RBE550: Motion Planning

Valet – Report

Chaitanya Sriram Gaddipati

Vehicle Kinematics:

I implemented three types of vehicles. Their kinematics are discussed below.

Delivery Robot:

The delivery robot has a simple differential drive or skid steer kinematics model as shown below. This model controls the robot by controlling the wheel velocities.

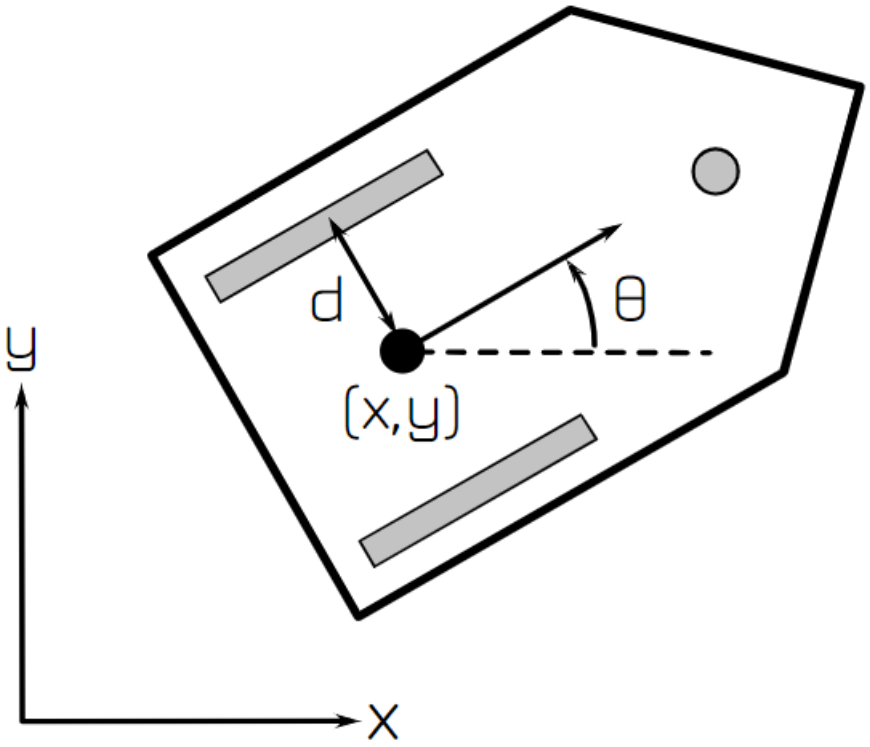

Figure 4: Differential drive (or skid steer) kinematics

The action inputs of the robot are the wheel velocities of the right and left wheels U_r and U_l respectively. The wheel radius r is 0.5m and the wheel base L is 2m. The system model at each time step is:

$$x = x_i + \frac{r}{2}(U_l + U_r)\cos(\theta)\Delta t$$

$$y = y_i + \frac{r}{2}(U_l + U_r)\sin(\theta)\Delta t$$

$$\theta = \theta_i + \frac{r}{L}(U_r - U_l)\Delta t$$

Car:

The car has an Ackerman steering kinematics as shown below. For this model the controls I have chosen are the car velocity and steering rate.

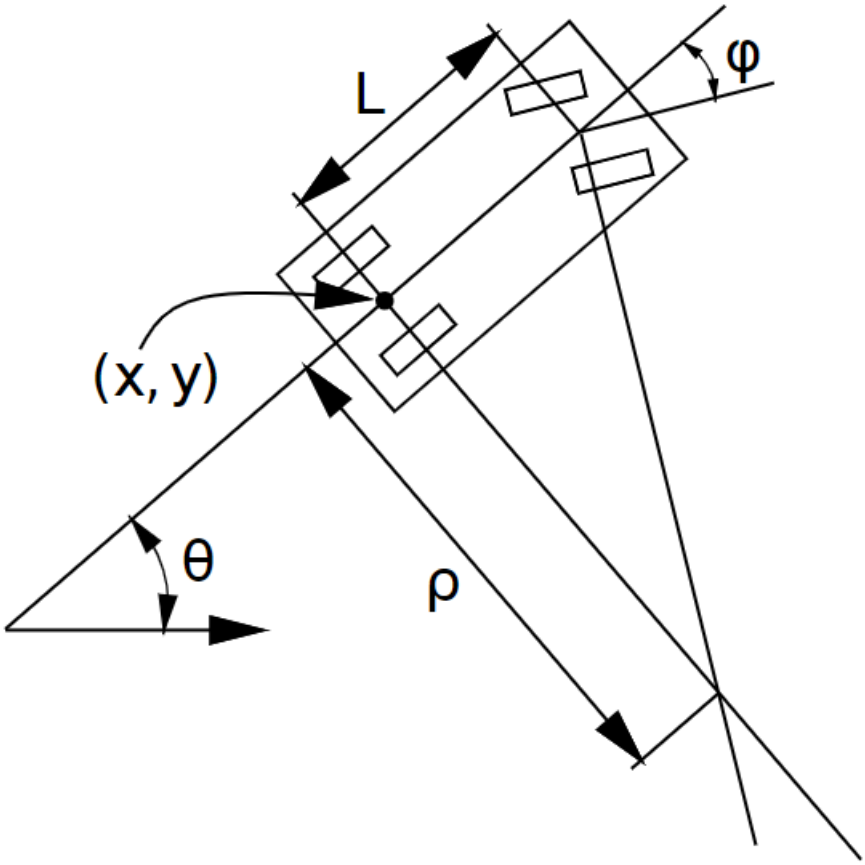

Figure 6: Ackerman steering kinematics

The action inputs to the model are car velocity V and steering rate ω. The wheelbase L is 2.8m as given in the assignment. The system model at each time step is:

$$x = x_i + V\cos(\theta)\Delta t$$

$$y = y_i + V\sin(\theta)\Delta t$$

$$\theta = \theta_i + \frac{V}{L}(\tan(\delta))\Delta t$$

$$\delta = \delta_i + \omega\Delta t$$

Here the state/configuration of the car at each time step is given as $[x, y, \theta, \delta]$.

Truck Robot:

The truck has a single trailer attached to a car that has Ackerman kinematics. The trailer do not have steering capabilities. The figure is shown below.

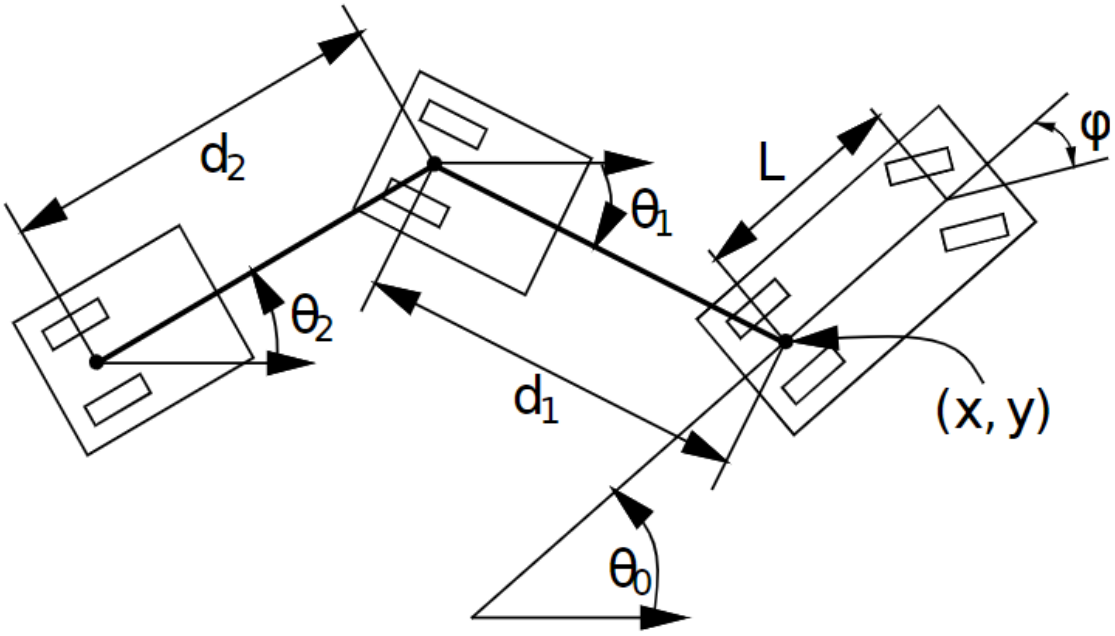

Figure 8: Trailer kinematics

The action inputs to the truck are the same as that of car model i.e. velocity of the car V and the steering rate δ. The wheelbase of the car part L is 3m and the distance between the rear axle of the truck and axle centre of the trailer d_1 is 5m. The system model at each step is:

$$x = x_i + V\cos(\theta)\Delta t$$

$$y = y_i + V\sin(\theta)\Delta t$$

$$\theta_0 = \theta_{0_i} + \frac{V}{L}(\tan(\delta))\Delta t$$

$$\delta = \delta_i + \omega\Delta t$$

$$\theta_1 = \theta_{1_i} + \frac{V}{d_1}(\sin(\theta_0 - \theta_{1_i}))\Delta t$$

Here the state/configuration of the car at each time step is given as $[x, y, \theta_0, \delta, \theta_1]$.

Approach and Planner algorithm design:

I implemented an A* search algorithm for planning. For this search I utilized motion primitives to expand the search tree. By using state lattice approach, I discretized the continuous space. The graph starts at the initial node which is the initial state of the vehicle and keeps exploring by calculating the future states by considering the action/control inputs from the set of motion primitives and applying the vehicle kinematics to it for each time step. The new nodes/states are stored in a Priority queue from which we try to pick the state with least cost. The cost function differed from one vehicle model to other, but it typically had the Euclidean distance from the start to the current state (g), a heuristic 'h' that measures the Euclidean distance from the current state to the goal, the difference in the vehicle orientation (θ) and the vehicle steer angle (δ). Each of the individual costs have

specific weights for each model. These weights are obtained by experimentation and are fixed. To save the computational time, I have initially chosen a small action/control input set (motion primitives) and as we are at a certain distance closer from the goal the action input set is increased. This way the path till certain distance from the goal is calculated quickly and as we approach the goal more states are explored in the state lattice i.e. the state lattice becomes finer as we go near the goal (this idea was an inspiration from the KD tree structure). The goal is reached when the current state of the robot is within certain limits.

Algorithm 22 CAR_GRID_SEARCH

Input: Start configuration q_{start}, goal region $\mathcal{G}(q_{\text{goal}})$

Output: A path from q_{start} to $\mathcal{G}(q_{\text{goal}})$ or FAILURE

```
Initialize search tree T and list OPEN with start configuration q_start
while OPEN not empty and size(T) < MAXTREESIZE do
  q ← first in OPEN, remove from OPEN
  if q ∈ G(q_goal) then
    Report SUCCESS, return path
  end if
  if q is not near a previously occupied configuration then
    Mark q occupied
    for all actions in {L±, R±, S±} do
      Integrate forward a fixed time to q_new
      if path to q_new is collision-free then
        Make q_new a successor to q in T
        Compute cost of path to new configuration q_new
        Place q_new in OPEN, sorted by cost
      end if
    end for
  end if
end while
Report FAILURE
```

My code is developed based on the above pseudocode taken from page 454 of the book "Principles of robot motion". In the above code the collision checking is done by rejecting all the states where the centre of the rear and front axle of the vehicle is in certain distance from the obstacles. In my implementation I have used a dictionary called 'search_tree' that keeps track of the neighbours of a state. A dictionary called 'Prev' contains the current state as key and the previous state as its value. A list of all visited nodes is also made. Wherever necessary an if statement is used to differentiate between different vehicle models.

Results/ Path Depiction:

The images of the vehicles in an obstacle map are attached below for each vehicle. The cyan line shows the kinematic path taken by the vehicle from the start to the goal location which is in between the red vehicle obstacles. In the picture the goal pose of the robot for a given goal of (110,30) is also shown.

For delivery robot:

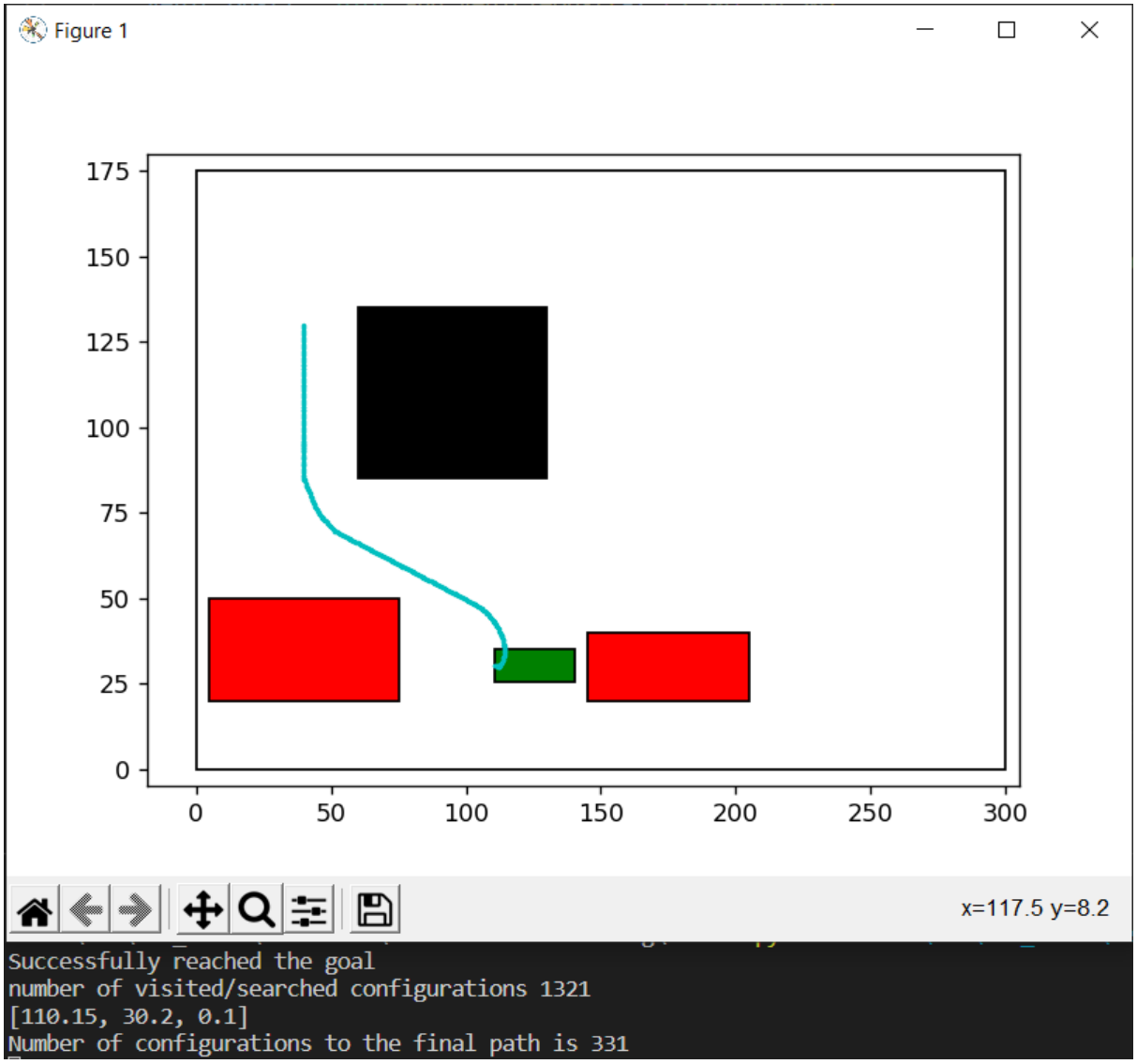
Figure 1
175
150
125
100
75
50
25
0
0
50
100
150
200
250
300
x=117.5 y=8.2
Successfully reached the goal
number of visited/searched configurations 1321
[110.15, 30.2, 0.1]
Number of configurations to the final path is 331

For car:

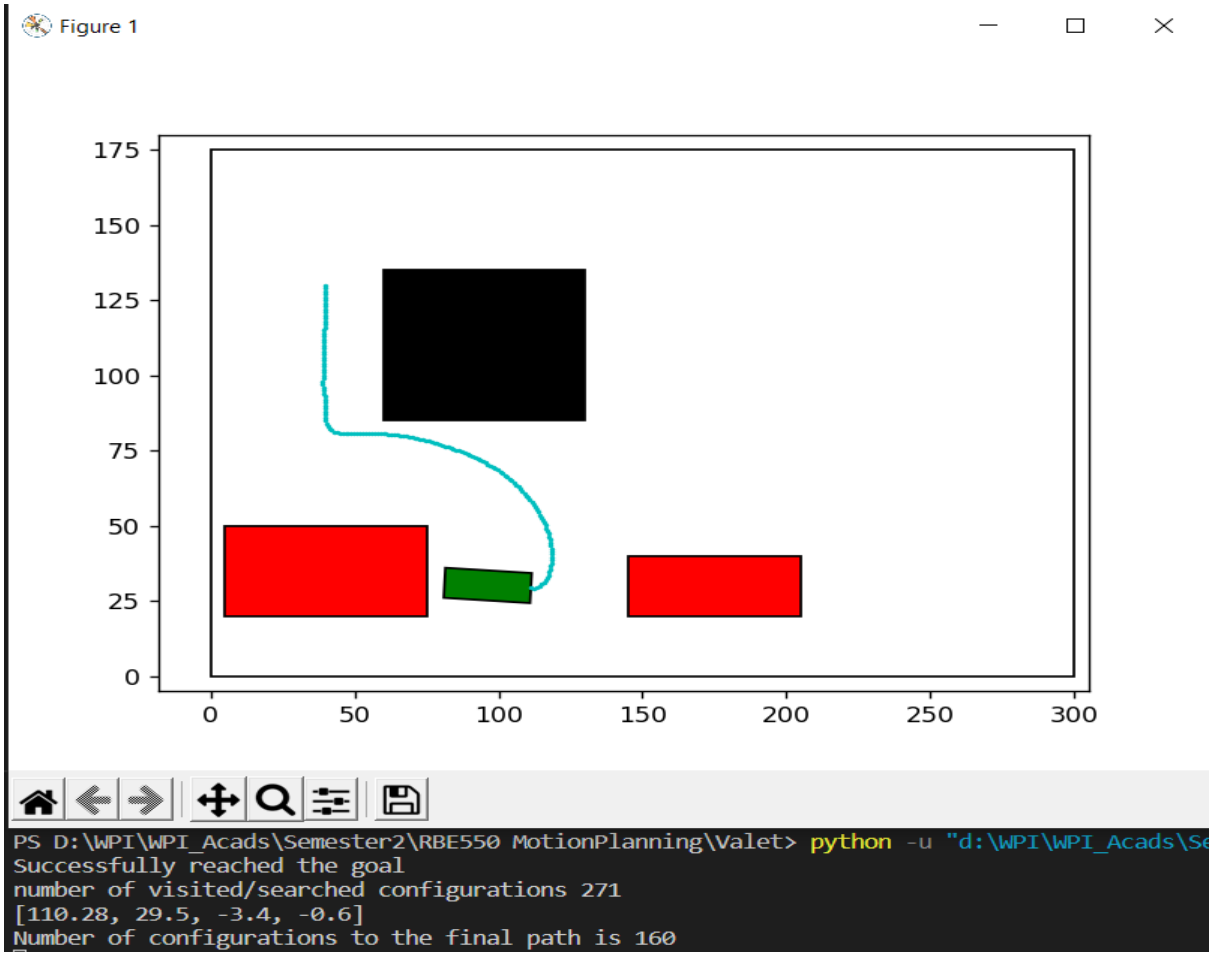
Figure 1
175
150
125
100
75
50
25
0
0
50
100
150
200
250
300
PS D:\WPI\WPI_Acads\Semester2\RBE550 MotionPlanning\Valet> python -u "d:\WPI\WPI_Acads\Se
Successfully reached the goal
number of visited/searched configurations 271
[110.28, 29.5, -3.4, -0.6]
Number of configurations to the final path is 160

For truck:

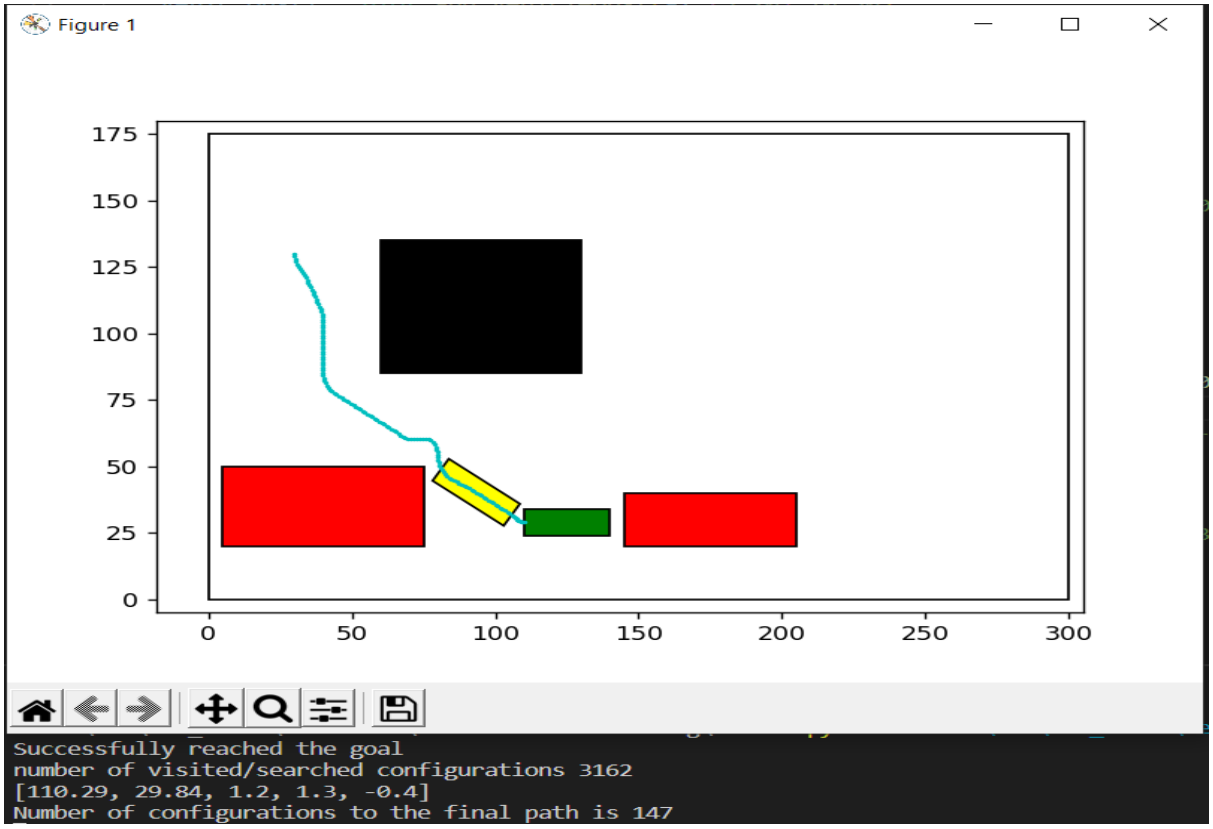
Figure 1
175
150
125
100
75
50
25
0
0
50
100
150
200
250
300
Successfully reached the goal
number of visited/searched configurations 3162
[110.29, 29.84, 1.2, 1.3, -0.4]
Number of configurations to the final path is 147

Conclusion:

During the implementation of the assignment various challenges are encountered. The collision checking of the vehicle needs to be improved as you can see in the videos some of the edges of the vehicles sometimes grazes the obstacles. This is because we are doing the collision check for the front and rear axles of the vehicle but not for the whole body shape. Another difficulty is in finding proper weights for the cost function of the search algorithm that ensures the vehicle path is smooth. Another difficulty is faced when making the vehicle park parallel. This can be improved by the use of a controller that makes sure the vehicle reaches the goal pose. I believe A* is not a good choice of algorithm for this problem because it often searches an area more thoroughly rather than exploring different places hence resulting in a longer search trees and higher computational times.