RRT*-Quick

A Motion Planning Algorithm with Faster Convergence Rate

By:

Pooja Kabra
Paras Savnani

UNIVERSITY OF MARYLAND

Introduction

Conventional graph-based planning methods require us to discretize the search space.

However, as the dimension of the search space increases, the memory requirement grows exponentially and the solution becomes more difficult to reach.

Sampling-based algorithms are employed extensively in a high dimensional space and are shown to work well. Besides they are also theoretically proven to be probabilistically complete and asymptotically optimal.

In this project, we explore RRT*-Quick as an improved version of RRT*

RRT

- searches for a ***feasible*** trajectory
- repeatedly connect a randomly sampled state to nearest node in the existing tree if there is no collision
- **no improvement** with increase in iterations, as there is no rewiring or tracking of cost

RRT*

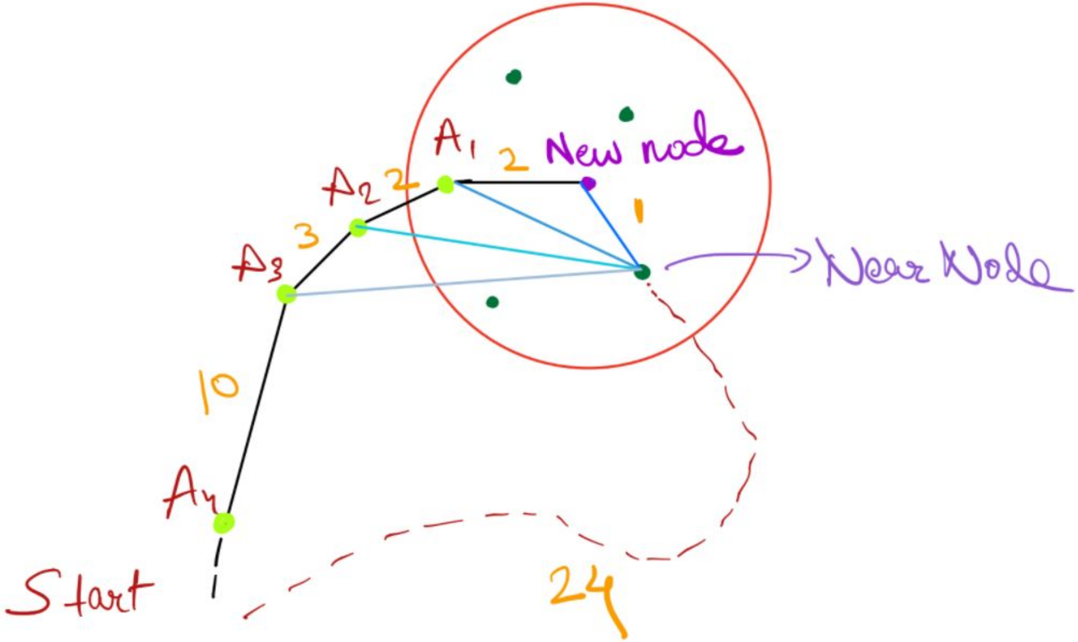
- focuses on ***improving*** the quality of solutions
- While RRT simply connects a random sample to the nearest node, RRT* searches the nearby nodes within a **volume sphere** to find that node which makes the lowest cost-to-come to the new node
- It also tries to **rewire the nearby nodes via the new node** if it lowers their cost
- converges to optimal as iterations increase, **radius** of the volume sphere decides the converge rate

RRT* Quick

- The concept of RRT* Quick is based on the premise that nearby nodes tend to share ***common parents***
- Triangular inequality:

  If we rewire the nearby nodes via parents of the new node instead of the new node itself, it will definitely result in lower cost

RRT* Quick



Psuedocode

RRT*

```
Input: T = (V, E)
Output: T
while i < N do
    x_rand ← Sample(i);
    x_nearest ← Nearest(T, x_rand);
    x_new ← Extend(x_nearest, x_rand);
    σ_new ← Steer(x_nearest, x_rand);
    if CollisionFree(σ_new) then
        X_near ← Near(T, x_new);
        x_min ← x_nearest;
        c_min ← Cost(x_nearest) + Cost(σ_new);
        foreach x_near ∈ X_near do
            σ_near ← Steer(x_near, x_new);
            if Cost(x_near) + Cost(σ_near) < c_min ∧ CollisionFree(σ_near) then
                x_min ← x_near;
                c_min ← Cost(x_near) + Cost(σ_near);
            end
        end
        V ← V ∪ x_new;
        E ← E ∪ {(x_min, x_new)};
        foreach x_near ∈ X_near do
            σ_near ← Steer(x_new, x_near);
            if Cost(x_new) + Cost(σ_near) < Cost(x_near) ∧ CollisionFree(σ_near)
            then
                E ← E \ {(Parent(x_near), x_near)};
                E ← E ∪ {(x_new, x_near)};
            end
        end
    end
end
T ← (V, E);
return T;
```

RRT* Quick

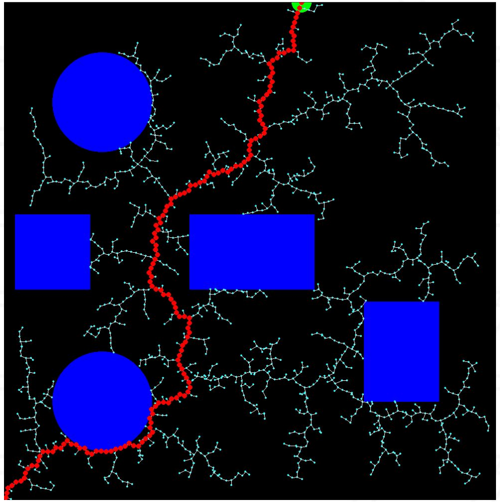
```
Input: T = (V, E)
Output: T
while i < N do
    x_rand ← Sample(i);
    x_nearest ← Nearest(T, x_rand);
    x_new ← Extend(x_nearest, x_rand);
    σ_new ← Steer(x_nearest, x_rand);
    if CollisionFree(σ_new) then
        X_near ← Near(T, x_new);
        x_min ← x_nearest;
        c_min ← Cost(x_nearest) + Cost(σ_new);
        foreach x_near ∈ X_near ∪ Ancestors(X_near, degree) do
            σ_near ← Steer(x_near, x_new);
            if Cost(x_near) + Cost(σ_near) < c_min ∧ CollisionFree(σ_near) then
                x_min ← x_near;
                c_min ← Cost(x_near) + Cost(σ_near);
            end
        end
        V ← V ∪ x_new;
        E ← E ∪ {(x_min, x_new)};
        foreach x_near ∈ X_near do
            σ_near ← Steer(x_new, x_near);
            X_candidates ← Ancestors(x_new, degree) \ Ancestors(x_near, degree);
            x_min ← x_new;
            c_min ← Cost(x_new) + Cost(σ_new);
            foreach x_c ∈ X_candidates do
                σ_c ← Steer(x_c, x_near);
                if Cost(x_c) + Cost(σ_c) < c_min) ∧ CollisionFree(σ_c) then
                    x_min ← x_c;
                    c_min ← Cost(x_c) + Cost(σ_c);
                end
            end
            E ← E \ {(Parent(x_near), x_near)};
            E ← E ∪ {(x_min, x_near)};
        end
    end
end
T ← (V, E);
return T;
```

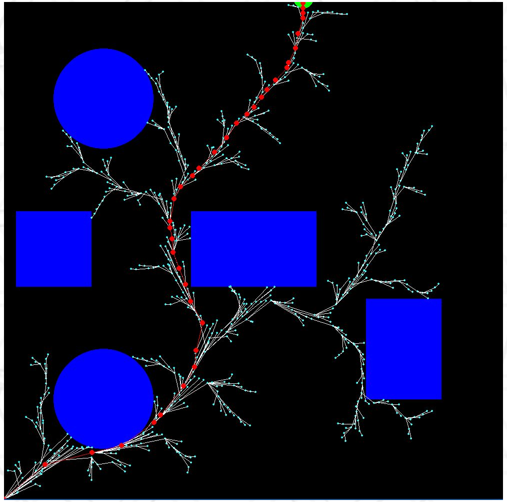
Increasing Radius vs Using Ancestors ?

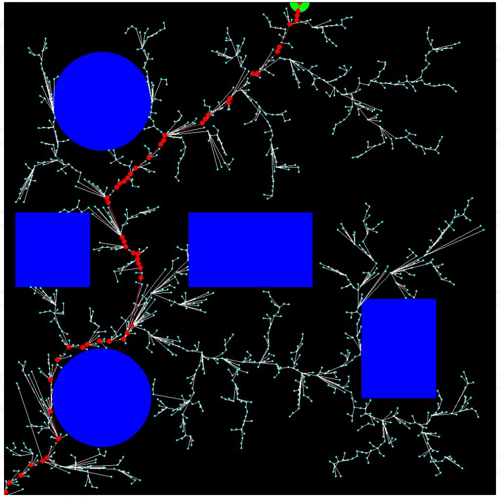
ROS Gazebo Simulation Turtlebot(WIP)

- 2 nodes (RRT* Quick → Robotcontrol)
- Movebase Action client(Robot control)
- AMCL (Adaptive Monte Carlo Localization)
- Waypoint coordinates transmitted to the main controller node by RRT node.

RRT vs RRT* vs RRT* Quick

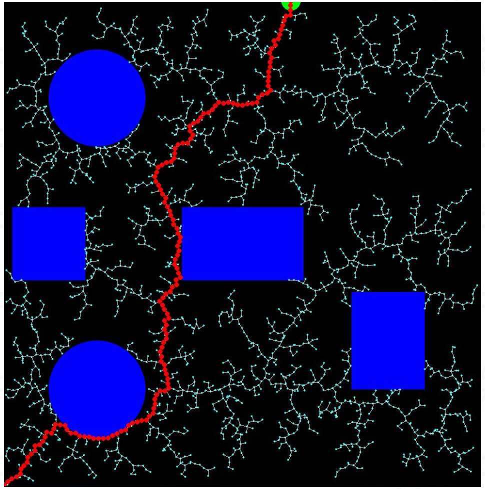


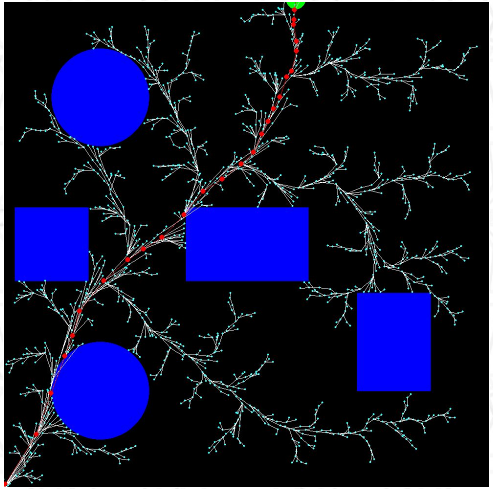


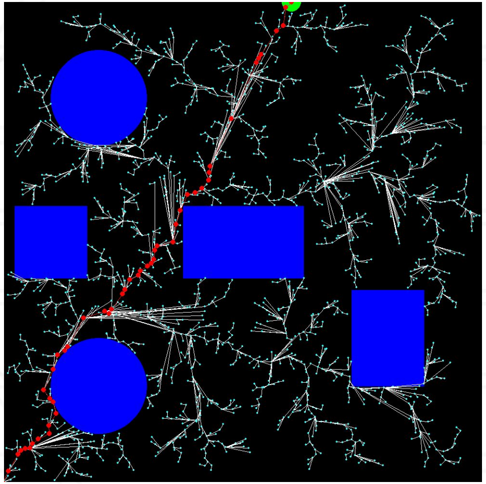


2000 iterations

RRT vs RRT* vs RRT* Quick



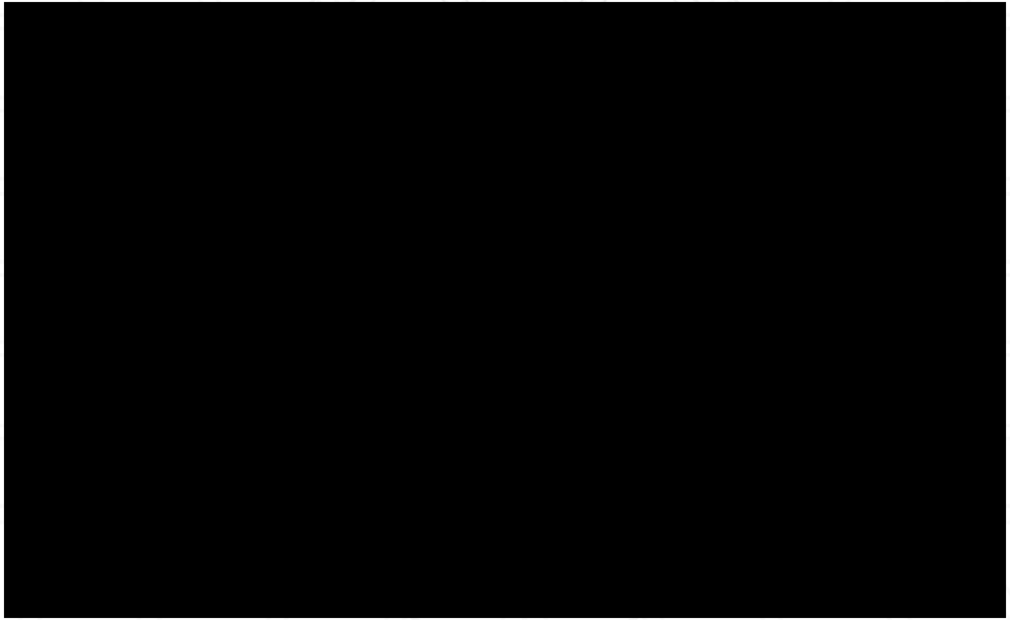




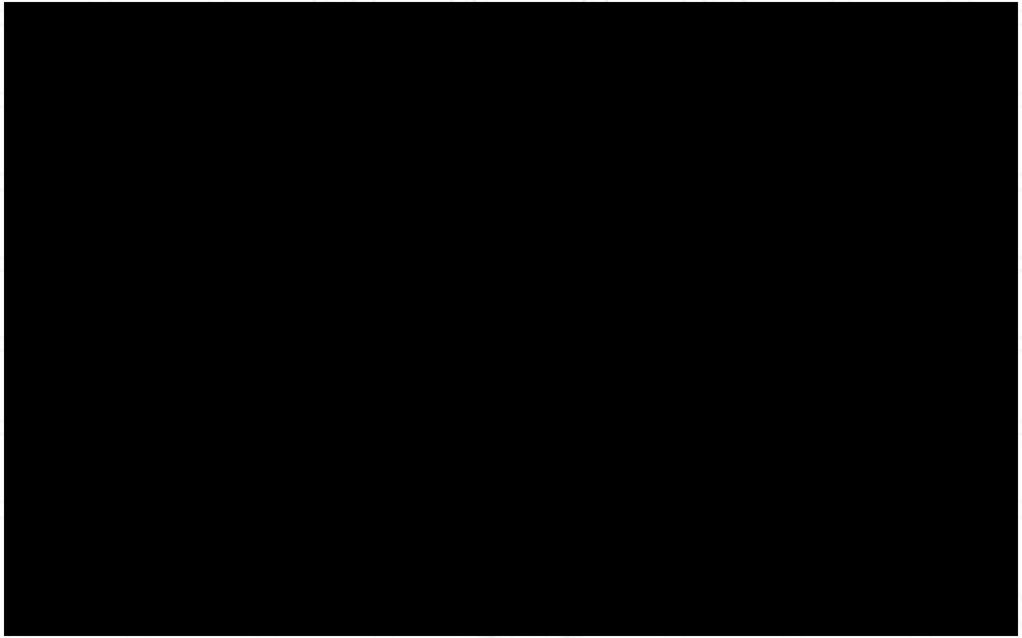
3000 iterations

Visualization

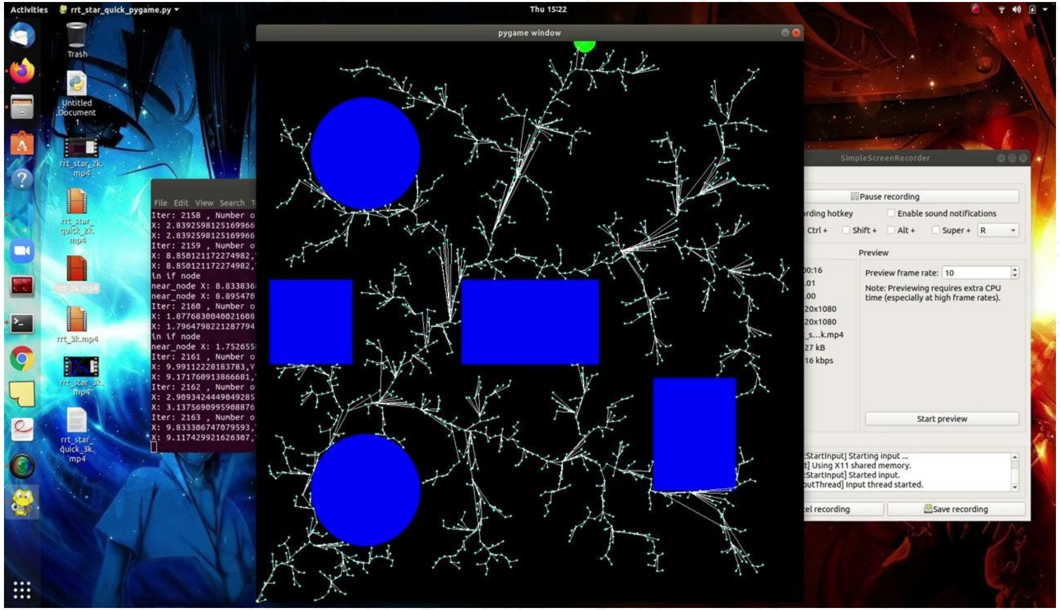
RRT(3000 iterations)



RRT*(3000 iterations)



RRT* Quick(3000 iterations)



Conclusion and Future Work

- RRT* Quick performs much better than RRT and RRT* with almost same time complexity

  - Implement Cost to go with RRT* Quick to even improve it further.
  - Multi-agent path planning using RRT* Quick.

UNIVERSITY OF
MARYLAND