RRT-Connect algorithm with Artificial Potential Fields for enhanced path planning

Raajith Gadam
Department of MAGE
University of Maryland,
College Park,MD,USA
raajithg@umd.edu

Advait Kinikar
Department of MAGE
University of Maryland,
College Park,MD,USA
kinikara@umd.edu

***Abstract*—There have been attempts to address the randomness, blindness, and low efficiency issues with RRT Connect using a variety of approaches. This report describes how artificial potential fields were used to overcome the issue. It validates the outcomes and puts into practice a potential fix offered in "Path Planning of Manipulator Based on Improved RRT Connect Algorithm" by Ying Zhu et al. The implementation is being done in Python and tested on a 2D Maze. It is evident from the implementation results that the new algorithm suggested by the original authors offers shorter paths and executes faster. The RRT Connect method is used to compare the suggested solution.**

***Index Terms*—Path Planning, Sampling based Algorithms, RRTConnect, Artificial Potential Fields, BSpline**

I. Introduction

In both manipulator-based robotics and mobile robotics, path planning has emerged as a key field of study. Finding a collision-free route for a robot to move it from its starting configuration to its desired configuration while avoiding obstacles in its surroundings is the essence of path planning. The focus of current research is mostly on optimizing the code and shortening the paths. Sampling-based algorithms have grown in prominence among many approaches to tackling these issues. Algorithms based on sampling have several benefits. They perform better computationally, can handle dynamic settings, and are well suited for high dimensional configuration spaces and complex environments. RRT [1] is a popular sampling-based algorithm. By extending the tree towards unexplored regions and randomly sampling the configuration space, this technique gradually creates a tree structure. Now, RRT may efficiently explore complex environments, but its computational complexity is restricted and it is unable to trace optimal paths.

To overcome this problem, the RRT Connect [2] algorithm was introduced. It grows two distinct trees from each configuration and attempts to connect the start and goal configurations in a holistic way. It is "probabilistically complete," which means that if a solution exists, it will undoubtedly find it given enough time. It can also adjust to changes in the surroundings and is efficient. It still has optimality problems, though. Moreover, it could have trouble navigating through spaces with tight corners or passageways.

In [3], Ying zhu et al. provide an improvisation over the RRT Connect algorithm. 'Artificial Potential Fields' are included in order to direct the random tree toward the target and reduce the blindness of the path searching. After that, they use "Dijkstra's algorithm" to optimize the path and reduce its length. Lastly, "B-spline curves" are used to smooth the path. The 'B-spline curves' are used so that a continuous trajectory is obtained the manipulator could avoid issues like abrupt direction changes during grasping, etc. The authors then test the algorithm in MATLAB in a simple 2D Map with obstacles, for a point robot.

Our project is a humble attempt at implementing the improvements suggested in [3] in a 2D environment in Python and validating our results with the authors results. We follow the same methodology as proposed by the authors and compare the improved RRT Connect algorithm with the RRT Connect algorithm

II. Literature review

Introduced by LaValle and Kuffner [1], the "Rapidly exploring Random Trees (RRT)" method is widely used in autonomous robotic motion planning because of its capacity to manage obstacles and differential constraints, such as kino-dynamic and non-holonomic constraints. RRT's fundamental idea is to generate points at random and connect them to the nearest accessible node in the tree while making sure that every connection stays clear of obstacles. The algorithm continues until a node is generated within the goal region or a predefined limit is reached. Now, the RRT algorithm has a few of drawbacks such as its high computing cost or its inability to guarantee path generation optimality, which makes it less appealing to be used.

Now, RRT can generate paths, which are not continuous, meaning, the paths generated have sharp turns or discontinuities. Also, in a few cases, it may happen that RRT is not able to give a path. To address these issues and the ones discussed above, in 2000, LaValle and Kuffner introduce RRT Connect [2]. RRT-Connect is an extension of RRT which starts by growing two trees from the initial configuration

of the robot towards the goal configuration and from the goal configuration towards the initial configuration.Each tree is grown seperately and a random sample taken from the configuration space determines the growth direction of each tree. The algorithm makes an effort to link the two trees once they have both grown to a suitable size. To accomplish this, the algorithm locates a node that is close the other tree and then 'joins' two trees, which would give a path from initial configuration to goal configuration. Although the algorithm is an improvement over RRT, it still has the problem of being computationally expensive. Also, since the nodes are randomly sampled, the algorithm struggles with the issue of being blind, meaning it doesn't know its direction of exploration. It can happen that it may start exploring in direction away from the desired location, taking longer time to generate the path.

To address the above issues Ying Zhu et. al. [3] use 'Artificial Potential Functions' "to drive the random tree towards the target and accelerate planning". They then use Djikstra's algorithm, to optimize and shorten the path and use BSpline curves, to smoothen the path. Although, the authors intend to use the improved algorithm for manipulators, they test it out on a point robot in a 2D map.

Now, as discussed above, to drive the path search towards the goal and simultaneously avoid obstacles, Ying Zhu et. al use Artificial Potential Fields. Now Artficial Potential Fields or APFs are virtual force fields, in which the goal configuration has an 'attractive force' and the obstacles have a 'repulsive force'. In simple terms, when APFs, are introduced in path an motion planning, the robot and its path are attracted towards the goal and repulsed from the obstacles. The implementation of APFs for obstacle avoidance and path planning is discussed by O. Khatib [6]. In this paper, the author explains what Artificial Potential Fields are. Then the author implements APFs in COSMOS and PUMA560 robots for demonstration.

In [3], the authors also smooth the obtained path using BSpline Curves. In [5], Koch and Kesheng, show how Bspline can be used in trajectory planning in manipulators and robots. They describe, how Bspline can be used to construct joint trajectories for manipulators. They first describe the motion in terms of position and orientation of the end-effector of the manipulator. They then consider a 6 joint robots and create joint variables, which include position and orientation of each joint. They get 6 joint variables. Then linear combinations of B-Splines are used, to fit the sequence of joint variables for each of the six joints. They then use a computationally very simple formula to generate spline.

Now before smoothing the path, the authors in [3], first optimize and shorten the path using Dijkstra's algorithm. Now, Dijkstra algorithm, is a graph search algorithm, that finds the shortest path in a tree, using non-negative edge path costs. In the [4], the Dijkstra's algorithm is explained.

III. Methodology

To understand the methodology, we first need to understand all the concepts involved in implementing the algorithm. The 3 concepts around whicch the algorithm is centered are:
a. RRT Connect algorithm
b. Artificial Potential Fields
c. Djikstra's Algorithm
d. BSpline Following is a brief explanation of the above concepts.

A. RRT Connect Algorithm

As discussed earlier, the RRT Connect algorithm is an extension of the RRT algorithm, specifically designed to make the paths generated more continuous and feasible. This is achieved by growing two trees from both the stand and the end positions and then connecting the two trees to get a continuous path. Following are some steps taken in the RRT Connect algorithm:

a. Initialization:
In this step, we start two trees from both the start and the goal position, wherein in both the trees, the first node is the start node and the goal node respectively.

b. Expansion:
Then, for both the trees, we randomly sample a point from the configuration space. Determine the nearest node in the tree to the point and add the point to the tree, with nearest node as the parent. It should be noted that the randomly sampled point is being taken from a set max distance. For example, we set some maximum distance say 1, then the randomly sampled point is within 1 unit radius of its nearest node.

c. Connection:
After expanding the trees, we then check if the trees are within, some threshold distance of each other. If they are, we attempt to connect them. To connect, we find the nearest node in each tree. Then, we check if a feasible path exists between corresponding nodes of each tree. If the feasible path exists, we connect the trees, by linking corresponding nodes.

d. Path extraction
We then find the final path from start node to goal node, by concatenating the path from start position to connection point and from goal position to connection point.

```
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;

CONNECT(T, q)
1  repeat
2      S ← EXTEND(T, q);
3  until not (S = Advanced)
4  Return S;

RRT_CONNECT_PLANNER(q_init, q_goal)
1  T_a.init(q_init); T_b.init(q_goal);
2  for k = 1 to K do
3      q_rand ← RANDOM_CONFIG();
4      if not (EXTEND(T_a, q_rand) = Trapped) then
5          if (CONNECT(T_b, q_new) = Reached) then
6              Return PATH(T_a, T_b);
7      SWAP(T_a, T_b);
8  Return Failure
```

Fig. 1. Pseudo Code for RRT Connect

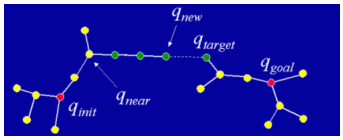

Fig. 2. Trees attempting to connect [9]

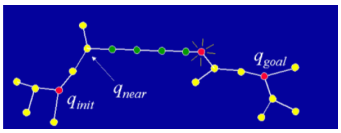

Fig. 3. Trees Connecting [9]

Fig1. is the pseudo code for RRT Connect. The primary function in the pseudo code is the 'EXTEND' function. It involves, finding the nearest node, q_near, in the existing tree to a given target configuration, q_target, based on a defined distance metric in the configuration space. The demonstration can be seen in Fig2. If the distance between q_near and q_target is less than a predefined parameter called the RRT_step size (e), a new branch is attempted directly to q_target. Otherwise, an intermediate node, q_new, is generated along the line connecting q_near and q_target at a distance of e.If the new branch is free from collisions, it is added to the tree. If an intermediate node was created, the algorithm continues to extend the new branch towards q_target (the "CONNECT" operation) until either the target configuration is reached or a collision is encountered. Thus, the planning queries involve constructing two RRTs rooted at the start and goal configurations. The algorithm aims to connect these trees, implicitly defining a collision-free path from start to goal. Whenever a new branch is successfully added to one tree, the terminal node of that branch becomes the target configuration for the other tree.

B. Artificial Potential Functions

Artificial potential fields are 'virtual force field' inspired by physical fields such as electrical, magnetic, and gravitational fields. APF divides space within a grid of cells, with the obstacles and the goal within this grid. The algorithm assigns an artificial potential field to every point in the world using potential field functions. The robot then moves from the highest potential to the lowest potential, with the goal node having the lowest potential and the starting node having the maximum potential

C. How does APF work?

a. Attractive function:
The attractive field is generated by the goal node, which pulls the robot towards it.

$$U_{att}(X) = \frac{1}{2}K_a(X - X_d)^2$$

Above is the equation that describes or simulates the attractive field generated by the goal node. It represents the force exerted on the robot by the goal configuration, pulling the robot towards it. The equation is inspired from Coulombs law for electric forces or Newtons law of universal gravitation. In the equation, $U_{att}(X)$ is the attractive force acting on the robot. X is the goal configuration and X_d is the current configuration of the robot. $(X - X_d)$ indicates a vector pointing from current configuration to goal configuration. It gives the direction in which the robot should move. It is evident from the equation that the strength of the attraction is highest, when the robot is farthest from the goal and it decreases with distance from the goal. Here, the constant K determines, the strength of attraction.
b. Repulsive Function:
The repulsive field is generated by obstacles, that pushes the robot away.

$$U_{rep}(X) = \begin{cases} 1/2k_r[\frac{1}{\rho(X,X_g)} - \frac{1}{\rho_0}]^2 \|X - X_d\|_2, \rho \le \rho_0 \\ 0 \qquad\qquad\qquad\quad ,\rho > \rho_0 \end{cases}$$

Above is the equation that describes or simulates the repulsive field generated by the obstacles. It represents the force exerted on the robot by the obstacles, pushing the robot away from it. It helps the robot avoid collisions and navigate around obstacles. In the equation, $U_{rep}(X)$ is the attractive force acting on the robot. K is the constant, that determines the strength of repulsion. $[\frac{1}{\rho(X,X_g)} - \frac{1}{\rho_o}]$, represents the gradient of the repulsive force. It gives the direction in which the repulsive force is acting. It is evident from the equation, that the strength of repulsion increase, with distance from the obstacles.

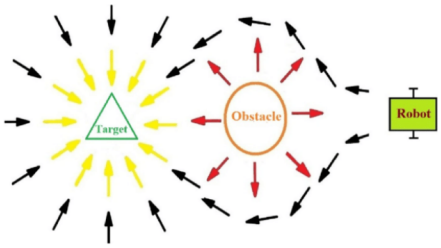

Fig. 4. Artificial Potential Field Acting on a robot [7]

The Fig. 4. above shows exactly how the robot would behave with Artificial Potential Fields used. The arrows represent the attractive and repulsive force vectors described in the equations above. APF can be used in a generic 'steer' function that dictates the direction in which the autonomous robot would move.

D. Dijkstras Algorithm

As discussed above, Dijkstra algorithm, is a graph search algorithm, that finds the shortest path in a tree. Starting from a source node, the algorithm visits all the nodes in the graph, iteratively choosing the unvisited node that is closest to the source, updating the distances to its neighboring nodes, and marking it as visited. This process continues until all nodes have been visited, resulting in the shortest path from the source to all other nodes in the graph. Following are some steps taken in Dijsktra's algorithm:

a. Initialization:
The algorithm first, starts by setting the 'cost-to-come' (distance) for the start node as zero. The cost-to-come for the rest of the node is set to infinity. It also initializes a set of unvisted nodes.

b. Node Selection:
From the set of unvisted nodes (next to the source node), the algorithm selects the node with the smallest cost-to-come from source node. This node is then marked as visited.

c. Updating the node:
For each neighbor of the selected node, if the distance to the neighbor through the selected node is less than the previously recorded distance, the distance is updated. This is an essential step in determining the shortest path..

d. Iteration:
The 2nd and the 3rd step is repeated until the goal node is visted. At the end, the shortest path from source to goal node and hence the shortest path from source to each node along the path is obtained.

```
Create two empty lists named OpenList and ClosedList
Get the initial (Xi) and goal node (Xg) from the user
OpenList.put(Xi)
While (OpenList not EMPTY) and (Not reached the goal) do
      x ⟵ OpenList.get()
      Add x to ClosedList
      if x = Xg
            Run backtrack function
            return SUCCESS
      else
            forall u ∈ U(x)
                  x' ⟵ f(x,u)      # Generating each valid action
                  if (x' ∉ ClosedList) and (NOT in the obstacle space)
                        if (x' ∉ OpenList) or (CostToCome(x') = ∞)
                              Parent(x') ⟵ x
                              CostToCome(x') ⟵ CostToCome(x) + L(x,u)
                              Cost(x') ⟵ CostToCome(x')
                              OpenList.put(x')
                  else
                        If Cost(x') > CostToCome(x) + L(x,u)
                              Parent(x') ⟵ x
                              CostToCome(x') ⟵ CostToCome(x) + L(x,u)
                              Cost(x') ⟵ CostToCome(x')
return FAILURE
```

Fig. 5. Pseudo Code for Dijkstra's Algorithm

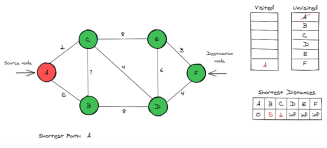

Fig. 6. Dijsktra's Algorithm on a simple graphv [10]

Fig. 5. is the pseudocode for Dijkstras algorithm .It consists of 2 lists, an 'Open list' and a 'closed list'. The algorithm first puts the source node in the open list. Then until the open list is not empty, the algorithm keeps on getting nodes from the open list. It checks if node is the goal node. If it is the goal node, then the algorithm stops. If it is not, then it checks its neighboring nodes and selects the node with the smallest distance. It then puts the source node in the closed list and adds the nearest node to the open list. This process continues until the goal node is reached. Fig. 6. shows how the algorithm works visually.

E. BSpline

B-splines, short for Basis splines, are a type of curve defined by a set of control points and a set of basis functions, which together determine the shape of the curve. These curves are smooth, continuous, and can be used to represent complex shapes with fewer parameters compared to other curve representations.
As discussed above, B-Splines are defined by a set of control points and a set of basis function.

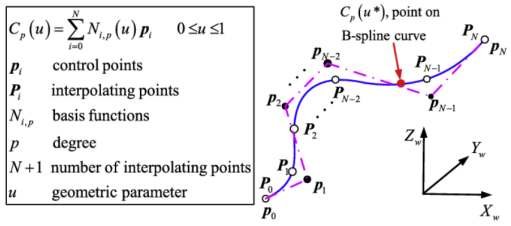

Fig. 7. BSpline curve and its equation [8]

The Fig. 7. shows a Bspline curve along with its equation. P_i are the set of control points, defining the curve and

$N_{i,p}(U)$ is the basis function associated with each control point. $C_p(U)$ is the BSpline curve, which is the weighted sum of the control points, where each control point is multiplied by its corresponding basis function. The basis functions are piecewise polynomial functions, that depend on the curve's degree p and the parameter U within the range of the curve.

B-splines are used in path planning to create curved paths that go through or close to a set of waypoints. These waypoints serve as the B-Spline control points, which are positioned to guarantee that the curve passes through them. Sharp twists in the path can be smoothed out by increasing the number of control points and modifying the degree of the B-Spline. In order to accomplish this and have the curve transition between segments more smoothly, more control points are added in between two already-existing control points. The degree of smoothing can be controlled by varying the curve's degree; smoother transitions are produced at higher degrees.

F. Improved RRT Connect algorithm

As discussed above, the improved RRT Connect allgorithm, implements APFs to guide and expand the tree towards the target destination and reduce the 'blindness' of the algorithm, whilst avoiding obstacles. Following are some steps taken in the algorithm. It should be noted that the steps are unique to the code that we have implemented in this project and there could be a different way to write the code.

1. Initialize the start and goal configurations, obstacles, step size, and other parameters(attractive force constant, repulsive force constant, radius of influence near the obstacles, number of iterations).

2. Create two trees: forward and backward.

3. Add the start node to the forward tree and the goal node to the backward tree.

4. Repeat the following until a connection is found or the maximum number of iterations is reached:

   a. For each tree in forward and backward:
   b. Sample a random point in the workspace.
   c. Find the nearest node in the tree to the random point.
   d. **Steer** the nearest node towards the random point while considering attractive and repulsive forces.
   e. If the resulting node is collision-free, add it to the tree and create an edge from the nearest node.
   f. Attempt to connect the two trees if the distance between them is within a threshold.

5. Extract the path by backtracking from the connection point and optimize it using Dijkstra's algorithm.

6. Smooth the path using B-splines.

7. Output the final smoothed path and its length.

In the steps mentioned above, the steering part is important since, this is where, the attractive and repulsive forces are calculated and utilized to guide the robot towards the goal and avoid the obstacles.
In our code, we have a steer function that does the 'steering' for the point robot. Following is an explanation of the steer function
The inputs to the steer function are start position, goal position, step size, obstacles, attractive force constant, repulsive force constant, obstacle radius of influence

1. First, we calculate the direction and distance from the current position (start) to the target position (goal).

2. Then, ff the distance is less than the step_size, set the new point directly to the goal position.

3. Otherwise, calculate the attractive force towards the goal using the 'calculate_attractive_force function'. This function basically has the equation discussed above, which calculates the attractive force.

4. Calculate the repulsive force from obstacles using the 'calculate_repulsive_force' function. This function has the equation discussed above, which calculates the repulsive force.

5. Combine the attractive and repulsive forces to obtain the net force acting on the robot.

6. Normalize the net force to ensure it's a unit vector. This is important, since normalizing helps us emphasize more on the direction of the force vector rather than magnitude.

7. Calculate the new point by applying the net force to the current position, scaled by the step_size.

8. Clip the new point to ensure it stays within the workspace boundaries.

We ran the algorithm in Python and visualized it in 2D map of dimensions 50,30 for a point robot. The map had multiple obstacles(mostly circles and rectangles). The trees from the start node and the goal node were visualized using red and blue respectively. The path was visualized in green. The visualization was shown using Matplotlib. Following are the libraries we used: Numpy, matplolib, shapely, time, heapq, scipy.

The primary operations in the RRT Connect algorithm are generating random nodes, finding the nearest neighbor, calculating the new state, checking for collisions, and extracting the path. The generation of random nodes and calculation of the new state are constant-time operations,

O(1). Finding the nearest neighbor involves searching through the list of nodes, which has a time complexity of O(n), where n is the number of nodes in the tree. Collision checks involve geometric calculations that are O(1) per check but can be O(k) for all checks, where k is the number of obstacle vertices or circles. Extracting the path involves traversing the tree, which is O(m), where m is the number of nodes in the path. Therefore, the overall time complexity of the code is O(nki), where n is the number of nodes in the tree, k is the number of obstacles, i is the number of iterations

The algorithm stores nodes in two lists, contributing to a space complexity of O(n), where n is the number of nodes generated during the execution of the algorithm. The environment and obstacles are stored in static structures and do not grow with the size of the input, so their space complexity is O(1).The path extracted from the trees is stored in memory, contributing to the space complexity. The size of the path is proportional to the number of nodes in the path, so the space complexity for the path is O(m), where m is the number of nodes in the path.Therefore, the overall space complexity of the code is O(n + m), where n is the number of nodes and m is the number of nodes in the path.

IV. Results

Now, we ran the algorithm for 4 combinations of test cases each 50 times and simultaneously compared it with regular RRT Connect algorithm. Following are the test cases, we ran:

A. Test Case 1

Parameters for Improved RRT Connect Algorithm:

1. Start Position : 2,2
2. Goal Position: 48,24
3. Attractive force constant (C): 150
4. Repulsive force constant (K): 80
5. Radius of influence (R): 20

Parameters for Regular RRT Connect Algorithm:

1. Start Position : 2,2
2. Goal Position: 48,24

A. Test Case 2

Parameters for Improved RRT Connect Algorithm:

1. Start Position : 2,2
2. Goal Position: 48,24
3. Attractive force constant (C): 300
4. Repulsive force constant (K): 50
5. Radius of influence (R): 10

Parameters for Regular RRT Connect Algorithm:

1. Start Position : 2,2
2. Goal Position: 48,24

A. Test Case 3

Parameters for Improved RRT Connect Algorithm:

1. Start Position : 24,5
2. Goal Position: 4,24
3. Attractive force constant (C): 150
4. Repulsive force constant (K): 80
5. Radius of influence (R): 20

Parameters for Regular RRT Connect Algorithm:

1. Start Position : 24,5
2. Goal Position: 4,24

A. Test Case 4

Parameters for Improved RRT Connect Algorithm:

1. Start Position : 24,5
2. Goal Position: 4,24
3. Attractive force constant (C): 300
4. Repulsive force constant (K): 50
5. Radius of influence (R): 10

Parameters for Regular RRT Connect Algorithm:

1. Start Position : 24,5
2. Goal Position: 4,24

Now, when the parameters were, C= 150, K= 80, R= 10, the runtime of the code for the Improved RRTConnect algorithm was much faster than the regular RRTConnect algorithm, but the path length genertaed was longer. But, when the, parameters were, C= 300, K= 50, R= 10, the runtime was faster and the path length of the Improved RRT Connect algorithm was shorter than regular RRTConnect algorithm. We realized that when the attractive force constant (c) is increased and the repulsive force constant(k) and the radius of influence (R) is decreased, the algorithm focuses or rather APF makes the algorithm prefer 'reaching the goal' over 'avoiding the obstacles'. Now this doesnt mean that the robot will not consider the obstacles and collide with it. It will still avoid the obstacle, but, it while generating the path, the algorithm may make the robot go closer to the obstacles than before. Also, by reducing the radius of influence of the obstacles, the algorithm, considers the obstacles, only when it goes closer to it and does not plan the path considering the obstacles from afar. These things do make the path shorter.

We also observed that initially, the path generated was jittery and not smooth enough. We realized that we had to normalize the repulsive force and the net force vector.

Normalizing the repulsive force ensured that its magnitude doesn't vary significantly based on the distance to the obstacles. This prevents sudden changes in direction or magnitude of the repulsive force as the robot moves closer to or farther away from obstacles. Without normalization, the repulsive force became excessively large or small depending on the distance to the nearest obstacle, leading to erratic behavior of the robot.

Normalizing the net force ensured that the overall magnitude and direction of the force applied to the robot are consistent.

Normalization prevented the net force from becoming disproportionately influenced by either the attractive or repulsive components, maintaining a balanced effect on the robot's movement. Without normalization, the net force was dominated by either the attractive or repulsive force, causing the robot to either move too aggressively towards the goal or get excessively deflected by obstacles.

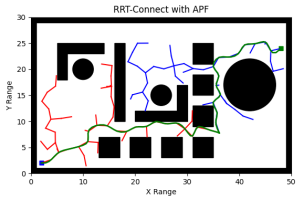

Fig. 8. Improved RRT Connect Test Case 1

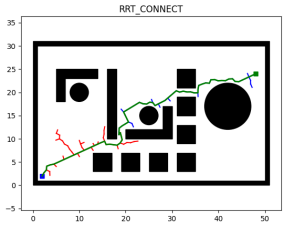

Fig. 9. RRT Connect Test Case 1

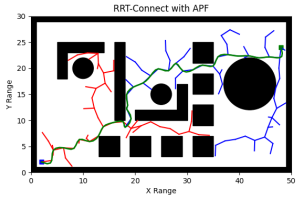

Fig. 10. Improved RRT Connect Test Case 2

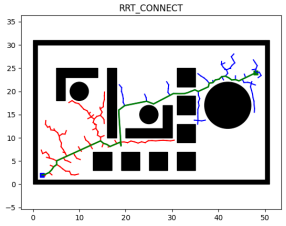

Fig. 11. RRT Connect Test Case 2

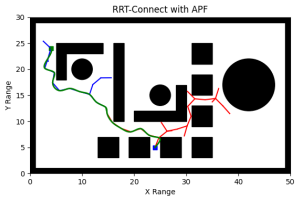

Fig. 12. Improved Connect Test Case 3

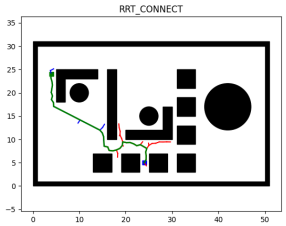

Fig. 13. RRT Connect Test Case 3

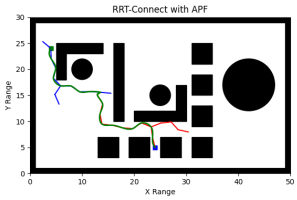

Fig. 14. Improved RRT Connect Test Case 4

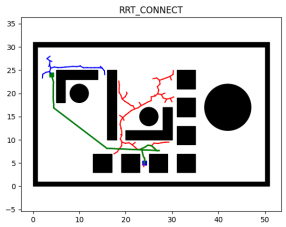

Fig. 15. RRT Connect Test Case 4

Now, from the above figures, intuitively, it may seem that there isn't much difference in the paths generated by both the algorithms. But after running multiple iterations, we calculated the average path and time for all the test cases and found that Improved RRT Connect is faster and provides shorter paths than regular RRT Connect in Test Case 2 and Test Case 4. From the figures it is evident that the path generated by Improved RRT Connect algorithm are smooth, meaning that the BSpline Curve applied is effective

Algorithm	Mean Time	Average Path Length
RRT Connect Test Case1	4.796215	63.31558
Improved RRT Connect Test Case1	0.540016	69.46378
RRT Connect Test Case2	5.225027	77.20452
Improved RRT Connect Test Case2	0.531298	69.66029

TABLE I
ALGORITHM COMPARISON FOR TEST CASE 1 AND 2

Algorithm	Mean Time	Average Path Length
RRT Connect Test Case3	4.034317	39.84235
Improved RRT Connect Test Case3	0.44786	41.46891
RRT Connect Test Case4	3.148066	59.38731
Improved RRT Connect Test Case4	0.44781	40.44611

TABLE II
ALGORITHM COMPARISON FOR TEST CASE 3 AND 4

From the above tables, it is evident, that the Test case 2 and 4 provide shorter paths. Following is a link to the excel sheets with the results data: https://docs.google.com/spreadsheets/d/1XFN_5F_hMVApUQJqxsB1MOzVVqy-a-vKM02LxpWUA0w/edit?usp=sharing

The time and space complexity for the code is mentioned at the end of the methodology section

V. CONCLUSION

Thus, we were able to achieve our goals for the project which involved implementing the 'Improved RRT Connect Algorithm' mentioned in the paper and validating the results. We were able to apply Artificial Potential Fields to drive the tree growth towards the target, optimize the path using Dijkstra's algorithm and get smoother paths using BSpline. After getting the result, we found that the algoritm mentioned in the paper, does work and we can get shorter paths in much less time by fusing APF with RRT Connect.

REFERENCES

[1] LaValle, Steven M.. "Rapidly-exploring random trees : a new tool for path planning." The annual research report (1998): n. pag.

[2] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), San Francisco, CA, USA, 2000, pp. 995-1001 vol.2, doi: 10.1109/ROBOT.2000.844730.

[3] Y. Zhu, Y. Tang, Y. Zhang and Y. Huang, "Path Planning of Manipulator Based on Improved RRT-Connect Algorithm," 2021 2nd International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE), Zhuhai, China, 2021, pp. 44-47, doi: 10.1109/ICBASE53849.2021.00016.

[4] Ahuja, R.K., Magnanti, T.L. and Orlin, J.B. (1993), Network Flow Theory, Algorithms, and Applications, Prentice-Hall, EnglewoodCliffs, NJ.

[5] Koch, Per & Kesheng, Wang. (1988). Introduction of b-splines to trajectory planning for robot manipulators. Modeling, Identification and Control. 9. 10.4173/mic.1988.2.2.

[6] Khatib, O. (1986). The Potential Field Approach And Operational Space Formulation In Robot Control. In: Narendra, K.S. (eds) Adaptive and Learning Systems. Springer, Boston, MA. https://doi.org/10.1007/978-1-4757-1895-9_26

[7] Hosseini Rostami, Seyyed Mohammad & Kumar, Arun & Wang, Jin & Liu, Xiaozhu. (2019). Obstacle avoidance of mobile robots using modified artificial potential field algorithm. EURASIP Journal on Wireless Communications and Networking. 2019. 10.1186/s13638-019-1396-2.

[8] Yang, Jixiang & Chen, Youping & Chen, Yuanhao & Zhang, Dailin. (2015). A tool path generation and contour error estimation method for four-axis serial machines. Mechatronics. 10.1016/j.mechatronics.2015.03.001.

[9] http://www.kuffner.org/james/plan/algorithm.php

[10] https://algodaily.com/lessons/an-illustrated-guide-to-dijkstras-algorithm