Sampling based Planning Algorithm for Autonomous Navigation

Amith Ramdas Achari, University of Illinois, Urbana Champaign, *amithr3@illinois.edu*

***Abstract*- Recently, sampling-based algorithms like the Rapidly Exploring Random Tree (RRT) have been proposed as a viable solution to the computationally difficult motion planning problem. While the RRT algorithm is known to find a feasible solution quickly, there are no guarantees about the quality of that solution, for example, in terms of a given cost functional. To overcome this limitation, the paper discusses a variant of RRT known RRT*, a new algorithm that ensures asymptotic optimality, or almost certain convergence of the algorithm's solution to an optimal solution, while maintaining the same properties as the standard RRT algorithm in terms of computation of feasible solutions and computational complexity. The RRT algorithm is extended to deal with differential constraints in this paper. It is demonstrated that the RRT* algorithm, when combined with any local steering procedure that meets this condition, almost always converges to an optimal solution. The results show how the algorithm's behaved with changes in hyperparameters, mostly bias and radius.**

I. Introduction

Autonomous vehicle research has entered a period of rapid growth in recent years. Autonomous vehicles are being tested on real city roads in increasing numbers. Some businesses have even started testing self-driving automobiles. In the passenger car sector, Google Waymo has established a self-driving taxi service in California. Autonomous vehicles use their sensing devices to perceive their surroundings, make executable safety decisions via the decision-making subsystem, confirm the behavior's effectiveness, plan its trajectory, and finally complete autonomous driving. Many autonomous vehicle technologies are derived from intelligent robots, which is a research branch of intelligent robots. It's difficult to navigate in a congested environment while maintaining maximum safety and task efficiency.

In the field of autonomous mobile robots, motion planning is a critical topic that allows them to move from one location to another in a variety of situations, including both static and dynamic obstacles, without the need for human interaction. Decision-making is a crucial component of autonomous driving. Decision-making examines if the car can accomplish various driving actions smoothly and accurately. It is extremely difficult to plan a route while driving due to the complexity of the real-world urban environment and the non-holonomic vehicle system (The vehicle is constrained from moving sideways).
The final trajectory optimization is a constrained optimization problem in which the constraints include the vehicle's current state (i.e., planned and beginning state) behavior and target vehicle dynamics. The final output trajectory can be based on the trajectory that fits these conditions. Autonomous driving technology has been developed as a promising technology throughout the last decade. Trajectory planning has made great progress as a major difficulty of autonomous driving technology. There are four types of algorithms for trajectory planning: interpolating-curve-based, graph-search-based, sampling-based, and numerical-optimization-based.

Rapidly exploring random tree (RRT) is the most popular sampling-based approach for trajectory planning. It can be solved quickly with convergence and completeness in a high-dimensional system. Using random samples from the search space, an RRT grows a tree rooted at the starting configuration. As each sample is drawn, a link is made between it and the state closest to it in the tree. This results in the addition of the new state to the tree if the connection is feasible (passes entirely through free space and obeys all constraints). However, the sampling-based approach has several drawbacks, and there is a lot of research being done to overcome it. One major drawback of RRT is that it does not result in optimal path. And RRT* is a great example of its variant, which guarantees asymptotic optimality. This algorithm can further be improved for different use case scenarios, and in this study, a variant of RRT* for autonomous navigation with non-holonomic constraints using motion-based primitives will be explored. This algorithm should be capable of generating human-like trajectories in day-to-day life.

II. Background

A key limitation of RRT* is that it is applicable only to systems with simple dynamics, as it relies on the ability to connect any pair of states with an optimal trajectory (e.g., holonomic robots, for which straight lines through the state space represent feasible motions). [2] The optimal RRT* is a version of the RRT algorithm that has the asymptotic optimality property, i.e., almost-sure convergence to an optimal solution. Which would mean that if a feasible path exists RRT* will return the best possible time as time approaches infinity. There have been several attempts to expand RRT*s applicability to kinodynamic systems where a straight-line link between two states is often not a valid trajectory due to the system's differential constraints. The approaches used by many researchers is to minimize the cost function to find the optimal path while satisfying the dynamics of the system.

The main contribution of [1] is the identification of a set of sufficient conditions that guarantee the RRT* algorithm's asymptotic optimality for systems with differential constraints. Locally controllable systems, particularly controllable linear systems, and locally optimal steering algorithms, satisfy these sufficient requirements. There are also simulation examples

involving systems with nonholonomic dynamics. The work includes a formal specification of the optimal kinodynamic motion planning issue as well as an RRT* algorithm that has been developed to deal with systems with differential constraints.

$$\dot{x}(t) = f(x(t), u(t)), \quad x(0) = x_0 \qquad (1)$$

The dynamics of the system in [1] are described by a smooth continuous differentiable function f, with x ∈ X as trajectories and u ∈ U as controls. Finding a control u such that the unique trajectory X satisfies equation 1 by reaching the goal region while avoiding the obstacle region is the kinodynamic motion planning problem. [1] presents the optimal kinodynamic motion planning problem, which is the solid kinodynamic motion planning problem with a cost function that is minimized. The author assumes that the cost function is the line integral of a lipschitz continuous function G, and then the author outlines the procedure that the algorithm relies on. Before giving the details of RRT*, he briefly describes sampling, distance function, nearest neighbor, adjacent vertices, local steering, collision check, which form the primitive functions in RRT algorithm, and then adds his idea of *extend* function to improve the trajectory generated. [1] claims that the results are consistent across a wide range of sampling strategies, however in the model uses uniform distribution as the sampling strategy. [1] presents a set of sufficient requirements for asymptotic optimality in systems with differential constraints and provides three dynamical systems as examples, all of which meet the author's assumptions. Dobin's car model, 2D double integrator, and simple airplane 3D model are used as examples. The simple 3D airplane model consists of a car model (containing x, y, theta) on the plane and a 2D double integrator for altitude dynamics, totaling 5 states. [1] compares the quality of solutions given by RRT and RRT* algorithms and it is clearly demonstrated in simulations. RRT*, on the other hand, is often able to deliver significant quality improvement with minimal computational work. [1] The RRT algorithm, on the other hand, is frequently stuck with a solution found and unable to improve it.

[2] This paper describes a Kinodynamic RRT*, which is an incremental sampling-based approach for asymptotically optimal motion planning for robots with linear differential constraints (for controllable linear dynamics). [2] The approach also works for nonlinear dynamics, as shown in the paper, which is accomplished by linearizing the system at a lower computational cost than RRT* for holonomic robots. [2] This is done by extending the well-known formulation for a fixed final state and fixed final time optimum control issue to construct an optimal, open loop, fixed final state free final time control policy. [2] The author formally defines the problem as a optimization problem, given a start state and end state, find the collision-free trajectory with minimal cost. The total cost is the concatenation of the optimal trajectories between a series of successive states found in the optimization step. [2] It should be noted that for the model proposed in [2] requires for the dynamics matrix A to be nilpotent after all linearization, so that it is possible to use numerical methods for computing connections between states. [2] However, this class of nilpotent systems that generate trajectories is computationally efficient, making optimal planning for kinodynamic systems even in high-dimensional state spaces computationally possible.

[3] proposes a method for applying Linear Quadratic Regulator (LQR) to the problem of determining optimal finite-horizon trajectories (determination of optimal trajectories for a finite interval) and costs in the setting of RRT*. For problems with affine dynamics and quadratic cost functions, this algorithm converges to the optimal plan with probability one. [3] augments time as a dimension to the space in which the tree grows, which is a typical way to solving issues in time-varying contexts. We may impose limitations on the length of time of the solutions since the search tree explicitly reflects state-time and examines all potential pathways in this space. As a result, the technique can be used to solve a wider range of problems. In general, these approximate dynamics are affine, meaning they contain a zero-order term. Because LQR is usually used with linear systems, [3] includes an extension to LQR that can be used with affine systems. The optimal kinodynamic motion planning issue aims to determine a trajectory that starts in a given state, ends in a desired region, avoids obstacles, and fulfills differential dynamic constraints given a deterministic system with known process dynamics. [3] employs finite-horizon LQR to address the challenge of finding a state and control trajectory, x and u, that minimizes the cost function given deterministic linear process dynamics. Non-linear systems are also well-suited to the model provided in the paper. [3] This is accomplished by making linear transformations in the augmented space behave as affine transformations in the original space by modifying the homogeneous coordinates in the augmented space. In the augmented system, the affine version of the issue is addressed using LQR in the standard way. The Q matrix in the cost function is penalized such that the LQR generates trajectories that terminate very close to the random sample. The most interesting part of the paper is the extension of the system to time horizon, where the optimality can be based on the distance and time. The trajectories generated are specific to constraints on the time horizon. Karaman's [1] assumption is also satisfied by the model provided in [3]. A 2D double integrator was used to evaluate the approach to affine systems. [3] The extension to non-linear system is done by linearizing the system about an operating point and LQR is called, which then is very similar to that used to solve the affine problem. The true cost function over the trajectory is integrated after the trajectory is found. [3] Similarly, the true cost of each of these trajectories is stored in the tree, even though the algorithm employs approximations to calculate trajectories from the sample to other vertices in the tree within the cost radius. As a result, the method rewires pathways based on their true execution costs rather than the approximation.

III. Algorithm

RRT generates graphs that are very cubic. As nodes are connected to their nearest neighbors, this is to be expected. The structural nature of these graphs makes finding an optimal path difficult. The two legs of a triangle are navigated across instead of taking the hypotenuse between two points. This is clearly a

greater distance. RRT* addresses the cubic nature and irregular paths generated by RRT.

RRT* is a more efficient version of RRT. RRT* has the same basic principle as RRT, but two key additions to the algorithm produce significantly different results. The following steps must be followed in order to implement RRT*; where the first three steps of the algorithm are identical to RRT as mentioned in [4]. The next step is to locate the graph's closest node, after which a neighborhood of nodes within a radius of the new nodes is investigated. Then there's the rewiring step, which involves rebuilding the tree within the specified radius in order to keep the tree as cost-effective as possible between tree connections.

The RRT* algorithm is built on the following primitive procedures:
Sampling (*RandomPosition*): Sampling procedure that returns independent and identically distributed samples from the obstacle-free space. There are a variety of strategies that can be used for sampling, but I would recommend using a uniform sampling distribution. because of the cost of generating the distribution.
Distance Function (*Distance*): This is a function that returns the optimal cost of the trajectory between two states assuming no obstacles.
Nearest Neighbor (*Nearest*): Given a graph G= (V, E) and state $z \in X$, the *Nearest* (G, z) returns the vertex $v \in V$ that is closes to z, according to the distance function.
Near-by-Vertices (FindNeighbors): Given a graph G= (V, E) and state $z \in X$, and a radius r, the *FindNeighbors* procedure returns all the vertices in V that are near z, where the closeness is parametrized by r.
Connect Nodes *(Chain)*: Given two nodes, it connects these two nodes which forms a link.
Local Steering *(Steer)*: Given two states z1, $z2 \in X$, the steer procedure returns the optimal trajectory in some local neighborhood, starting at z1 and ending at z2.
Collision Check *(CollisionFree)*: Given a trajectory σ: [0, T], the *collisionFree* procedure returns true iff x lies entirely in the obstacle-free space for all $t \in [0, T]$

Algorithm 1: LQR-RRT*
Rad = r
G (V, E) //Graph containing edges and vertices
for itr in range (0…n)
  Xnew = *RandomPosition* ()
  If *CollisionFree* (Xnew) == True, try again
    Xnearest = *Nearest* (G (V, E), Xnew)
    Xneighbors = *findNeighbors* (G (V, E), Xnew, Rad)
    //Select u_i to move from Xneighbors towards Xnew (From LQR)
    for x' in Xneighbors
      Xbest = Xneighbors(i) + [f (xi, ui) 1] $^{T}\Delta t$
      //Update Xnew based on ui
        If *CollisionFree* (Xbest) == True:
          Link = *Chain*(Xnew,Xbest)
          Parent(x') = Xnew
          G += {Xnew,x'}
          G += Link
Return G (V, E)

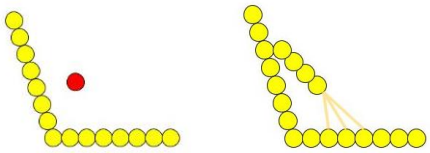


Figure 1. Rewiring Step, left: Red is the random node (Path generated by RRT, right: chain to the nearest node (Path generated by RRT*)

Firstly, RRT* keeps track of how far each vertex has traveled in relation to its parent vertex. This is referred to as the vertex's cost(). After locating the closest node in the graph, a neighborhood of vertices within a fixed radius of the new node is examined. If a node with a lower cost() than the proximal node is found, the cheaper node replaces the proximal node.
The second difference RRT* adds is the rewiring of the tree. After a vertex has been connected to the cheapest neighbor, the neighbors are again examined. Neighbors are checked if being rewired to the newly added vertex will make their cost decrease. If the cost does indeed decrease, the neighbor is rewired to the newly added vertex. This feature makes the path more smooth. The rewiring of the tree is the second difference RRT* introduces. The neighbors are examined once more after a vertex has been connected to the cheapest neighbor. Neighbors are examined to see if being rewired to the newly added vertex will lower their costs. The neighbor is rewired to the newly added vertex if the cost decreases. This feature improves the path's smoothness.

IV. Experiments

Python 3.8 was used to create an implementation of the algorithm. The algorithm has been implemented in such a way that it has a bias, and depending on the bias, it either explores or exploits knowing the goal's end position. By increasing the likelihood of sampling states from a specific area, RRT* growth can be biased. The majority of RRT* implementations use this to direct the search toward the goals. This is accomplished by modifying the state sampling procedure to include a small probability of sampling the goal. The higher this probability, the greedier the tree becomes in its pursuit of the goal. When the state reaches the goal, the algorithm is terminated. The radius r is another hyperparameter that can be changed. We could vary how long each step was by varying the radius r. The nodes explored by the algorithm and the final path chosen by backtracking the parent nodes were visualized using a pygame interface. The inputs to the algorithm were the start point and end point. The map also included randomly generated obstacles within the defined region. Based on the obstacles and the inputs provided the algorithm return a path. The visualization also conveys an understanding of how bias influences the states being explored.

A. Experimental Results

The parameter bias was varied for each randomly generated obstacle in a map, with a fixed start and end goal, and the results were collected. The states explored decreased as bias increased, and vice versa. However, the decrease in Path length was not excessive in relation to the increase in bias values. This is evident due to the rewire step, as the algorithm can find the

optimal path until it reaches a goal. Figure 1 shows the relationship between Explored States vs Bias and Path Length vs Bias.

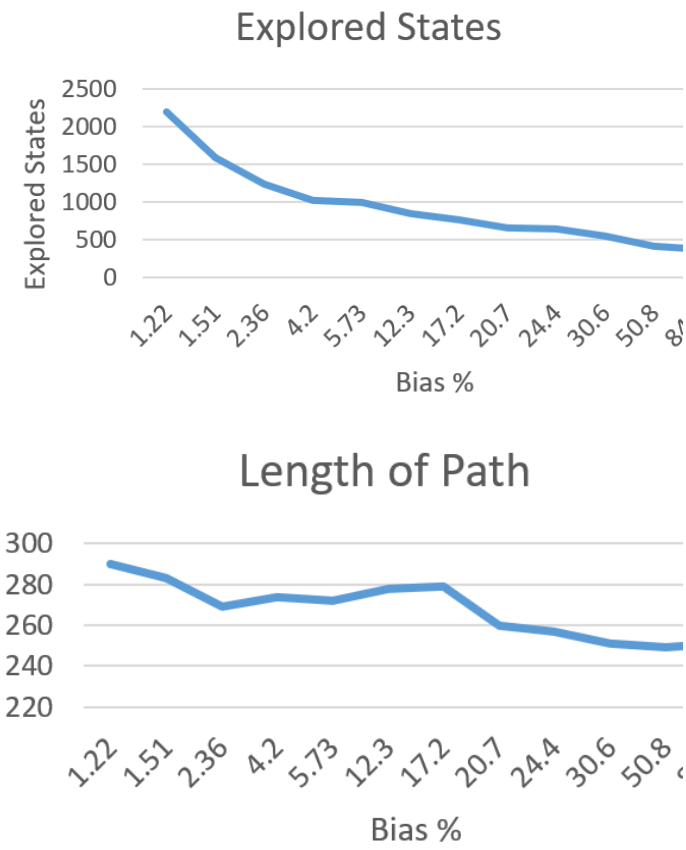
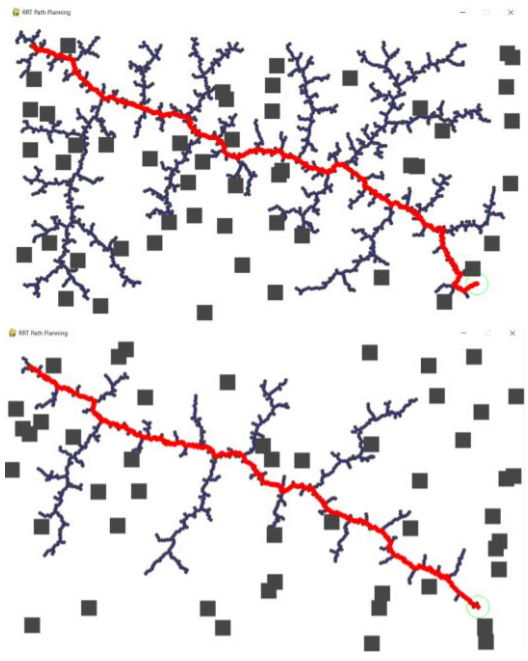


Figure 2. Top: Bias vs Explored States, Bottom: Bias vs Path Length



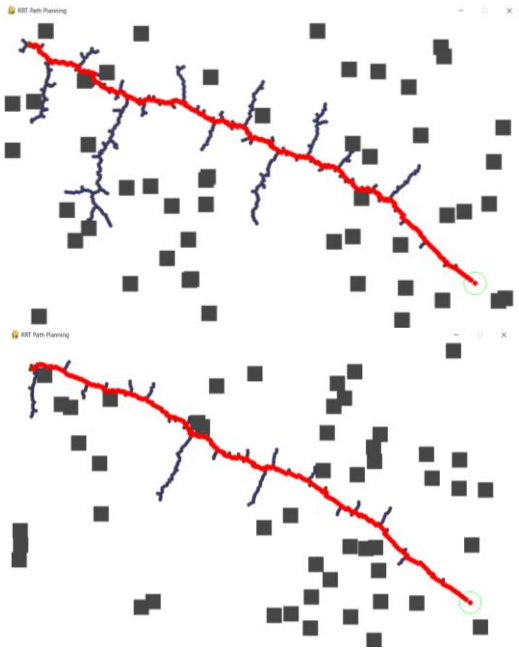
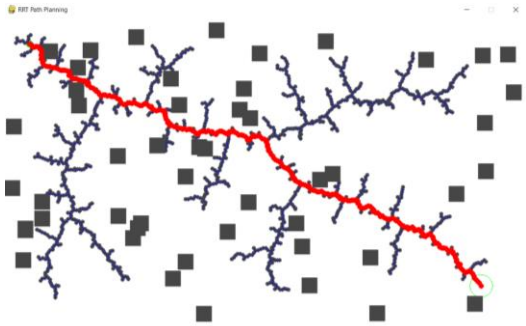


Figure 3. Graph generated for different values of Bias. First 1.22%, Second 5.73%, Third 20.7%, Fourth 50.8%.

Figure 2 shows that as the bias increases, the number of states explored decreases, and the number of states explored is reduced by nearly 80%. However, because bias always goes in the direction of the goal, we can't increase the value of bias as much as we want. So, if an obstacle exists between the goal and the current state, the algorithm will continue to try to reach the goal state without exploring other states, increasing the computation and the number of paths explored. A smaller bias value can be used to solve these types of problems.

The radius of the step is another parameter that was varied, and the results were collected. It is seen that with a larger step radius, the goal was reached faster, but the path generated was not smooth at all. As a result, the step radius was reduced in order to achieve a smoother path with more accurate representation of the way points. At a radius of 5 pixels, the compromise between accuracy and time for the state to reach its goal was found. The time taken for 5px and 25px was 3.35ms and 3.03ms, respectively, for a 6 percent bias. Figure 3 shows the graph generated for both pixel values.



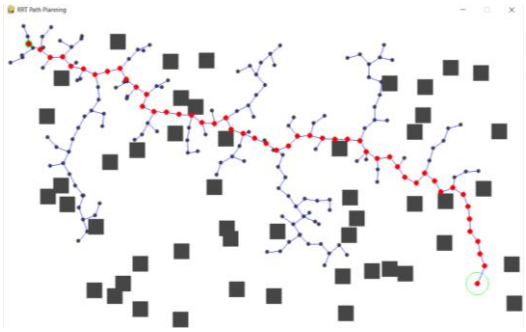


Figure 4. Graph generated for radius = 5px (top) and radius = 25px(bottom)

V. Discussion

According to the previous section's results, the algorithm can successfully handle the kinodynamic motion planning problem. Sufficient conditions on the system dynamics and the local steering function considered in the algorithm allowed us to guarantee asymptotic optimality. For 50 trials, it is always able to generate an optimal path for a randomly generated environment. It is also observed that with increase in bias towards the goal reduced the number of explored states, which infers that we could employ much better sampling strategies to yield better results. However, having large bias returned poor results when the goal region was blocked partially when compared to smaller bias, so a better sampling strategy such A*(grid-based planning algorithm) heuristic could be employed. Hence, this becomes difficult to ascertain a particular range of bias which can handle most motion planning algorithms. The sampling method could use stochastic techniques to infer distributions of samples that are likely to contribute to an optimal trajectory. Furthermore, the ability to connect any pair of states can be used to perform trajectory smoothing as a postprocessing step using iterative shortcutting. This could help to improve the quality of the solutions and provide more accurate estimates of the algorithm's convergence rate. For future work, we could adaptively change the radius in which it finds its neighbors based on the value the heuristic returns. Future work will include applying the outlined algorithm in this paper to high-dimensional dynamical systems, and practical demonstration on robotic platforms preferably on GEM (Autonomous Electric Vehicle) provided by the University of Illinois at Urbana Champaign.

VI. References

[1] S. Karaman and E. Frazzoli, “Incremental sampling-based algorithms for optimal motion planning,” Robotics: Science and Systems VI, 2010.

[2] Webb, Dustin J., and Jur Van den Berg. “Kinodynamic RRT*: Asymptotically Optimal Motion Planning for Robots with Linear Dynamics.” 2013 IEEE International Conference on Robotics and Automation, 2013, doi:10.1109/icra.2013.6631299.

[3] Goretkin, Gustavo, et al. “Optimal Sampling-Based Planning for Linear-Quadratic Kinodynamic Systems.” 2013 IEEE International Conference on Robotics and Automation, 2013, doi:10.1109/icra.2013.6630907.

[4] S. M. LaValle, “Sampling-based motion planning,” Planning Algorithms, pp. 153–205, 2006.

[5] F. Zhang, R. Xia, and X. Chen, “An optimal trajectory planning algorithm for autonomous trucks: Architecture, algorithm, and experiment,” International Journal of Advanced Robotic Systems, vol. 17, no. 2, p. 172988142090960, 2020.