Artificial Intelligence Mini Project

REPORT - 2

Team Members:

CS20B1100 - Anuj

CS20B1075 - Jitin

We have decided to use RRT based AI algorithms for the Path Finding problem. The main RRT based search algorithms are the following:

Rapidly exploring random tree (RRT):

The RRT (Rapidly-Exploring Random Tree) algorithm is a motion planning algorithm used to find feasible paths for a robot or a mobile agent to navigate through a workspace cluttered with obstacles. It is a probabilistically complete algorithm, which means that it can find a solution if one exists, given enough time and resources.

It is particularly useful when the agent needs to navigate through cluttered environments with obstacles that have irregular shapes and configurations. It is generally used in Robot Path Planning, Autonomous vehicle navigation, Virtual environment exploration and Motion planning for unmanned aerial vehicles (UAVs).

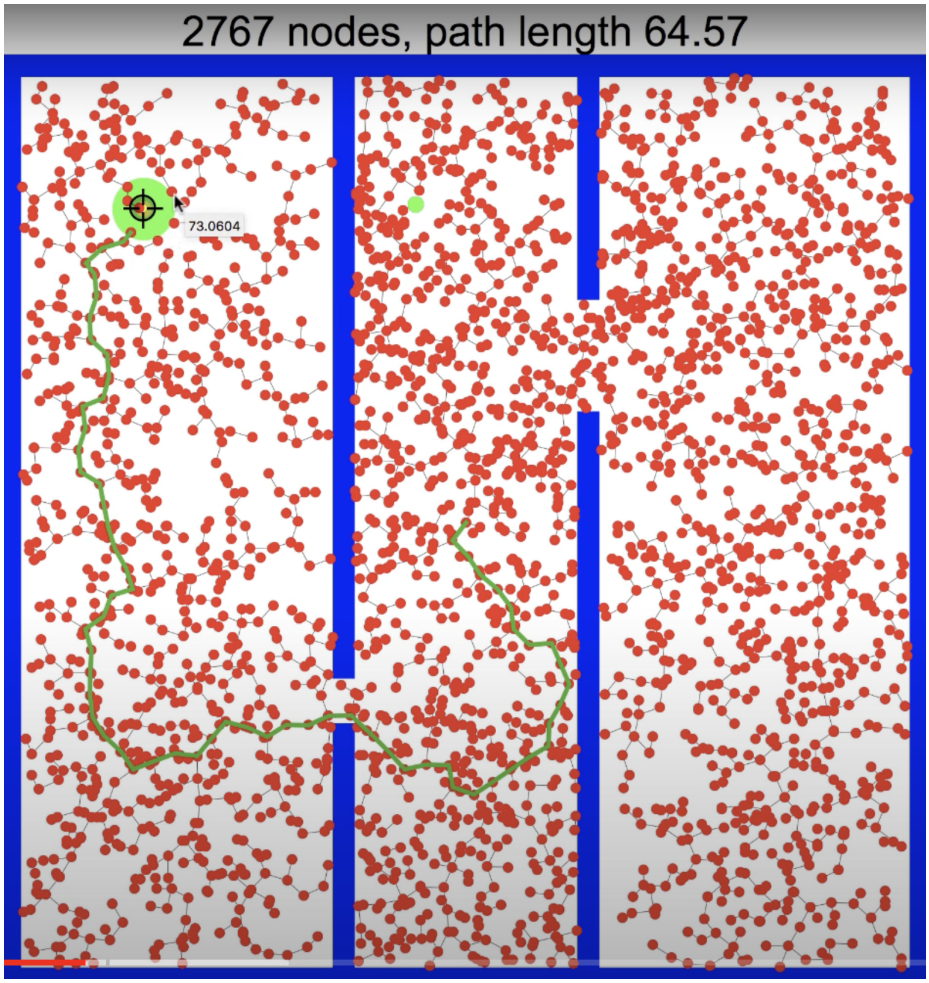
The RRT algorithm has several advantages over other motion planning algorithms, such as A*:

- It can handle high-dimensional spaces: The RRT algorithm can be applied to spaces with high dimensions, such as robotic manipulator configuration spaces.

- It is probabilistically complete: The RRT algorithm is guaranteed to find a solution if one exists, given enough time and resources.

- It can handle complex obstacle shapes: The RRT algorithm can handle complex obstacle shapes, as it generates random configurations and extends the tree toward them, which allows it to explore the search space more efficiently.

To perform the RRT algorithm, you start with an initial configuration (e.g., the starting point of a robot) and randomly sample a new configuration in the space. The algorithm then attempts to connect the sampled configuration to the nearest configuration in the existing tree. If the connection is feasible (i.e., it does not collide with obstacles), the new configuration is added to the tree. This process is repeated until a desired goal configuration is reached or a certain number of iterations are completed. The resulting tree represents a connected roadmap of the configuration space, which can be used to find a collision-free path from the initial to the goal configuration.



Pseudocode:

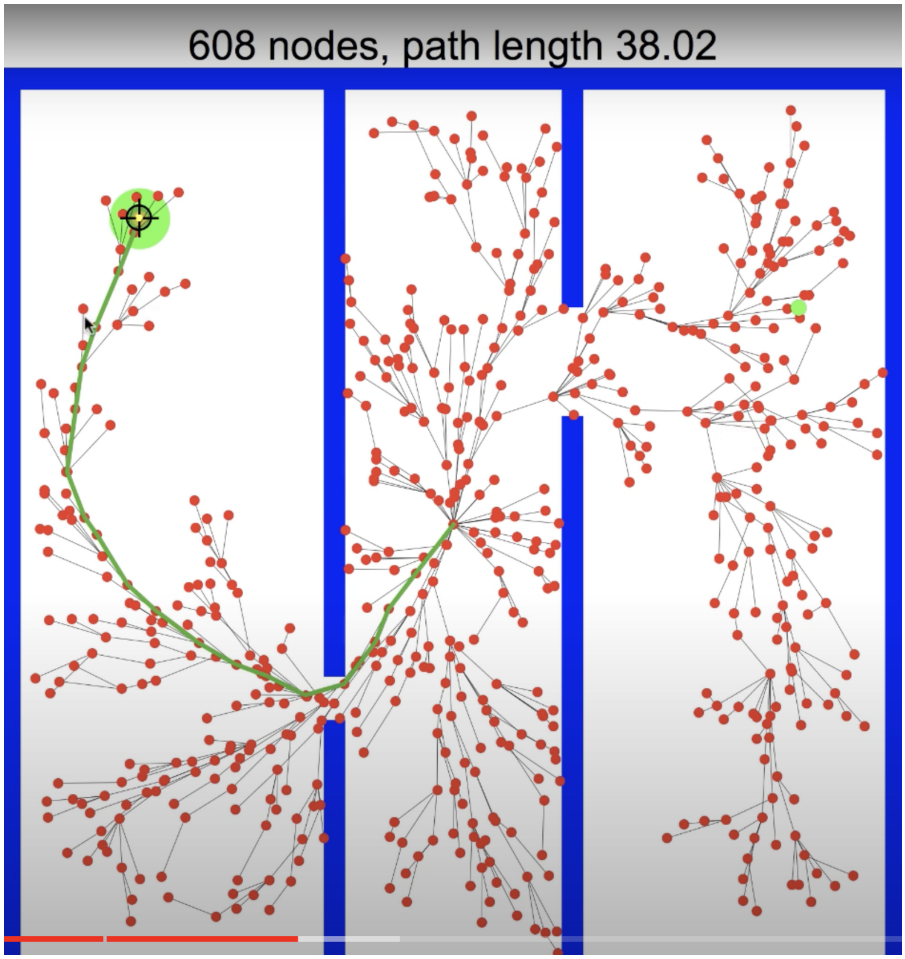
```
function RRT(start, goal, max_iter, step_size):
    create an empty tree T with only the start node
    for i = 1 to max_iter do:
        q_rand = randomly generate a configuration in the search space
        q_near = find the node in T that is closest to q_rand
        q_new = extend(q_near, q_rand, step_size)
        if q_new is not None and not in collision:
            T.add(q_new, q_near)
            if q_new is close to goal:
                path = find_path(T, q_new, goal)
                return path
    return None

function extend(q_near, q_rand, step_size):
    q_new = move q_near towards q_rand by step_size
    if q_new is in collision:
        return None
    else:
        return q_new

function find_path(T, q_start, q_goal):
    path = [q_goal]
    q_curr = q_goal
    while q_curr != q_start:
        q_prev = T.parent(q_curr)
        path.append(q_prev)
        q_curr = q_prev
    path.reverse()
    return path
```

RRT has various real-time applications in robotics and autonomous systems. For example, RRT can be used for the path planning of drones, autonomous vehicles, and robotic arms. In the context of computer games, RRT can be used to generate realistic character motions or to navigate game agents in dynamic environments. RRT can also be used in other fields, such as virtual reality, computer graphics, and simulation-based training, where real-time motion planning is required.

RRTStar:

RRT* (Rapidly-exploring Random Trees Star) is an extension of the original RRT (Rapidly-exploring Random Trees) algorithm that offers several advantages. First, RRT* aims to find an optimal solution by minimizing the path cost, while RRT only finds a feasible path. Second, RRT* maintains a more balanced and evenly distributed tree structure, which leads to a more efficient exploration of the state space compared to RRT. Lastly, RRT* can efficiently handle changes in the environment, making it more suitable for dynamic environments compared to RRT.

The key difference between RRT and RRT* is that the RRT* algorithm maintains a cost-to-come function at each node in the search tree, which represents the lowest cost path from the start configuration to that node. This cost-to-come function is updated as the tree grows, ensuring that each node in the tree has the lowest possible cost-to-come value. Additionally, the RRT* algorithm re-wires the search tree to improve the cost-to-come value of the nodes, by connecting them to other nodes with lower costs, if such a connection can be made without introducing any collisions with obstacles.



Pseudocode

```
RRT*(q_start, q_goal, K, delta_q, gamma):
    create a search tree T
    add q_start to T as the root node
    for k = 1 to K do
        q_rand = sample a random configuration in the search space
        q_near = nearest_neighbor(q_rand, T)
        q_new = extend(q_near, q_rand, delta_q)
        if q_new is not null and collision_free(q_near, q_new) then
            T.add_node(q_new)
            near_nodes = near_neighbors(q_new, T, gamma)
            for q_near in near_nodes do
                if collision_free(q_new, q_near) and cost(q_new, q_near) +
cost_to_come(q_new) < cost_to_come(q_near) then
                    T.remove_edge(parent(q_near), q_near)
                    T.add_edge(q_new, q_near)
        end if
    end for
    return T
end function

nearest_neighbor(q_rand, T):
    find the node q_near in T that is closest to q_rand
    return q_near

extend(q_near, q_rand, delta_q):
    q_new = interpolate(q_near, q_rand, delta_q)
    if collision_free(q_near, q_new) then
        return q_new
    else
        return null
    end if

near_neighbors(q_new, T, gamma):
```

```
    create an empty set near_nodes
    for each node q in T do
        if distance(q, q_new) < gamma then
            add q to near_nodes
        end if
    end for
    return near_nodes

cost(q1, q2):
    return the cost of the path from q1 to q2

cost_to_come(q):
    return the lowest cost path from the start configuration to q

parent(q):
    return the parent node of q in T

interpolate(q1, q2, delta_q):
    return a new configuration that is delta_q away from q1 in the
direction of q2

collision_free(q1, q2):
    return true if the path from q1 to q2 is collision-free, false
otherwise
```

Gamma controls the size of the neighborhood around each node in the search tree that is used for re-wiring.

RRT* has real-time applications in robotics, autonomous vehicles, and motion planning. For example, in robotics, RRT* can be used for planning collision-free paths for robot arms or drones in complex and dynamic environments. In autonomous vehicles, RRT* can assist in finding optimal paths for navigation while avoiding obstacles in real time. RRT* can also be used for planning trajectories for robotic surgeries, animation, and video games where real-time decision-making is critical.

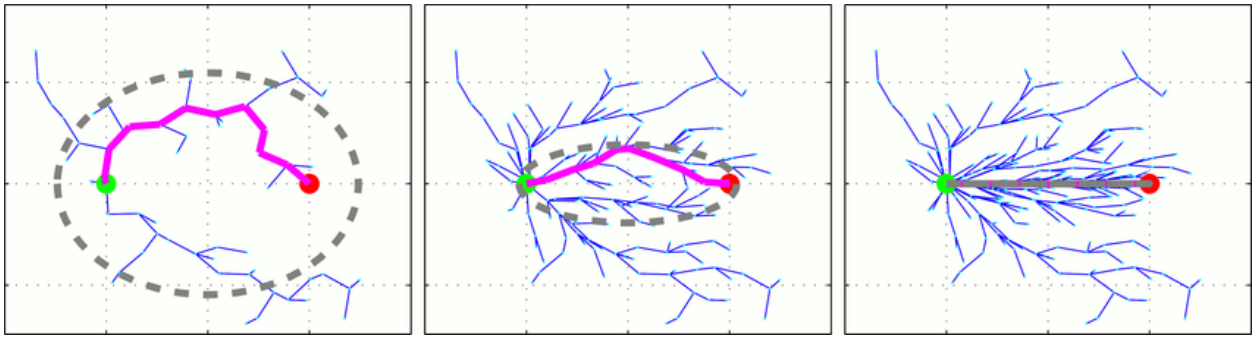
Informed RRTStar:

Informed RRT* is an improvement over the classic RRT* algorithm, which is a popular path-planning algorithm used in robotics and autonomous systems. Informed RRT*

improves the efficiency of RRT* by biasing the exploration towards the goal region, leading to faster convergence to a near-optimal solution. This is achieved by using heuristics or information about the goal region to guide the sampling and exploration process, resulting in fewer unnecessary samples and a more efficient search toward the goal.

To perform Informed RRT*, the following steps are typically followed:

- Define a heuristic function that estimates the cost from a given point to the goal region. This can be based on Euclidean distance, occupancy grid information, or any other domain-specific information.
- Initialize the tree with the starting point as the root.
- Sample a random point in the search space, biased towards the goal region based on the heuristic function.
- Extend the tree towards the sampled point, considering collision checking to avoid obstacles.
- Update the cost of the path from the root to the new node and rewire the tree to improve the path's quality.
- Repeat the sampling, extension, and rewiring steps until the goal region is reached or a desired solution quality is achieved.



Pseudocode:

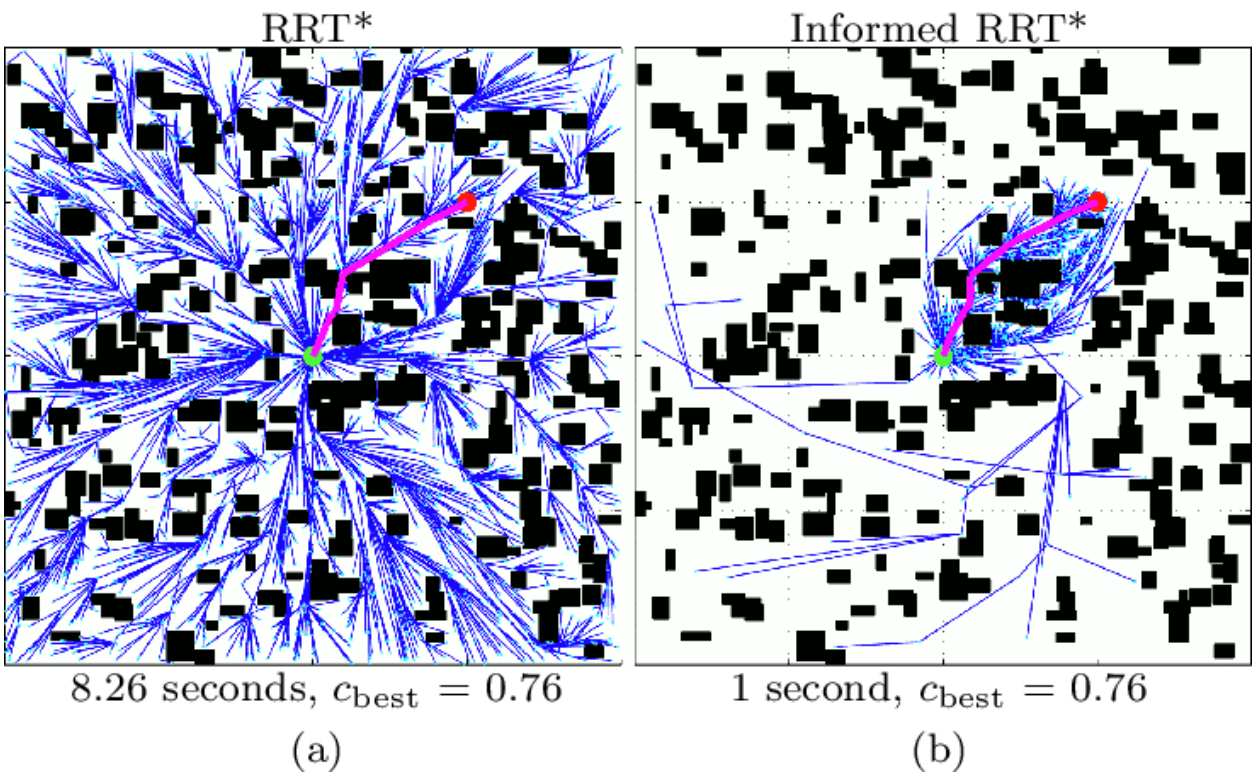
```
InformedRRT*(q_start, q_goal, K, delta_q, gamma, heuristic):
    create a search tree T
    add q_start to T as the root node
    set cost_to_come(q_start) = 0
    set heuristic(q_start) = estimate_cost(q_start, q_goal)
    for k = 1 to K do
```

```
        q_rand = sample a random configuration in the search space
        q_near = nearest_neighbor(q_rand, T)
        q_new = extend(q_near, q_rand, delta_q)
        if q_new is not null and collision_free(q_near, q_new) then
            T.add_node(q_new)
            set cost_to_come(q_new) = cost_to_come(q_near) + cost(q_near,
q_new)
            set heuristic(q_new) = estimate_cost(q_new, q_goal)
            near_nodes = near_neighbors(q_new, T, gamma)
            for q_near in near_nodes do
                if collision_free(q_new, q_near) and cost_to_come(q_new) +
cost(q_new, q_near) < cost_to_come(q_near) then
                    T.remove_edge(parent(q_near), q_near)
                    T.add_edge(q_new, q_near)
                    set cost_to_come(q_near) = cost_to_come(q_new) +
cost(q_new, q_near)
                    update_heuristic(q_near, q_goal, T)
        end if
    end for
    return T
end function

estimate_cost(q1, q2):
    return an estimate of the cost of the path from q1 to q2

update_heuristic(q, q_goal, T):
    set heuristic(q) = min(heuristic(q), estimate_cost(q, q_goal))
    for each node q_child in children(q) do
        update_heuristic(q_child, q_goal, T)
```

Informed RRT* has real-time applications in various fields, such as robotics, autonomous vehicles, and video games. For example:

- Robotic path planning: Informed RRT* can be used in real-time to plan efficient and collision-free paths for autonomous robots, allowing them to navigate complex environments and avoid obstacles while converging towards a goal.
- Autonomous vehicles: Informed RRT* can be used in real-time for trajectory planning and obstacle avoidance in self-driving cars, enabling them to plan safe and efficient routes on the fly while considering real-time sensor data.

- Video games: Informed RRT* can be used in real-time for generating realistic and dynamic motion planning for virtual characters in video games, providing interactive and responsive navigation behavior for game agents.



Summary of Research Contributions and Limitations of RRT*-based Optimal Path Planning Approaches for Mobile Robots.

Optimal path planning refers to finding the collision-free, shortest, and smooth route between start and goal positions. This task is essential in many robotic applications such as autonomous cars, surveillance operations, agricultural robots, and planetary and space exploration missions. Rapidly exploring Random Tree Star (RRT*) is a renowned sampling-based planning approach. It has gained immense popularity due to its support for high-dimensional complex problems. A significant body of research has

addressed the problem of optimal path planning for mobile robots using RRT* based approaches.

The most relevant papers, along with the research contributions and limitations are summarized below in Table 1. Different attributes of the state-of-the-art approaches are also listed in Table 2.

Table - 1:

Sr#	Author, year	Approach	Research Contributions	Limitations / Future Recommendations
1.	Karaman and Frazzoli [7], 2011	RRT*	• Proved asymptotically optimal property for RRT*. • Introduced new key features of near neighbor search and rewiring operations. • Visibly refined path quality than original RRT.	• New features had an efficiency trade-off. Insertion of good candidate node with best parent selection improved tree cost but on the other hand it also slowed down convergence rate of RRT*. • Jagged, suboptimal paths and slow convergence • Large memory requirements.
2.	Karaman et al. [4], 2011	Anytime RRT*	• It introduced two key features called committed trajectories and branch-and-bound adaptation. • It improved trajectory and computational efficiency by gradually removing nodes from tree which are unable to improve current solution path.	• Jagged and suboptimal paths. • Could overestimate and may cause unnecessary node removal during initial expansion of tree when it is not mature.
3.	Akgun and Stilman [58], 2011	B-RRT*	• Improved convergence speed and path refinement using sample rejection with an admissible heuristic.	• Attempt to connect both trees in each iteration incurred computational overhead. • Jagged and suboptimal paths. • Large memory requirements.
4.	Adiyatov and Varol [33], 2013	RRT*FN	• Memory efficient version of RRT* by forcing a fixed number of nodes in the tree.	• Jagged, suboptimal path. • Rate of convergence to optimal path is lower than base RRT*. • Worked only for static known environment.
5.	Nasir et al. [35], 2013	RRT*-Smart	• Two new features intelligent sampling and path optimization were introduced. • It accelerated the convergence rate with improved efficiency with respect to both time and cost	• It is dependent upon a heuristic called Biasing Ratio which has a trade-off between convergence rate and exploration of space. • Heuristic used by this approach are not automated and require programmer dependent value for different environments.
6.	Jordan and Perez [36], 2013	Optimal B-RRT*	• Introduced multiple heuristics, based on different conditions to increase convergence rate of bidirectional RRT*.	• Use of multiple heuristics caused computational overload. • Biased heuristics interfere with the algorithm characteristics (such as exploration, node rejection, cost function) and limited its application also.
7.	O. Arslan and P. Tsiotras [50], 2013	RRT# (RRT "sharp")	• Introduced a global replanning scheme to maintain promising nodes of tree to make fast convergence towards optimal path with low cost.	• Real time experiments and applications would be beneficial to investigate and improve the efficiency of approach.
8.	Webb and Berg [49, 64], 2012, 2013	Adapted RRT*	• Presented kinodynamic extension of RRT* with ensured asymptotic optimality with controllable linear dynamics, in multi dimension state space.	• Post processing steps for path smoothness and real world quadrotors experiments are planned for future work. However, such experiment would require controller stabilization for final trajectory.
9.	Lee et al. [44], 2014	SRRT*	• Proposed spline based RRT* based on a cubic Bézier curve for fixed-wing UAVs. • Presented geometric collision and dynamic feasibility function checking constraints during tree expansion. • Produced feasible smooth and cost-optimal path in 3D simulations.	• Online planner with real time application would be beneficial to further investigate and improve the efficiency of approach.
10.	Gammell et al. [34], 2014	Informed RRT*	• Proposed direct sampling technique based on ellipsoidal informed subset, which showed improved convergence than RRT*.	• Heuristic used to shrink planning problem is highly dependent upon initial solution cost which makes it effective only under certain conditions.
11.	Qureshi and Ayaz [37], 2015	IB-RRT*	• Introduced intelligent sample insertion heuristic with minimal memory requirements, which improved the path quality as compared to RRT* and B-RRT*.	• Its application needs further investigation for online planning.
12.	Moon and Chung [39], 2015	DT-RRT	• Addressed kinodynamic planning for high speed mobile robot and produced practically feasible trajectories. • Instead of using rewire operation, it introduced reconnect-tree scheme to maintain child nodes with reduced computational cost.	• Approach need advancement for dynamic and unknown scenarios as its reconnect-tree scheme is only beneficial in known environments.

			• Reduced the node rejection chances by using kinodynamic conditions.	
13.	Alejo et al. [3], 2015	RRT*i	• Predictable and practical trajectory generation for UAVs. • Smoother trajectories as compared to RRT and RRT* with improved path quality. • Introduced new local sampling technique.	• Could produce unexpected collisions in multi-UAV applications with increased uncertainty.
14.	Csorvasi et al. [43], 2015	RTR+CS*	• Used a local planner to generate feasible path for car-like robots using circular arcs and straight segments.	• Generated paths are not curvature continuous and natural. • It is still under experimental process to improve computational performance.
15.	Lan and Di Cairano [2], 2015	Mitsubishi RRT*	• Addressed the problem of G^2 continuous curvature smooth path for autonomous driving vehicle with the capability of lane management on roads. • Introduced a local replanning procedure to safely avoid and re-plan due to dynamic obstacles.	• In context of autonomous vehicle driving, uncertainty of dynamic environment is not managed.
16.	J. Suh et al. [65] , 2015	CARRT*	• Addressed the problem of large number of samples by using two trees and cross entropy in RRT*. • Produced energy efficient path in high dimensional space.	• It is limited to address the planning problems for humanoid robots only.
17.	Qureshi and Ayaz [48], 2016	PRRT*	• Addressed the problem of high memory consumption and slow convergence by incorporating artificial potential field characteristic in RRT*.	• Considering optimal results with fast convergence than RRT*, approach could be employed for online planning in future.
18.	Yi et al. [45], 2016	HARRT*	• Presented a human-robot interactive planner using RRT* and homotopy algorithm.	• Trade-off between computational efficiency and path quality. • Natural language processing or graphical user interface could be adapted for compatibility, usability and workload of human and robot interactions.
19.	Devaurs et al. [46], 2016	T-RRT*	• Integrated transition tests with RRT* for efficient extension of tree in a cost space.	• Performance analysis with RRT* in different problems would be of interest to show its beneficial problem class.
20.	Choudhury et al. [47], 2016	RABIT*	• Used an informed global technique BIT* with RRT* to find optimal path for narrow passages in high dimensions.	• Local optimizer suggested by approach needs further investigation for optimization of path.

Table - 2:

Approaches	Constraints	Planning Mode	Kinematic Model	Sampling Strategy	Metric
1. RRT* [7]	Holonomic	Offline	Point	Uniform	Euclidean
2. Anytime RRT* [4]	Non-holonomic	Online	Dubin Car	Uniform	Euclidean + Velocity
3. B-RRT* [58]	Holonomic	Offline	Rigid Body	Local bias	Goal biased
4. RRT*FN [33]	Holonomic	Offline	Robotic Arm	Uniform	Cumulative Euclidean
5. RRT*-Smart [35]	Holonomic	Offline	Point	Intelligent	Euclidean
6. Optimal B-RRT* [36]	Holonomic	Offline	Point	Uniform	Euclidean
7. RRT# [50]	IIolonomic	Offline	Point	Uniform	Euclidean
8. Adapted RRT* [64], [49]	Non-holonomic	Offline	Car-like and UAV	Uniform	A* Heuristic
9. SRRT* [44]	Non-holonomic	Offline	UAV	Uniform	Geometric + dynamic constraint
10. Informed RRT* [34]	Holonomic	Offline	Point	Direct Sampling	Euclidean
11. IB-RRT* [37]	Holonomic	Offline	Point	Intelligent	Greedy + Euclidean
12. DT-RRT [39]	Non-holonomic	Offline	Car-like	Hybrid	Angular + Euclidean
13. RRT*i [3]	Non-holonomic	Online	UAV	Local Sampling	A* Heuristic
14. RTR+CS* [43]	Non-holonomic	Offline	Car-like	Uniform + Local Planning	Angular + Euclidean
15. Mitsubishi RRT* [2]	Non-holonomic	Online	Autonomous Car	Two-stage sampling	Weighted Euclidean
16. CARRT* [65]	Non-holonomic	Online	Humanoid	Uniform	MW Energy Cost
17. PRRT* [48]	Non-holonomic	Offline	P3-DX	Uniform	Euclidean
18. HARRT* [45]	Holonomic	Offline	Point	Uniform	Homotopy check
19. T-RRT* [46]	Non-holonomic	Offline	Quadrotor	Uniform	Transition test
20. RABIT* [47]	Non-holonomic	Offline	Autonomous helicopter	Uniform	A* Heuristic

Reference:

1. Rapidly exploring random tree: Progress and Prospects
2. Sampling-based Algorithms for Optimal Motion Planning by Sertac Karaman and Emilio Frazzoli
3. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic
4. Optimal path planning using RRT* based approaches: a survey and future directions