each variant is tailored to address specific challenges, showcasing strengths in efficiently navigating complex environments. Though we were not able to achieve EP-RRT* as per the research paper, we were able to achieve a modified algorithm that is fast and is able to reach the goal node efficiently. However, issues such as parameter sensitivity, handling complex obstacles, algorithmic convergence, and computational efficiency remain significant challenges.

References

[1] J. Ding, Y. Zhou, X. Huang, K. Song, S. Lu, and L. Wang, "An improved RRT* algorithm for robot path planning based on path expansion heuristic sampling," *Journal of Computational Science*, vol. 67, pp. 101937, 2023. DOI: https://doi.org/10.1016/j.jocs.2022.101937.

[2] J.J. Kuffner and S.M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. 2000 ICRA. Millennium Conf. IEEE Int. Conf. on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, pp. 995-1001, 2000. DOI: 10.1109/ROBOT.2000.844730.

[3] 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*, pp. 2997-3004, 2014. DOI: 10.1109/IROS.2014.6942976.

[4] S. M. LaValle, "Rapidly-exploring random trees: a new tool for path planning," *The annual research report*, 1998. Available: https://api.semanticscholar.org/CorpusID:14744621.

[5] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *CoRR*, vol. abs/1105.1186, 2011. Available: http://arxiv.org/abs/1105.1186.