Apollo: Autonomous Parking (Motion Planning and Control)

Amith Ramdas Achari, Peng Han, Yuehui Yu, Xiaoxu Sun

amithr3, penghan2, yuehuiy2, xiaoxus2

Problem Statement

The problem is to plan and follow a trajectory from point A to point B avoiding obstacles with smaller errors.

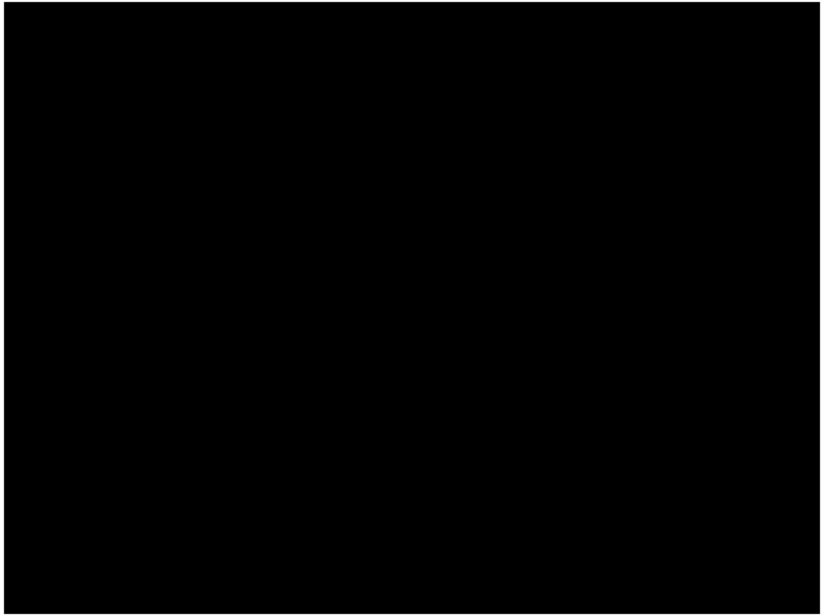
- **Planning:** Determine the trajectory that GEM has to follow, this is done using sampling based planning, namely RRT*
- **Control:** Control the vehicle to track the trajectory resulted from planning using a PID controller.

Motivation:

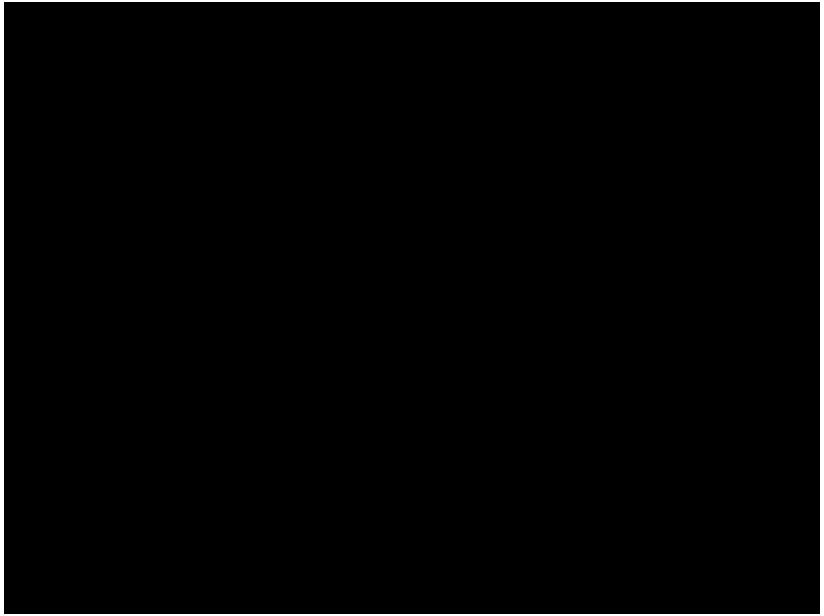
- Motion Planning is a difficult problem
- Planning technique can be used for different use cases. RRT* is one of state of the art techniques used in the autonomous driving sector

Problem Faced

Find the shortest path between two points using Astar algorithm



Apply to a real car
Oops!



Claim

- Sampling based technique RRT* is used to generate waypoints that obey the non-holonomic constraints of the vehicle
- Dubins RRT* is used to solve the problem of constraints
- The vehicle is successfully controlled to follow a second order polynomial generated by our planning algorithm's waypoints.

RRT vs RRT*

Algorithm 3: RRT

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, 2, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}(i)$;
  $v_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(v_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{CollisionFree}(v_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $E \leftarrow E \cup \{(v_{\text{nearest}}, x_{\text{new}})\}$;

Algorithm 6: RRT*

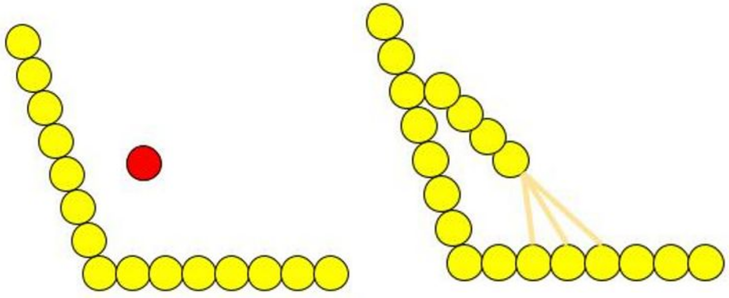
$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, 2, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}(i)$;
  $v_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(v_{\text{nearest}}, x_{\text{new}})$;
  if $\texttt{CollisionFree}(v_{\text{nearest}}, x_{\text{new}})$ **then**
    $U \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;

    // Connect along a minimum-cost path
    $v_{\min} \leftarrow v_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(v_{\text{nearest}}) + c(\texttt{Path}(v_{\text{nearest}}, x_{\text{new}}))$;
    for *all* $u \in U$ **do**
      if $\texttt{CollisionFree}(u, x_{\text{new}})$ *and* $\texttt{Cost}(u) + c(\texttt{Path}(u, x_{\text{new}})) < c_{\min}$ **then**
        $v_{\min} \leftarrow u$; $c_{\min} \leftarrow \texttt{Cost}(u) + c(\texttt{Path}(u, x_{\text{new}}))$;

    $E \leftarrow E \cup \{(v_{\min}, x_{\text{new}})\}$;

    // Rewire vertices
    for *all* $u \in U$ **do**
      if $\texttt{CollisionFree}(x_{\text{new}}, u)$ *and* $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Path}(x_{\text{new}}, u)) < \texttt{Cost}(u)$ **then**
        $v_{\text{parent}} \leftarrow \texttt{Parent}(u)$;
        $E \leftarrow (E \setminus \{(v_{\text{parent}}, u)\}) \cup \{(x_{\text{new}}, u)\}$;

return $G = (V, E)$;



Rewire Step

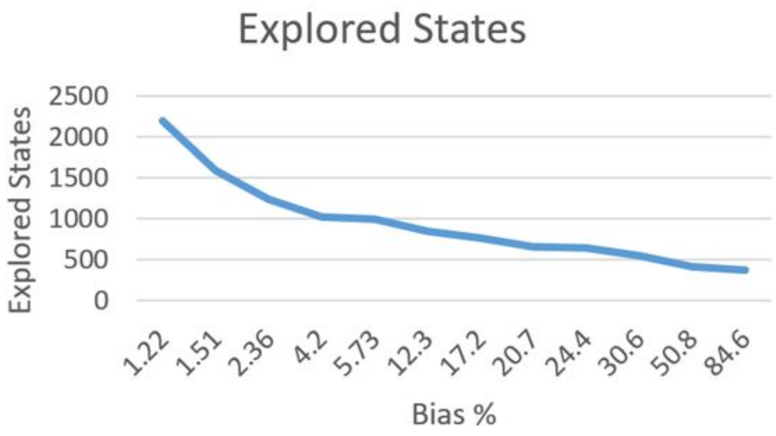
Results

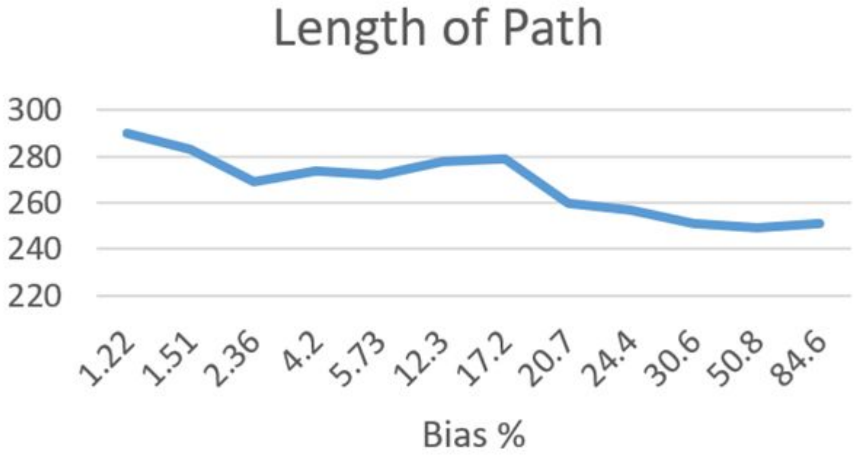
Important results to learn:

- Multiple tests simulated to find the tuning parameter of RRT*
    - **Bias** - It either explores or exploits knowing the goal position. By increasing the likelihood of sampling states from a specific area, RRT* growth can be biased.
    - **Step Radius** - The radius within which nodes look for new neighbors to connect with.

Hyperparameter: Bias

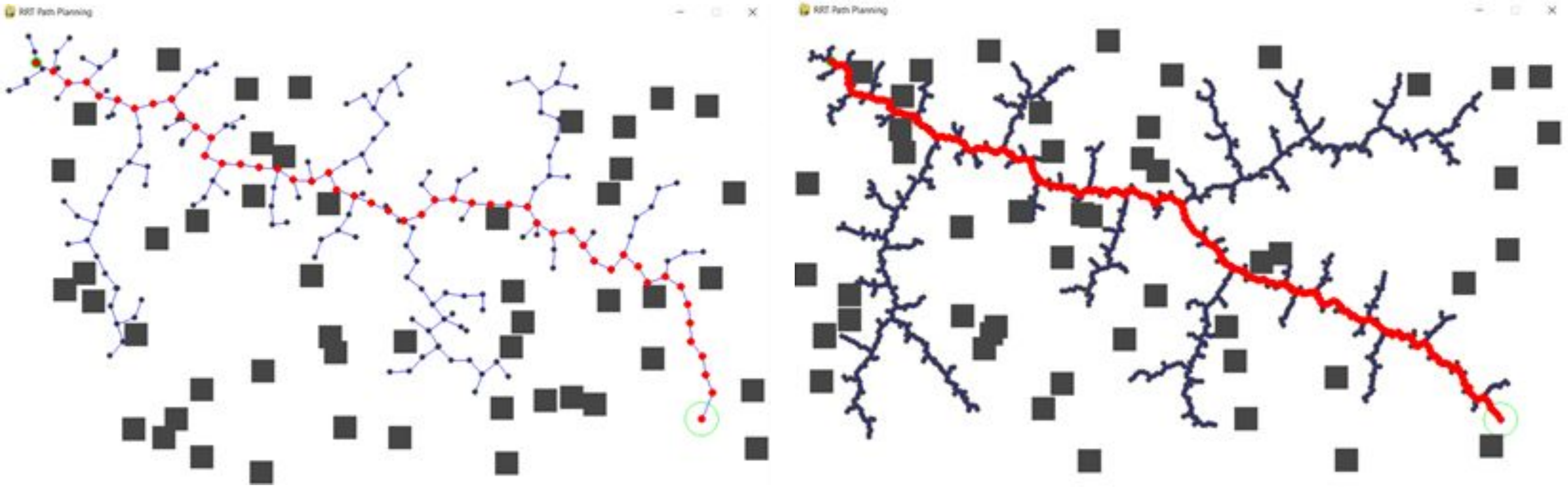
Bias





Problem with High Bias: Cannot be used in blocked environments.

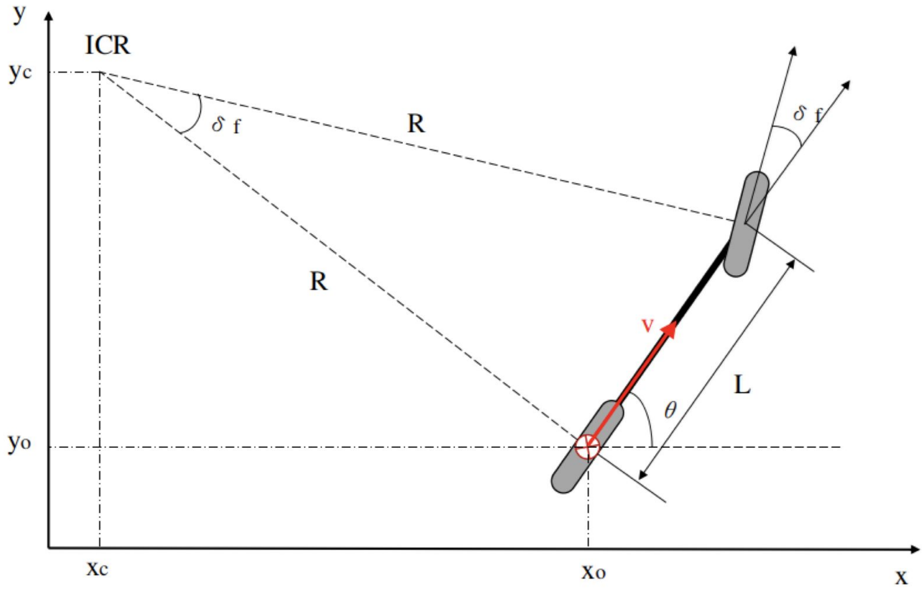
Hyperparameter: Step Radius



Left: Step Radius = 25px (3.03ms), Right: Step Radius = 5px (3.35ms)

RRT-star algorithm with Dubins path

Vehicle model and Dubins path



$L = 1.75$ m
$\delta_f = 35$ deg

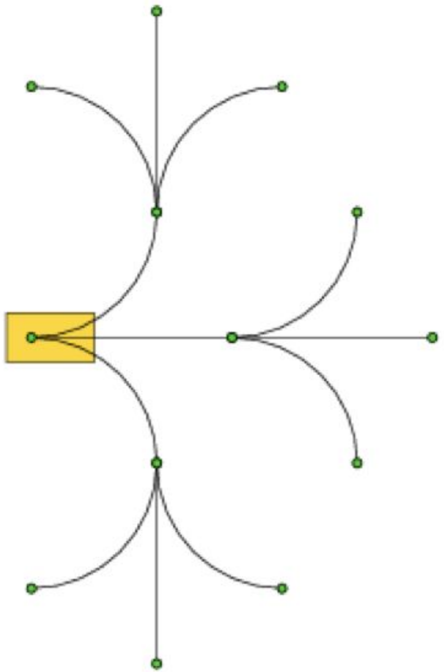
Minimum Radius $= R$
$= L / \tan(\delta_f)$
$= 2.7$ m

Curvature $= 1 / R$
$= 1 / 2.7$

RRT-star algorithm with Dubins path

Vehicle model and Dubins path

$$x_{\text{new}} \leftarrow \texttt{Steer}(v_{\text{nearest}}, x_{\text{rand}});$$



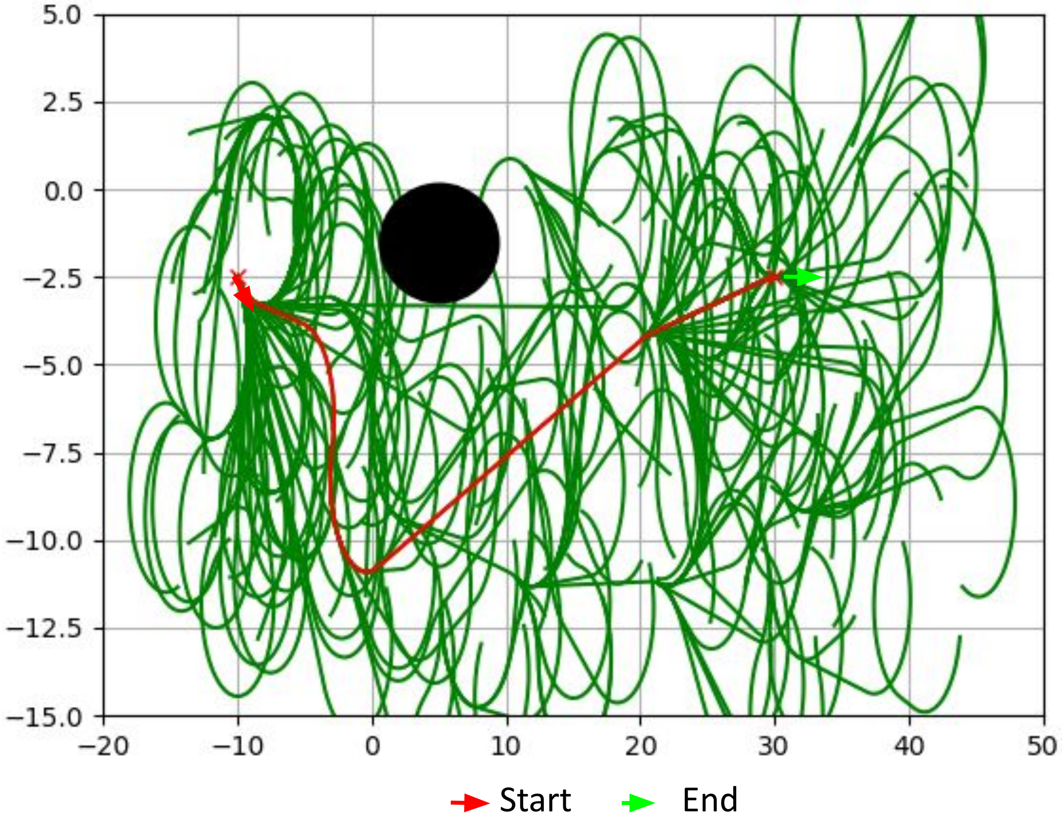
Two stages

```python
def Steer(self, node_start, node_end):
    sx, sy, syaw = node_start.x, node_start.y, node_start.yaw
    gx, gy, gyaw = node_end.x, node_end.y, node_end.yaw
    maxc = self.curv

    path = dubins.calc_dubins_path(sx, sy, syaw, gx, gy, gyaw, maxc)

    if len(path.x) <= 1:
        return None

    node_new = Node(path.x[-1], path.y[-1], path.yaw[-1])
    node_new.path_x = path.x
    node_new.path_y = path.y
    node_new.path_yaw = path.yaw
    node_new.cost = node_start.cost + path.L
    node_new.parent = node_start

    return node_new
```

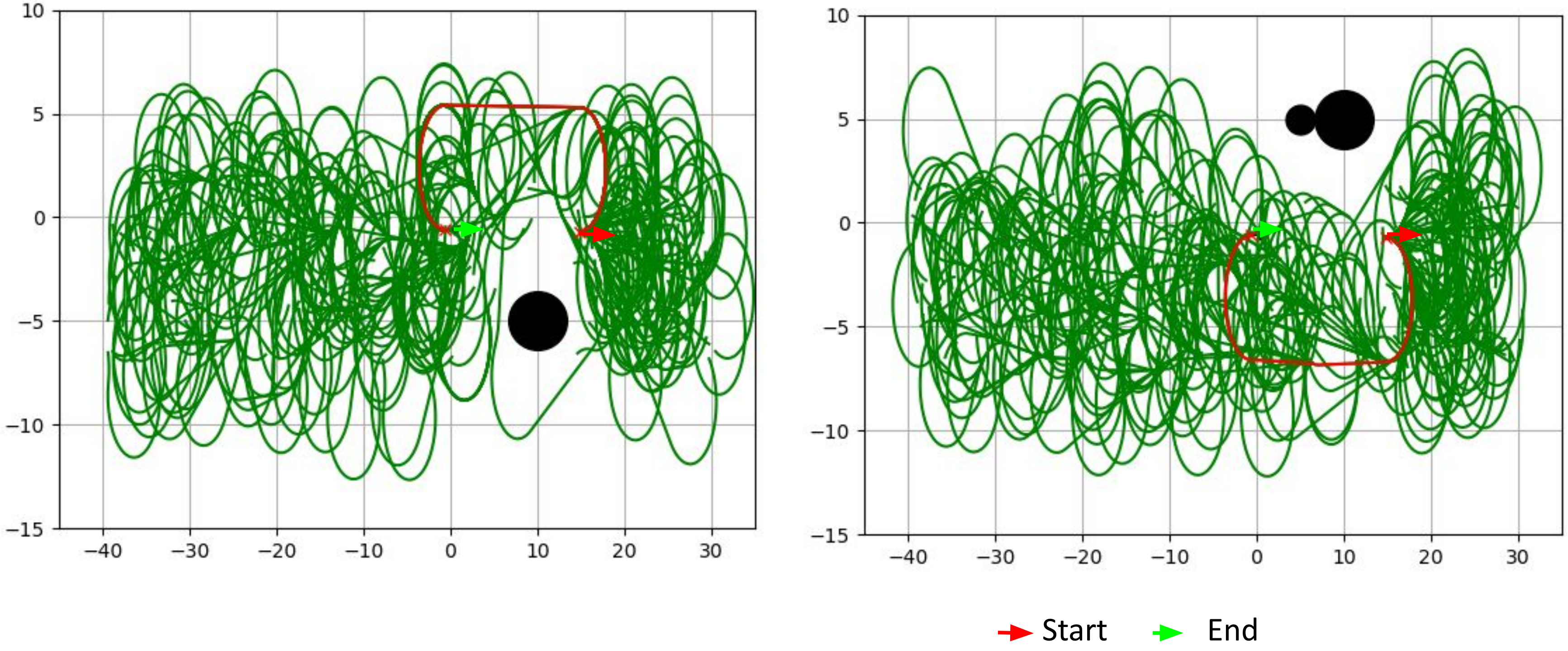
Dubins RRT* algorithm

- Generate smooth Trajectory
- Asymptotic optimality

- **Input:** Start, Goal, Map
- **Output:** Waypoints (x,y, heading)
- **Problem:** Doesn't guarantee completeness



Path generated with obstacles



Path for different Nodes

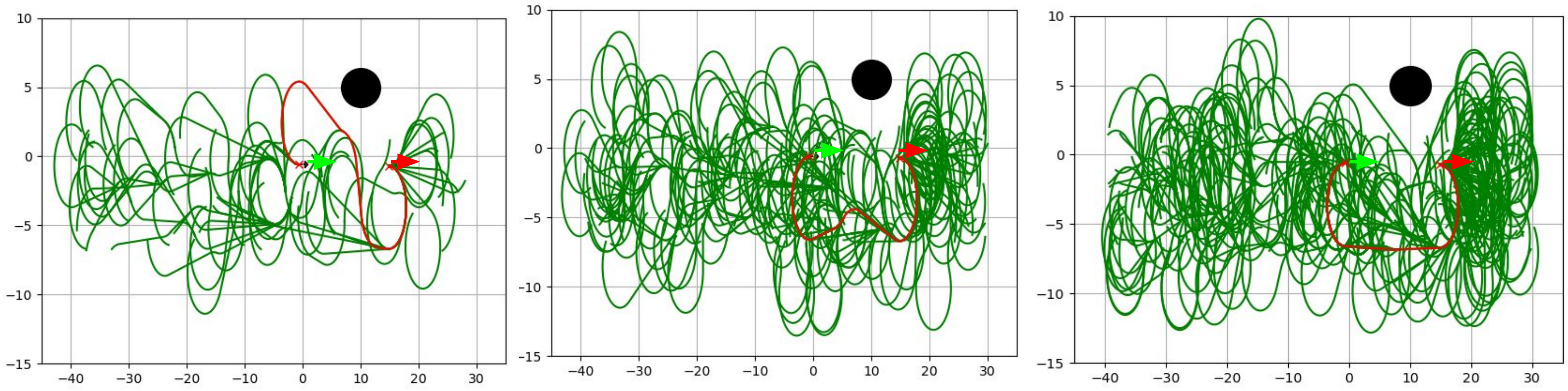
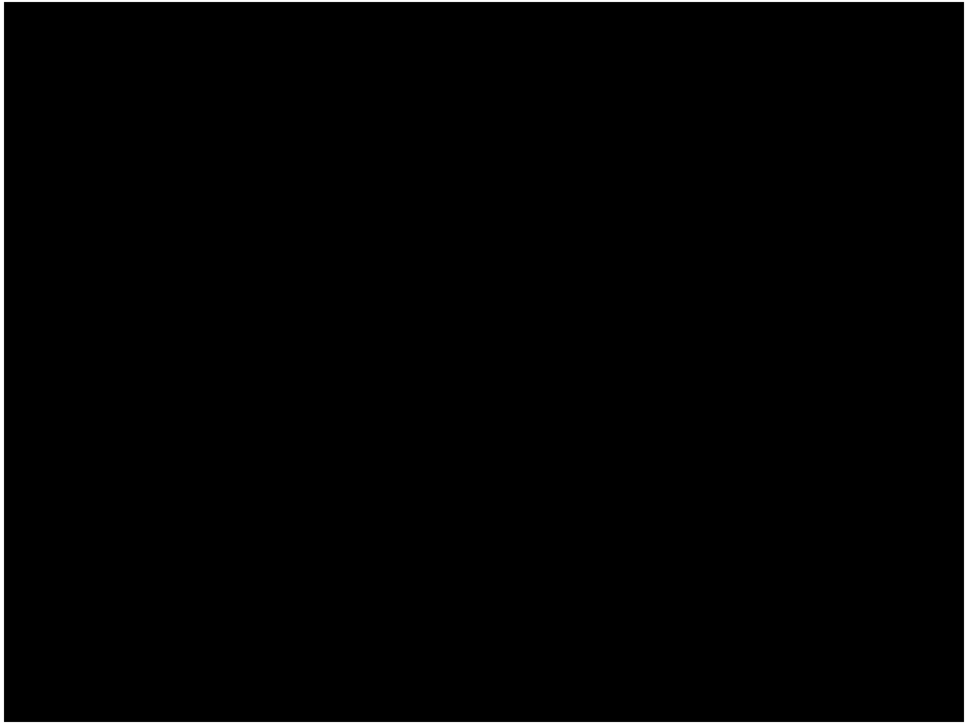
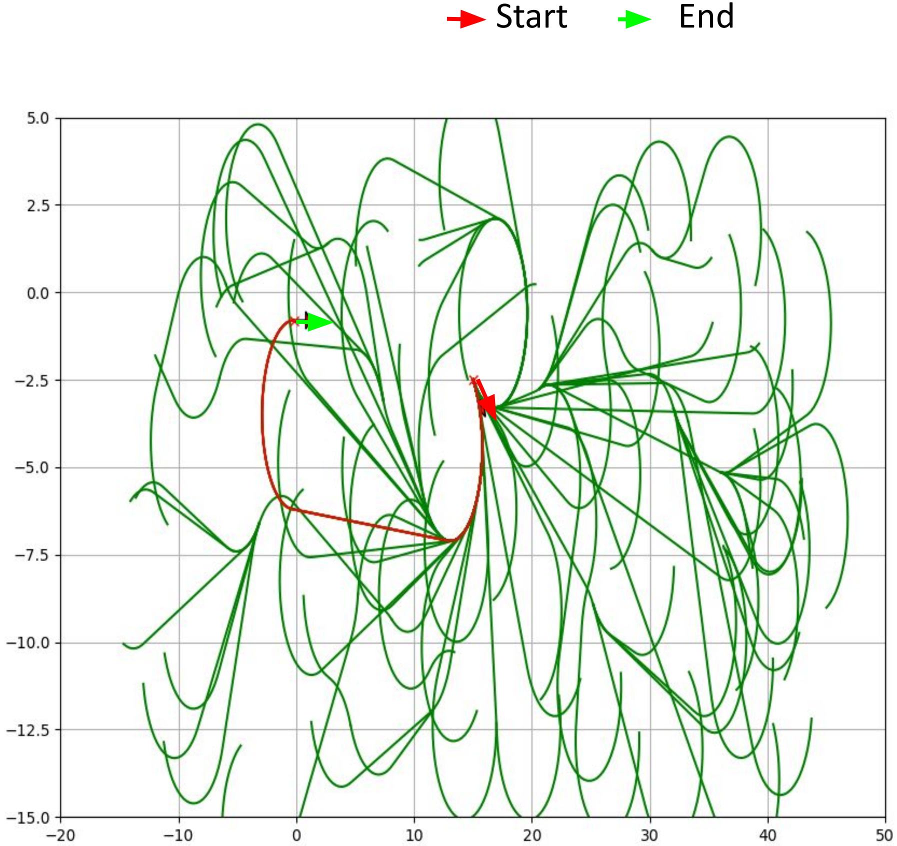


Figure 10. Left: 100 nodes, Middle: 300 nodes, Right: 500 nodes
With increase the number of nodes the path length decreased.

Video of Dubins RRT* in Action

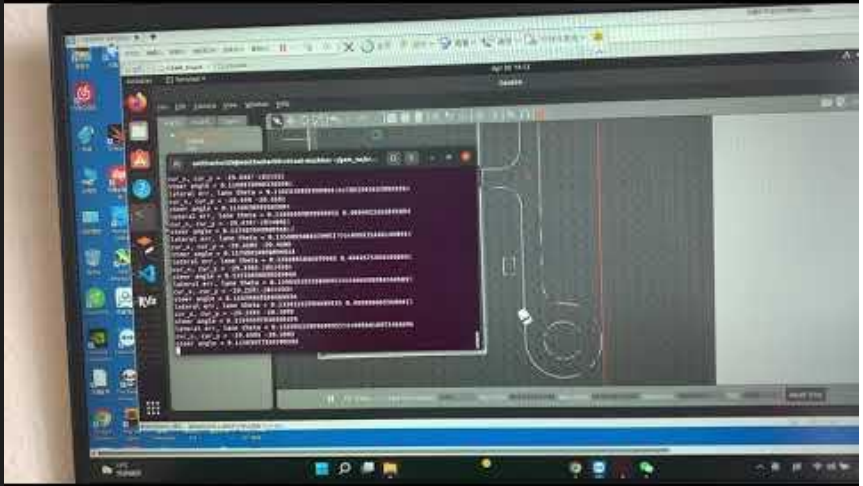




A-star algorithm with a polynomial

Code and simulation

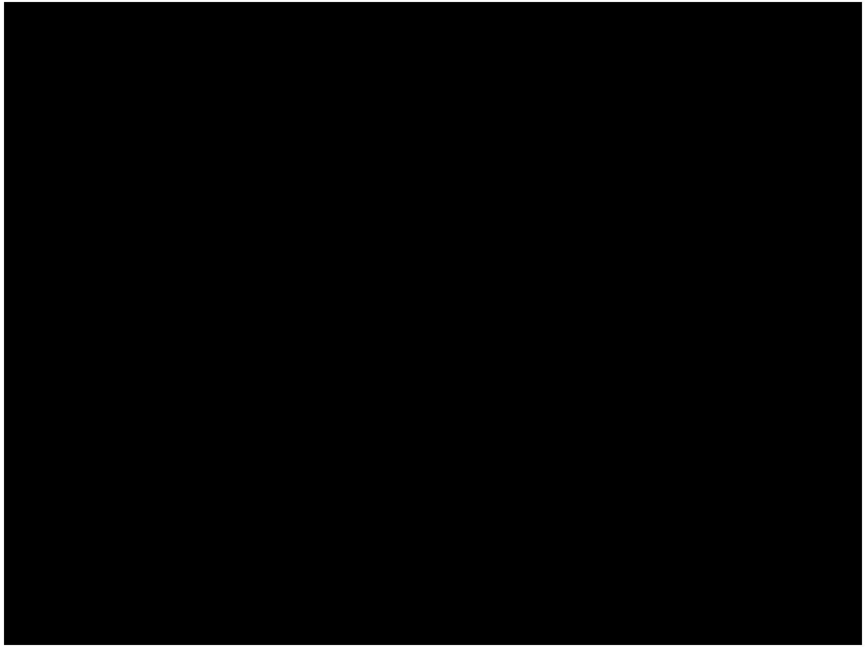
```
##################################################################
p_y = [-0.0263, -1.001, -27]
polynomial_y = p_y[0]*cur_x**2 + p_y[1]*cur_x + p_y[2]
lateral_error = polynomial_y - cur_y
frontcar_x = cur_x + 0.5
slope = p_y[0]*2*frontcar_x + p_y[1]
lane_theta = math.atan(slope)
if abs(-19.5-cur_x)>0.5
    ky = 1.0
    k_theta = 0.1
    velocity = 1.5
    steering_angle = ky*lateral_error + k_theta*lane_theta
    self.ackermann_msg.speed = velocity
    self.ackermann_msg.steering_angle = steering_angle
    self.ackermann_pub.publish(self.ackermann_msg)
else:
    self.ackermann_msg.speed = 0.0
    self.ackermann_msg.steering_angle = 0.0
    self.ackermann_pub.publish(self.ackermann_msg)
self.rate.sleep()
#####################################################################
```



A-star algorithm with a polynomial

Demo on GEM

- Generate a 2nd order polynomial
- Design a PD controller
- Transform coordinate system



Codes:
https://github.com/amithachari/ECE484_Final_Project.git

Videos:
https://youtu.be/QUgbshpXyBY

Future Work

- Implement advanced control strategy: Model Predictive Control
- Add perception module to detect cones and avoid them
- Make the system more robust and collect more test results because RRT-star with Dubins Path cannot return a solution every time

References

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

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

Thank you!