Deep Learning rooted Potential piloted RRT* for expeditious Path Planning

Snehal Reddy Koukuntla[*][†]
ksnehalreddy@iitkgp.ac.in
Indian Institute of Technology
Kharagpur, West Bengal

Manjunath Bhat[*]
manjunathbhat9920@iitkgp.ac.in
Indian Institute of Technology
Khargapur, West Bengal, India

Shamin Aggarwal[*]
shamin@iitkgp.ac.in
Indian Institute of Technology
Kharagpur, West Bengal, India

Rajat Kumar Jenamani[*]
rkj@iitkgp.ac.in
Indian Institute of Technology
Kharagpur, West Bengal, India

Jayanta Mukhopadhyay
jay@cse.iitkgp.ac.in
Indian Institute of Technology
Kharagpur, West Bengal, India

ABSTRACT

Randomised sampling-based algorithms such as RRT and RRT* have widespread use in path planning, but they tend to take a considerable amount of time and space to converge towards the destination. RRT* with artificial potential field (RRT*-APF) is a novel solution to pilot the RRT* sampling towards the destination and away from the obstacles, thus leading to faster convergence. But the ideal potential function varies from one configuration space to another and different sections within a single configuration space as well. Finding the potential function for each section for every configuration space is a grueling task. In this paper, we divide the 2 dimensional configuration space into multiple regions and propose a deep learning based approach in the form of a custom feedforward neural network to tune the sensitive parameters, upon which the potential function depends. These parameters act as a heuristic and pilots the tree towards the destination, which has a substantial effect on both the rate of convergence and path length. Our algorithm, DL-P-RRT* has shown the ability to learn and emulate the shortest path and converges much faster than the current random sampling algorithms as well as deterministic path planning algorithms. So, this algorithm can be used effectively in environments where the path planner is called multiple times, which is typical to events such as Robo-Soccer.

[*]The four authors contributed equally to this research.
[†]Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise or republish, to post on servers or to redistribute to lists requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org. CACRE'19 July 19-21, 2019, Shenzhen, China. © 2019 Association for Computing Machinery. ACM-ISBN 978-1-4503-7186-5/19/07...$15.00
https://doi.org/10.1145/3351917.3351990

KEYWORDS

Path planning, Deep Learning, Robotics, RRT, Potential energy

1 INTRODUCTION

Mobile automated bots require rapid motion in a highly dynamic environment where agents try to excel in their task with superior path planning and strategy. To achieve the former we need to design path planning algorithms that generate paths that are sufficiently short, and in minimal amount of computational time and space. Traditional graph algorithms like Dijkstra and A* [1, 2] give the very short path, but are computationally heavy and hence cannot be used in the fields of robotics which demand instantaneous results. On the other hand random sampling algorithms[3, 4] like Rapidly Exploring Random Trees (RRT) give stretched out, non-optimal paths and hence are not very feasible. RRT* solves this problem partially, but generally reduces the rate of convergence towards the goal.

Artificial Potential Field (APF) [5] is an interesting and intuitive concept that solves the above problems, by biasing the random sampling towards the destination and away from the obstacles by inducing artificial 'potential' fields. But traditional APF requires us to go over the wearying task of manually adjusting the extremely sensitive potential functions. Small changes in the potential functions can result in considerable change in both length of path generated and the rate of convergence and may also lead to the tree getting stuck in local minima and as a result it may never converge to the global minimum.

The rest of the paper is organized as follows . Section 2 presents several extensions made to RRT. Section 3 describes the problem and the objective. Section 4 presents our approach to solve the problem and an in-depth description of our algorithm. Section 5 showcases the effectiveness of our algorithm in comparison to other path-planning algorithms.

In Section 6 we discuss the further modifications which can be done to our model to improve it.

2 RELATED WORK

2.1 Rapidly-exploring Random Tree

This revolutionary work was introduced in [6]. Consider a general configuration space, C (a general state space that might include position and velocity information). An RRT that is rooted at a configuration q_{init} and has N vertices is constructed.

Algorithm 1 RRT algorithm

```
1:  procedure BUILD_RRT($q_{init}$, $N$, $\Delta q$ )
2:      $G.init(q_{init})$
3:      for $i = 1, 2, ..., N$ do
4:          $q_{rand} \leftarrow rand\_conf()$;
5:          $q_{near} \leftarrow nearest\_vertex(q_{rand}, G)$;
6:          $q_{new} \leftarrow new\_conf(q_{near}, \Delta q)$;
7:          if $Obstaclefree(q_{near}, q_{new})$ then
8:              $G.add\_vertex(q_{new})$;
9:              $G.add\_edge(q_{near}, q_{new})$;
10:     return $G$
```

Step 2 initializes a RRT, G, with the configuration q_{init}. G is a graph with the configurations $G.V$ as the set of vertices and $G.E$ as the set of edges. Step 4 chooses a random configuration, q_{rand}, in C. Alternatively, one could replace $rand\_conf$ with $rand\_free\_conf$, and sample configurations in C_{free} (by using a collision detection algorithm to reject samples in C_{obs}). It is assumed that a metric ρ has been defined over C. Step 5 selects the vertex, q_{near}, in G, that is closest to q_{rand}.

In Step 6 of $BUILD\_RRT$, new_conf selects a new configuration, q_{new}, by moving from q_{near} an incremental distance, Δq, in the direction of q_{rand}. This assumes that motion in any direction is possible. If differential constraints exist, then inputs are applied to the corresponding control system, and new configurations are obtained by numerical integration. Finally, a new vertex, q_{new} and a new edge, between q_{near} and q_{new}, are added to G.

Algorithm 2 NEAREST_VERTEX

```
1:  procedure NEAREST_VERTEX($q$, $G$)
2:      $d \leftarrow \infty$;
3:      for $v \in G.V$ do
4:          if $\rho(q, v) \leq d$ then
5:              $v_{new} = v$;
6:              $d = \rho(q, v)$;
7:      return $v_{new}$
```

2.2 RRT*

RRT*[7] inherits all the properties of RRT and works similar to RRT. However, it introduces two promising features called near neighbor search and rewiring tree operations. The near neighbor operation finds the best parent node for the new node before its insertion in the tree. This process is performed within the area of a ball of radius defined by :

$$k = \gamma(\frac{log(n)}{n})^{\frac{1}{d}} \tag{1}$$

where d is the search space dimension and γ is the planning constant based on environment. Rewiring operation rebuilds the tree within this area of radius k to maintain the sub-tree with minimal cost between tree connections.

Algorithm 3 RRT* algorithm

```
1:  procedure BUILD_RRT*($q_{init}$, $N$, $\Delta q$, $k$)
2:      $G.init(q_{init})$
3:      for $i = 1, 2, ..., N$ do
4:          $q_{rand} \leftarrow rand\_conf()$;
5:          $q_{near} \leftarrow nearest\_vertex(q_{rand}, G)$;
6:          $q_{new} \leftarrow new\_conf(q_{near}, \Delta q)$;
7:          $q_{min} \leftarrow ChooseParent(q_{new}, k)$;
8:          if $Obstaclefree(q_{min}, q_{new})$ then
9:              $G.add\_vertex(q_{new})$;
10:             $G.add\_edge(q_{min}, q_{new})$;
11:             $G \leftarrow Rewire(q_{new}, k)$;
12:     return $G$
```

2.3 RT-RRT*

RT-RRT*[8] initializes the tree with x_0 as its root. At each iteration, it expands and rewires the tree for a limited user-defined time (lines 5-6). Then it plans a path from the current tree root for a limited user-defined amount of steps. The planned path is a set of nodes starting from the tree root, (x_0,x_1,...,x_k). At each iteration it moves the agent for a limited time to keep it close to the tree root, x_0. When the path planning is done and the agent is at the tree root, it changes the tree root to the next immediate node after x_0 in the planned path, x_1. Hence, the agent moves on the planned path through the tree, towards the goal.

2.4 Potential piloted-RRT*

Potential Based Path Planning methods[9–13] treat the environment as a potential field such that the goal attracts and the obstacles repulse the agent. These methods stem from the original Artificial Potential Field (APF) guided directional-RRT* introduced in [14]. The relative attraction of the tree towards the goal in comparison to the repulsion away from the obstacles is an important parameter that determines the quality of the path generated both in terms of the time of convergence and the width of the tree. This effect is explained in section 4.2.

Algorithm 4 Potential Guided Directional-RRT* algorithm

```
1: procedure PGD-RRT*($q_{init}$, $N$, $\Delta q$, $k$)
2:     $G.init(q_{init})$
3:     for $i = 1, 2, ..., N$ do
4:         $q_{rand} \leftarrow rand\_conf()$;
5:         $q_{rand} \leftarrow GradientDescent(q_{rand})$;
6:         $q_{near} \leftarrow nearest\_vertex(q_{rand}, G)$;
7:         $q_{new} \leftarrow new\_conf(q_{near}, \Delta q)$;
8:         $q_{min} \leftarrow ChooseParent(q_{new}, k)$;
9:         if $Obstaclefree(q_{min}, q_{new})$ then
10:            $G.add\_vertex(q_{new})$;
11:            $G.add\_edge(q_{min}, q_{new})$;
12:            $G \leftarrow Rewire(q_{new}, k)$;
13:    return $G$
```

3 PROBLEM FORMULATION

The overall goal is to learn a policy that predicts high quality feasible paths in terms of lesser tree branch-out, length of path and convergence time. Algorithms such as Dijkstra's algorithm take too much time to give us the path, although they give the shortest possible path. In environments where frequency of re-computation of path is high, this is not suitable. So we need to get a faster computation version of Dijkstra's algorithm. Our model seeks to achieve this by optimizing the potential field guiding the random sampling. Let X denote a 2 dimensional configuration space. Let X_{obs} be the portion in collision and $X_{free} = X \backslash X_{obs}$ denote the free space. Let a path $\xi : [0, 1] \rightarrow X$ be a continuous mapping from index to configurations. A path ξ is said to be collision free if $\xi(\tau) \epsilon X_{free}$ for all $\tau \epsilon$ [0, 1]. Let c(ξ) be a cost function that maps ξ to bounded non negative cost [0, c_{max}]. The problem statement requires us to find a feasible path possible in minimum time and then converge upon the feasible path with the minimum cost. We do this by dividing the field into grids of predefined size and then predicting the ideal parameters of the potential-piloted planning algorithm at the centre of these grids using machine learning. Thereafter, any random point generated in the configuration space uses the predicted parameters of the nearest grid centre for the potential-piloted planning algorithm.

4 ALGORITHM

The previous works describe various ways in which the RRTs have been modified to improve the optimality, speed and dynamism of path generation. The potential piloted RRT*

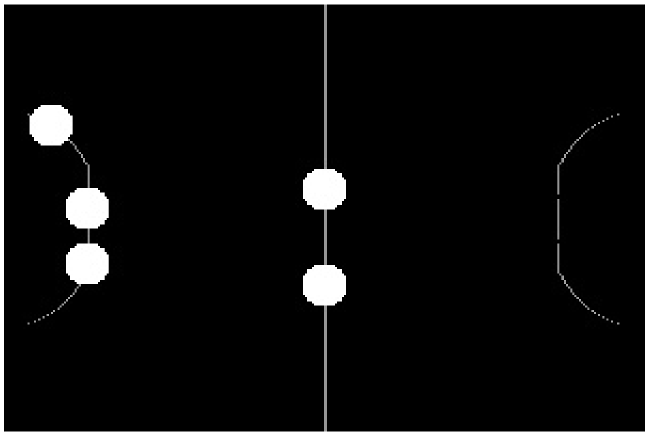


Figure 1: The field we considered for testing the algorithm

promises more optimal paths, but is not feasible as the potential parameters have different values at different parts of the configuration space and parameters determining the relative attraction towards goal and repulsion away from obstacles vary from field to field and have to be tuned laboriously for each field.

To overcome this, the image is divided into a number of grids, and a learning based algorithm is proposed, which automates the tuning process and finds the ideal parameters for each grid using the trained model quickly.

4.1 Field Area:

For testing and running this algorithm, we modelled our configuration space, X, on the field of the RoboCup Small Sized League. Here the bots are considered as obstacles, X_{obs}, and the rest of the field as free space, X_{free}. A popular defending tactic used by teams in RoboCup is considered, in which two bots tend to stay together in the D-Box to prevent direct shots towards the goal, one bot covers the opponent bot nearest to goal and meanwhile the other two bots stay near the halfline. Thus, the test field consists of three bots on the edge of the D-Box and two bots on the half-line as shown in Fig 1. The goal point is set to the centre of the goal line, which usually is the case in RoboCup SSL, while the start point is random. An example of the configuration space is shown in Figure 1. This space can be easily extrapolated to other use cases. We divided the configuration space into 12 (4*3) grids to inculcate the fact that potential function varies from one part of the configuration space to another.

4.2 Potential field constants:

Electrostatic field is a very coarse-grained approximation of the potential field under consideration. Considering sampled nodes to be carrying a negative charge, it is clearly evident that the destination should be made to act like a positive

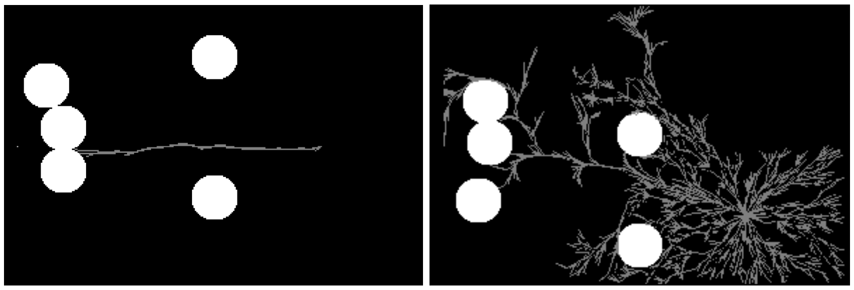
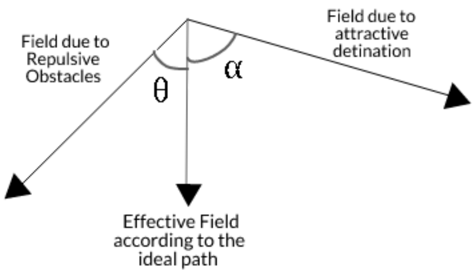


Figure 2: The tree growth and piloted sampling of the nodes in RRT-Star with APF for random values of Potential Function. The figures show the importance of tuning the parameters failing which the tree either gets stuck or needs generation of large number of nodes to find a feasible path.


Field due to Repulsive Obstacles
Field due to attractive detination
θ
α
Effective Field according to the ideal path
Figure 3: Vector representation of fields

charge to attract the tree to grow towards it. But since the obstacles should be avoided, they should be made to act like negative charges and repel the tree. Let $x_{obs} \epsilon X_{obs}$ be an obstacle node, carrying 'charge' $Q_{x_{obs}}$. Thus, at every point $x_{free} \epsilon X_{free}$, the potential due to x_{obs} is calculated as $v_{obs} = \frac{k.Q_{x_{obs}}}{\rho(x_{free}, x_{obs})}$ for some proportionality constant k. Similarly, the potential due to destination x_{dest} would be $v_{dest} = \frac{k'.Q_{x_{dest}}}{\rho(x_{free}, x_{dest})}$.

Thus the net potential at a point x_{free} is given by :

$$V = \frac{k'.Q_{x_{dest}}}{\rho(x_{free}, x_{dest})} + \sum_{x_{obs} \epsilon X_{obs}} \frac{k.Q_{x_{obs}}}{\rho(x_{free}, x_{obs})} \tag{2}$$

Here k and k' are proportionality constants due to obstacles and destination respectively.

The expanse by which the tree grows towards the destination relative to growing away from the obstacles is dependant on k and k'. This is a very sensitive parameter and has serious implications on the final path. Extreme potential fields can lead to the tree getting stuck near the obstacles or the tree growing very dense. The images shown in Fig 2 illustrate this fact.

4.3 Ideal Potential Data Generation:

To find the ideal constants of the potential function, the field is divided into a fixed number grids of predefined size. The grid-centre is a good approximation for the rest of the points in the grid while computing the ideal potential function at every point in the field. The ideal path is found by using Dijkstra's Shortest Path Algorithm from each of the grid's centre to the final destination. Using this ideal path the value of ideal k is found as follows -

Let the magnitude of net repulsive field be $k \sum_{i=1}^{n} \frac{Q_i}{r_i}$ and the magnitude of attractive field be $\frac{Q'}{r'}$.

Let the angle between the repulsive field vector and the ideal vector obtained from the ideal path be θ and let the angle between attractive field and the ideal vector be α, as shown in figure 3.

Hence,

$$k \sum_{i=1}^{n} \frac{Q_i}{r_i} sin\theta = \frac{Q'}{r'} sin\alpha \tag{3}$$

$$k = \frac{\frac{Q'}{r'} sin\alpha}{\sum_{i=1}^{n} \frac{Q_i}{r_i} sin\theta} \tag{4}$$

4.4 Training :

The position of the obstacle is an important determinant of the RRT* Tree. So is the area of the obstacle along with positions of the source and the destinations. So the centres of the each obstacle blob, the area of the obstacles and the source

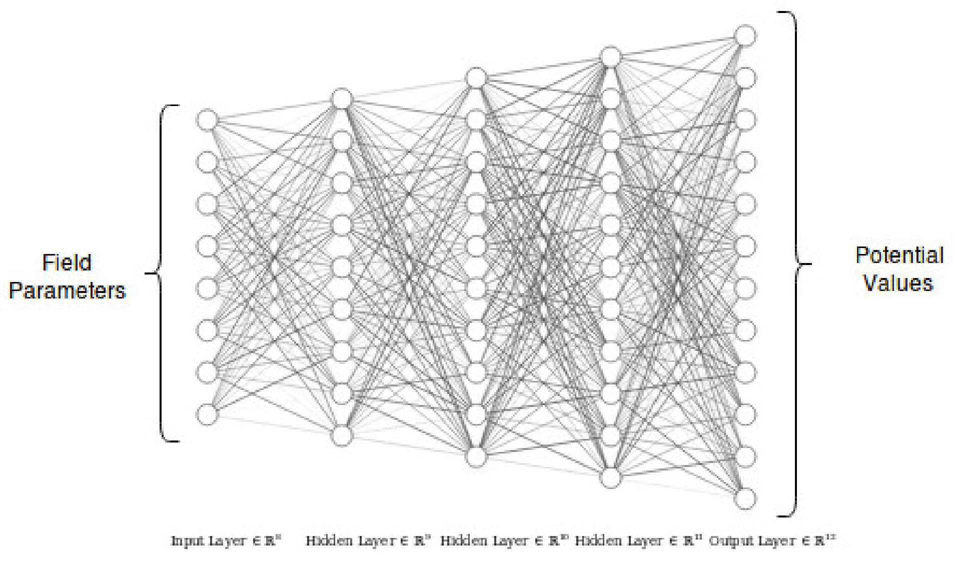


Figure 4: Neural Network neuron diagram with opacity of the edges proportional to the weights of the final trained model.

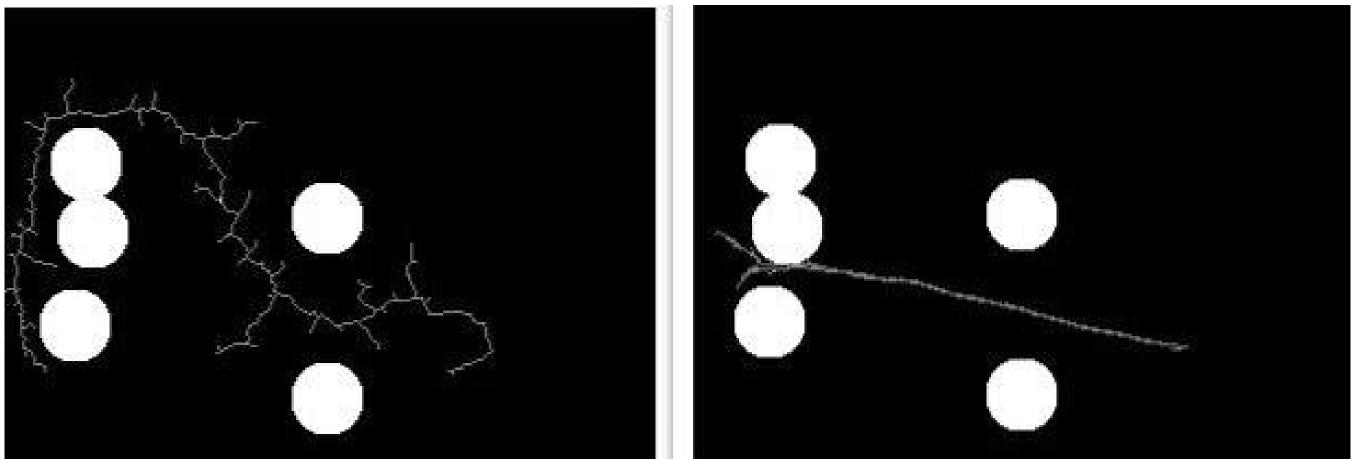


Figure 5: The image on the right shows the path predicted by our algorithm($DL - P - RRT^*$) and the image on the left shows the path predicted by RRT-Connect.

and destination coordinates were taken as the features of our neural network. These formed the 8 input nodes shown in Figure 4. The ideal potential functions were also fed to the neural network, which we acquired using the procedure explained in Section 4.2, into the neural network. These form the 12 output nodes, each corresponding to a specific region on the configuration space. Three hidden layers were used as shown in the Fig 4, with 9, 10 and 11 nodes respectively. 5-fold cross validation was used to tune the hyper-parameters. The SGD optimizer was used with Nesterov momentum. Clip-norm and l2 regularization was used to control the weights in the neural network. The importance of these optimisation techniques are mentioned in [15] The potential function thus predicted by the neural network was fed to run a potential guided RRT* tree and an example of the output tree is shown in Figure 5.

5 RESULTS AND COMPARISON:

5.1 Predicted Potentials:

The predicted potentials came out to be very close to the ideal ones and we plotted it with the help of OpenCV to help visualise the effect of them. As shown in the Fig. 6,

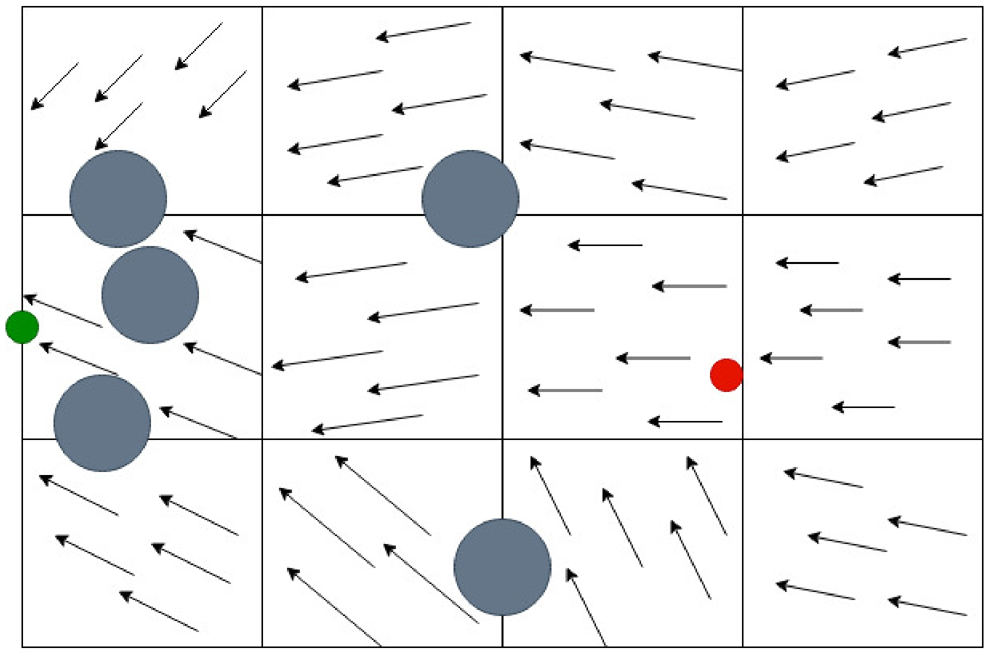


Figure 6: The potential at every section of the configuration space, with the length of the arrows proportional to the magnitude. The green circle indicate the destination and the red one, source. The grey circles indicate obstacles.

the potentials learnt by the neural network guide the path towards the destination and away from the obstacles in a clever fashion.

5.2 Time of convergence:

The RRT* tree with the predicted values of potential functions converges faster than normal RRT* tree. It is evident from the table given below, that final path is generated in much lesser time in comparison to Dijkstra's algorithm. The following table shows the comparison between

* Rapidly-exploring Random Trees Star (RRT*) algorithm as explained in Section 2.2,
* The Deep Learning rooted Potential piloted RRT* (DL-P-RRT*) tree with predicted potential functions,
* Dijkstra's algorithm which was used to generate the ideal path for learning the ideal path as discussed in Section 4.3,

with respect to the time of convergence, for the images given above respectively.

Time of Convergence (in seconds)			
S.No.	RRT*	DL-P-RRT*	Dijkstra
1	2.329800	0.986253	8.857095
2	1.998531	0.81104	9.103717
3	5.170486	0.978344	8.656968
4	2.336515	0.910896	8.927036
5	1.288411	0.238291	8.410355

5.3 Length of path:

The DL-P-RRT* algorithm has the ability to learn and emulate the shortest path generated by Dijkstra's algorithm and hence provides a much more shorter path than the RRT* algorithm. The following table shows the comparison between

* Rapidly-exploring Random Trees Star (RRT*) algorithm as explained in Section 2.2,
* The Deep Learning rooted Potential piloted RRT* (DL-P-RRT*) tree with predicted potential functions,
* Dijkstra's algorithm which was used to generate the ideal path for learning the ideal path as discussed in Section 4.3,

with respect to the length of the final path, for the images given above respectively. Length of the path is measured in terms of euclidean distance between pixels in accordance to the dimensions of the SSL field.

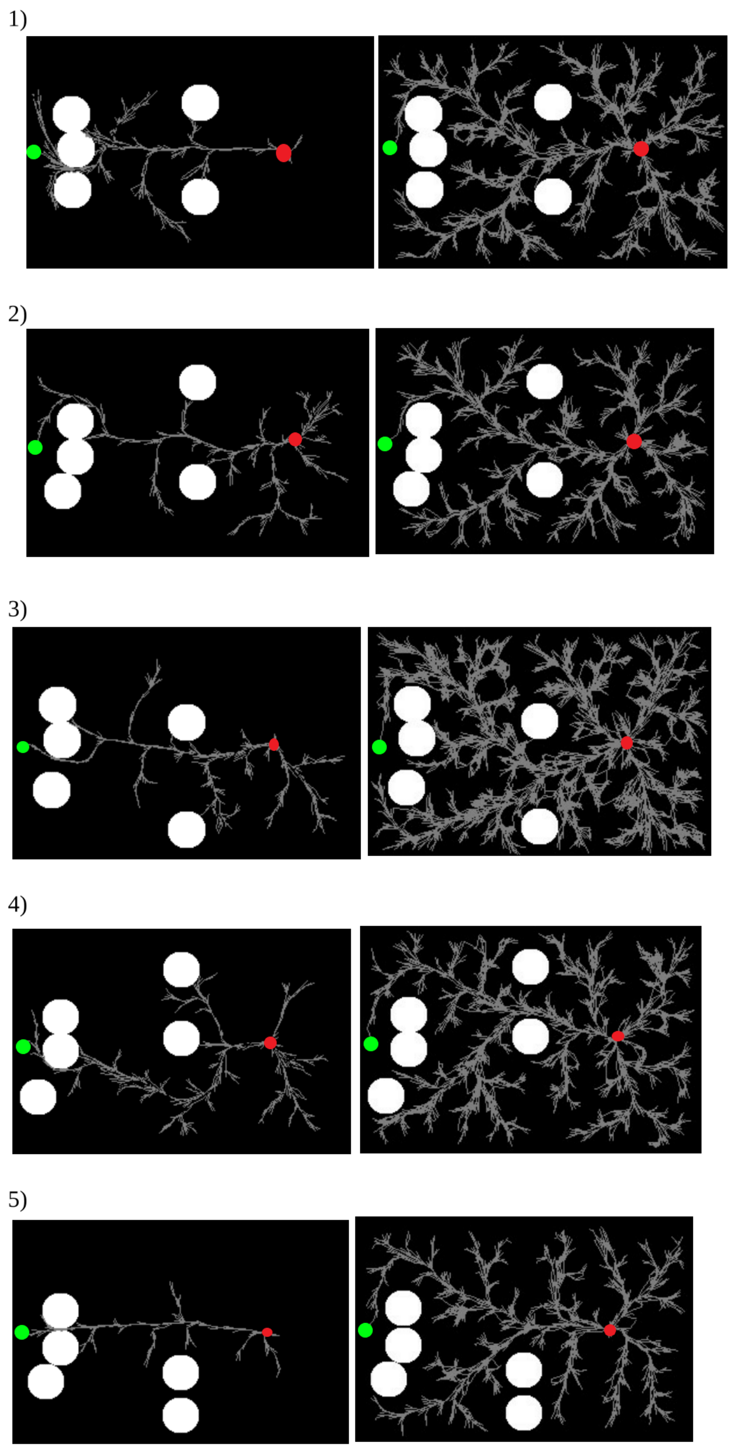


Figure 7: The images on the left are the paths formed with DL-P-RRT* and the ones on the right are the paths formed by RRT*.

Length of the final path			
S.No.	RRT*	DL-P-RRT*	Dijkstra
1	2733.29	2267.12	2207.77
2	2639.58	2597.48	2518.91
3	2971.05	2256.47	2198.63
4	2662.99	2697.83	2620.49
5	2679.56	2125.55	2090.69

5.4 Density of the tree:

The width of the tree also reduces significantly implying that the predicted values of potential functions are close to the ideal ones. The Figure. 7 above shows the comparison between RRT star and DL-P-RRT* with respect to the width of the tree. In each of the images the destination point is the centre of left edge.

Future Applications

This algorithm was trained for the configuration specific to Robo-Soccer[16] environment. The opponent bot obstacles can be replaced by whatever the robot should avoid, according to the scenario and so can the dimensions, source and destination. Thus this algorithm can be generalised to any path planning scenario. This algorithm requires computationally heavy training only once and that model can be used for the scenario, generating short paths in considerably lesser amount of time.

Acknowledgement

We sincerely thank Ashish Kumar (ashishkg0022@iitkgp.ac.in), Rahul Kumar Vernwal (vernwalrahul@iitkgp.ac.in) and Saurabh Agarwal (shourabhagarwal@iitkgp.ac.in) for assisting us in this project and supporting us as and when required.

REFERENCES

[1] Javaid, A.:Understanding Dijkstra Algorithm. In *SSRN Electronic Journal, 2013*

[2] Shrawan Kr. , Pal B.L. :Shortest Path Searching for Road Network using A* Algorithm. In *JCSMC, Vol. 4, Issue. 7, pg.513 to 522, July 2015*

[3] Arslan, O, Berntorp, K., Tsiotras, P.:Sampling-Based Robot Motion Planning: A Review. In *2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, pp. 4991 to 4996, 2017.*

[4] Arslan, O., Tsiotras P.: Machine Learning Guided Exploration for Sampling-based Motion Planning Algorithms . In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)Congress Center Hamburg, Sept 28 - Oct 2, 2015. Hamburg, Germany*

[5] Bin-qiang, Y., Mingfu, Z., Wang, Y.:Research of path planning method for mobile robot based on artificial potential field. In *International Conference on Multimedia Technology, Hangzhou, 2011*

[6] Steven M. LaVelle: Rapidly Exploring Random Trees: A new tool for path planning. In: RoboCup 2018. In *1998*

[7] Noreem, I., Khan, A., Habib, Z.:Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions. In *International Journal of Advanced Computer Science and Applications,Vol. 7, No. 11, 2016*

[8] Naderi, K., Rajamaki, J, Hamalainen, P.:RT-RRT*: a real-time path planning algorithm based on RRT*. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games at Paris,France. November 16 to 18, 2015*

[9] Qureshi, A.H, Ayaz, Y.:Potential Functions based Sampling Heuristic For Optimal Path Planning. In *Springer Autonomous Robots, Volume 40, August 2016*

[10] Agarwal, S., Gaurav, A.K., Nirala, M.K., Sinha, S.:Potential and Sampling based RRT* for Real-Time Dynamic Motion Planning Accounting for Momentum in Cost Function. In *International Conference on Neural Information Processing, 2018*

[11] Weerakoon, T.:An Artificial Potential Field Based Mobile Robot Navigation Method To Prevent From Deadlock. In *Journal of Artificial Intelligence and Soft Computing Research, 2015*

[12] Kadir F.: A study on Artificial Potential Fields. In *2011*

[13] Xie L., Chen H., Xie G.: Artificial Potential Field Based Path Planning for Mobile Robots Using Virtual Water-Flow Method. In *Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques, ICIC 2007*

[14] Khatib, O.:Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. In *IEEE International Conference on Robotics and Automation, 1985*

[15] Ruder, S.:An overview of gradient descent optimization algorithms. In *2016*

[16] Bhushan, M., Agarwal, S., Gaurav, A.K., Nirala, M.K., Sinha, S., et al.: KgpKubs 2018 Team Description Paper. In: RoboCup 2018. In *RoboCup Symposium, 2018*