Assignment-3

2023-03-20

Name: Varun Gampa
ID : 773686296

In the three sets of code, path planning for three different types of vehicles for parallel parking is written. All the codes utilize the hybrid A* algorithm. In this section, kinematic equations for each robot is mentioned.

Section 1: Kinematic equations and control inputs

DELIVERY ROBOT:

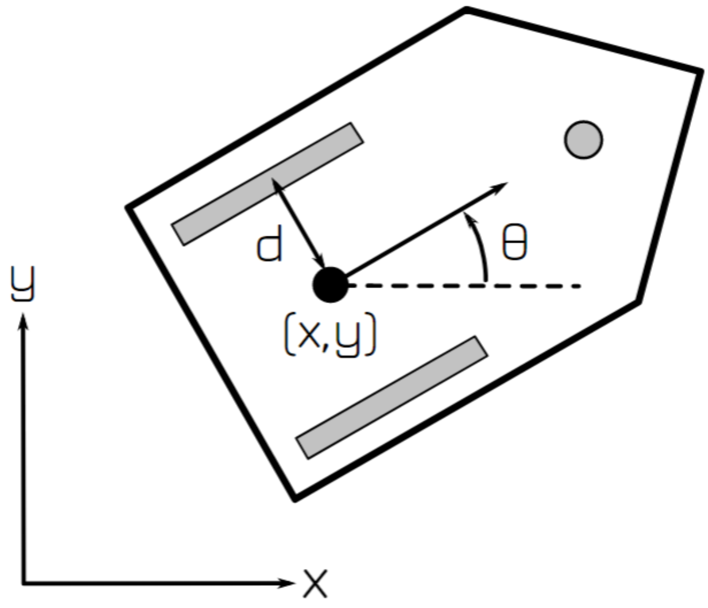


Figure 4: Differential drive (or skid steer) kinematics

Kinematic equations:

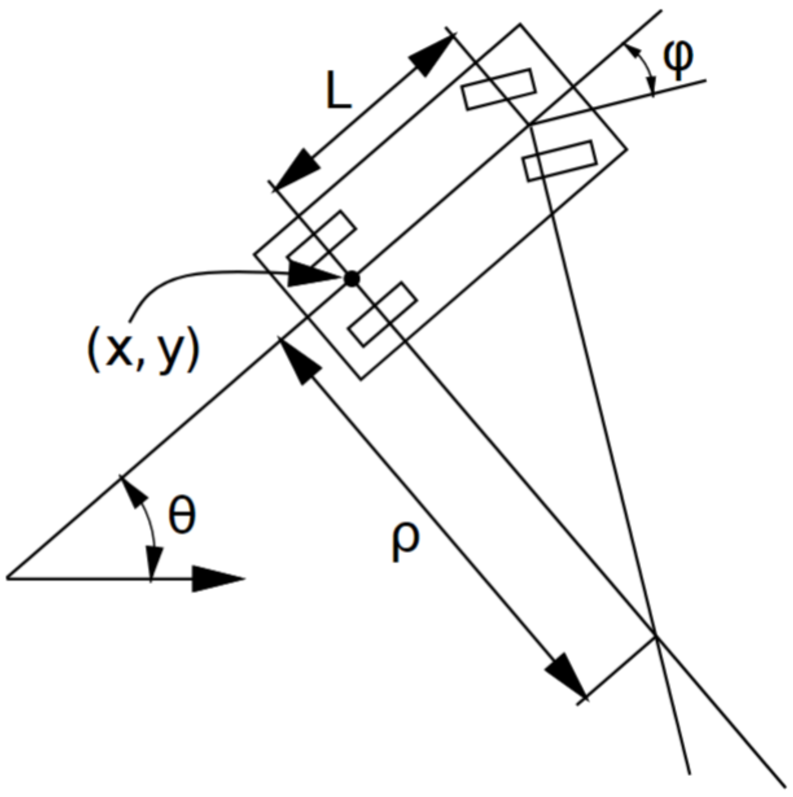
$$\dot{x} = r(u_l + u_r)sin(\theta)\dot{\theta}$$
$$\dot{y} = r(u_l + u_r)cos(\theta)\dot{\theta}$$
$$\dot{\theta} = \frac{r}{L}(u_r - u_l)$$

Here u_r, u_l are angular velocities of the wheel which are control inputs.

$r = 0.5m$, $L = 2m$

Control input chosen for wheel angular velocities is the set (-5,5) in steps 1.

THE CAR



Kinematic equations:

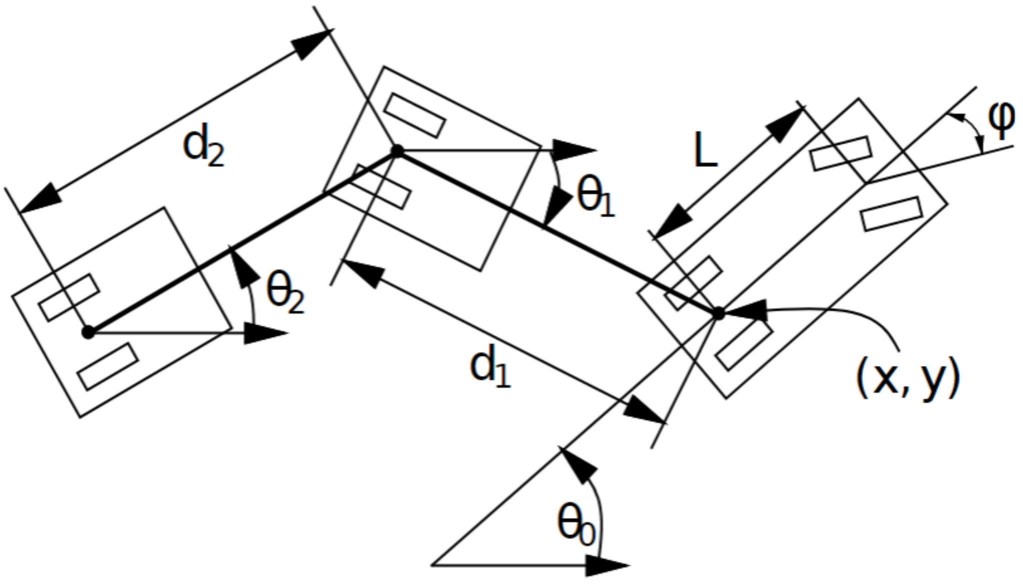
$$\dot{x} = v \cdot cos(\theta)$$
$$\dot{y} = v \cdot sin(\theta)$$
$$\dot{\theta} = \frac{v}{L} \cdot tan(\phi)$$

Here v and ϕ control inputs denoting velocity and steering angle of the car.

v is either -5 or 5, ϕ is in the range (-30,30) in steps of 4 degrees

$L = 2.8m$

THE TRUCK



Kinematic equations:

$$\begin{aligned}\dot{x} &= v \cdot cos(\theta)\\ \dot{y} &= v \cdot sin(\theta)\\ \dot{\theta_0} &= \frac{v}{L} \cdot tan(\phi)\\ \dot{\theta_1} &= \frac{v}{d_1} \cdot sin(\theta_0 - \theta_1)\end{aligned}$$

Here $v \text{ and } \phi$ control inputs denoting velocity and steering angle of the car.

v is in range(-4,4) in steps of 1 m/s , ϕ is in the range (-60,60) in steps of 4 degrees

$L = 3.0m, d_1 = 5m$

Section 2: Hybrid A* Algorithm implementation

As mentioned hybrid A* is applied to all vehicles and its pseudo code is presented here.

In the A* star algorithm an empty priority queue is initialized which is the list of nodes to visit and another empty list is initialized which stores the nodes visited. In the list of nodes to visit initial point is added with cost zero.

When the goal is found , then the control inputs stored to reach the goal node is added to the bot and simulated forward which is seen.

Subsection: Heuristics used

Delivery Bot:

For the delivery bot, the heuristic is Euclidean distance of the start point and the goal point, added with absolute value of difference in angle between goal orientation and the current state's orientation multiplied by a scaling factor.

Python implementation:

```
np.sqrt((test_state[0] - goal_state[0])**2 + (test_state[1] -
goal_state[1])**2 ) + 0.1*(min(np.absolute((test_state[2] -
goal_state[2])*np.pi/180),2*np.pi - np.absolute((test_state[2] -
goal_state[2])*np.pi/180)) )
```

The Car

For the car **Reeds Shepp** shortest path's distance is used. The Reeds Shepp distance is the shortest distance between the initial position along with the configuration and the final position along with the configuration. The reference for the same is attached in the references section. The same has been implemented based on external library, which is also referred to.

Python implementation

```
reeds_shepp.path_length((test_state[0],test_state[1],test_state[2]*np.pi/18
0), (goal_state[0],goal_state[1],goal_state[2]*np.pi/180), 2.8*np.sqrt(3))
+ 1*np.exp(-((test_state[0] - 17.5)**2 + (test_state[1] - 12.5)**2)) +
1*np.exp(-((test_state[0] - 12.25)**2 + (test_state[1] - 20)**2))  +
1*np.exp(-((test_state[0] - 23)**2 + (test_state[1] - 20)**2))
```

The Truck

For the truck **Reeds Shepp** shortest path's distance is used for the truck and the trailer. The Reeds Shepp distance is the shortest distance between the initial position along with the configuration and the final position along with the configuration. The reference for the same is attached in the references section. The same has been implemented based on external library, which is also referred to.

Python implementation:

```
1*reeds_shepp.path_length(np.array([test_state[0],test_state[1],
(test_state[2])*np.pi/180]), np.array([goal_state[0], goal_state[1],
(goal_state[2])*np.pi/180]),3/np.sqrt(3))  +
1.5*reeds_shepp.path_length(np.array([x_trailer,y_trailer,
(test_state[3])*np.pi/180]), np.array([x_trailer_goal, y_trailer_goal,
(goal_state[3])*np.pi/180]),5/np.sqrt(3)) + 0.5*
(min(np.absolute(test_state[3])*np.pi/180, 2*np.pi -
np.absolute(test_state[3])*np.pi/180))
```

Note that the same heuristics are used in the cost_to_go and cost_to_come, though weighted differently to compute the total cost in the priority queue.

Section 3: Results

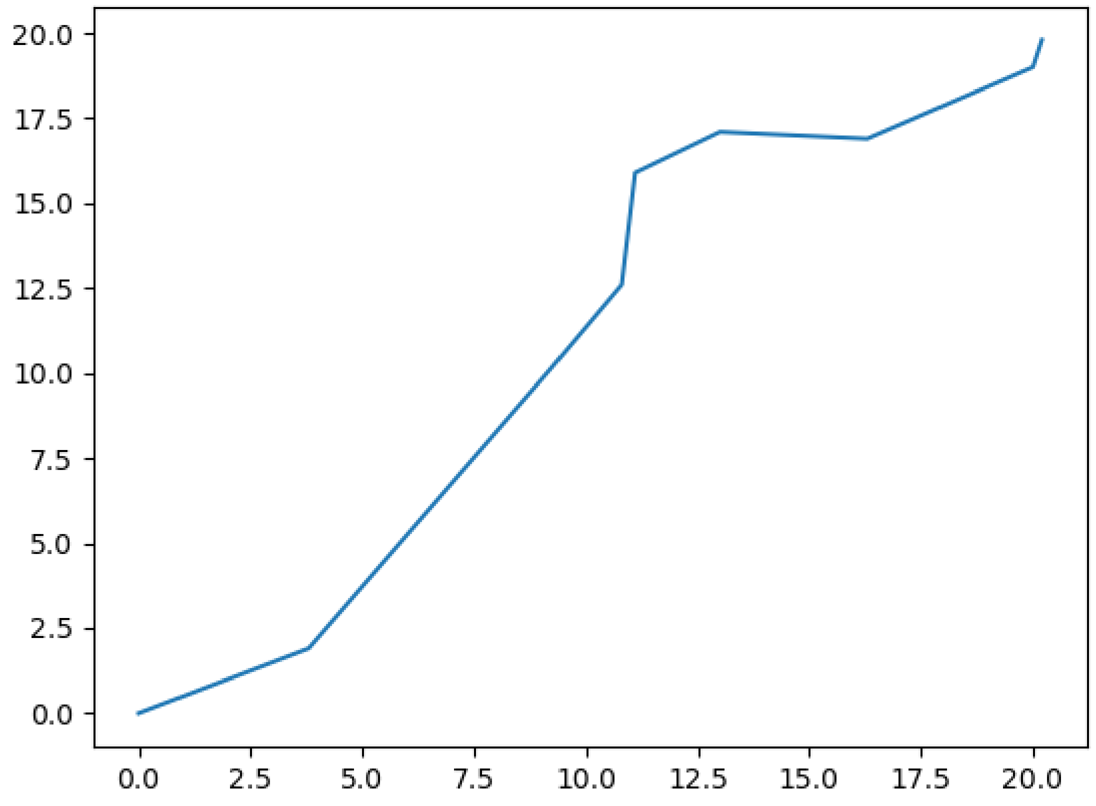
The path generated is flipped, meaning that though the vehicle moves from the north west corner to south east, the graph is flipped over the x-axis.

In all the cases , the vehicle is supposed to reach point 20,20 and face east.

Note that for simulation the scaling factor is 24, that is all lengths are scaled by 24.

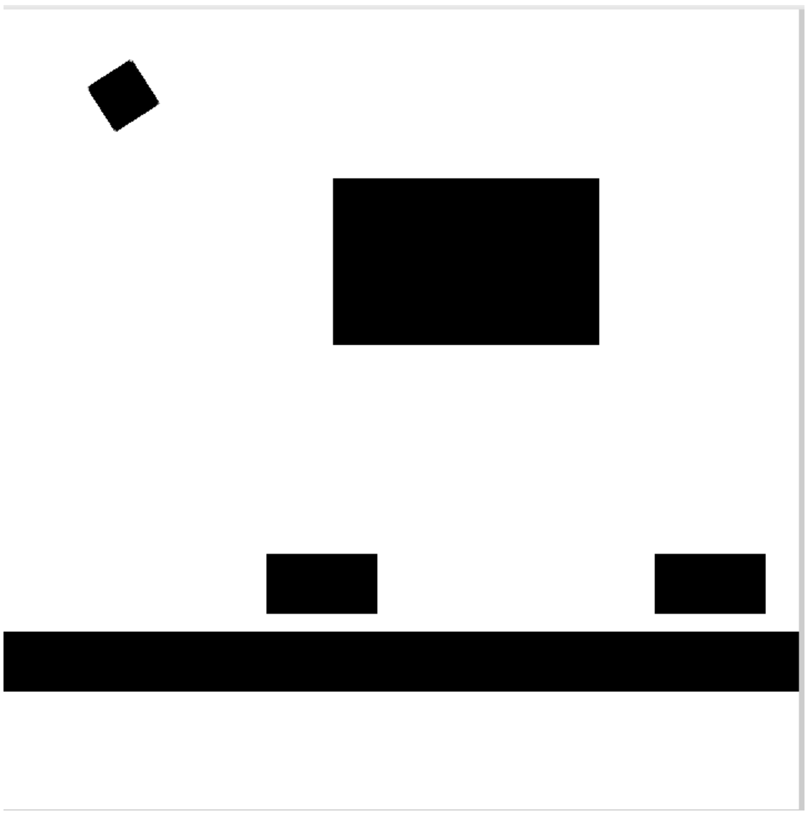
Delivery Bot:

Path generated:

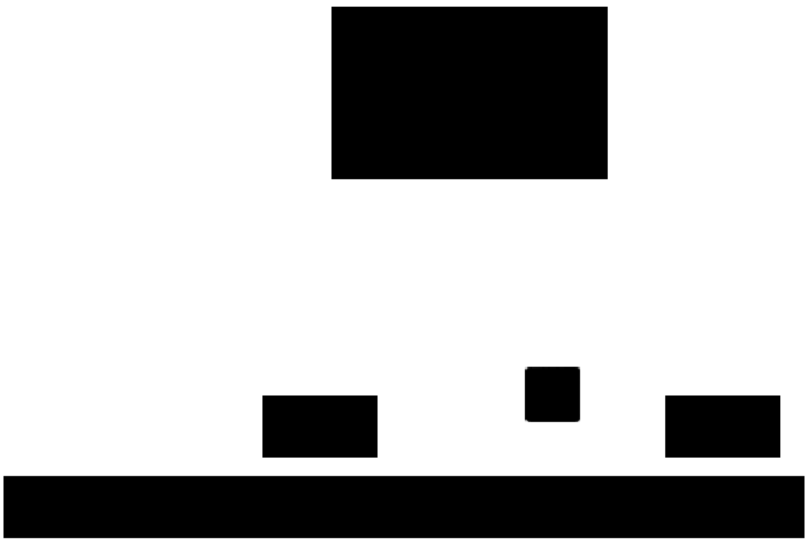


Time taken to generate path: 0.4190993309020996

Initial condition:



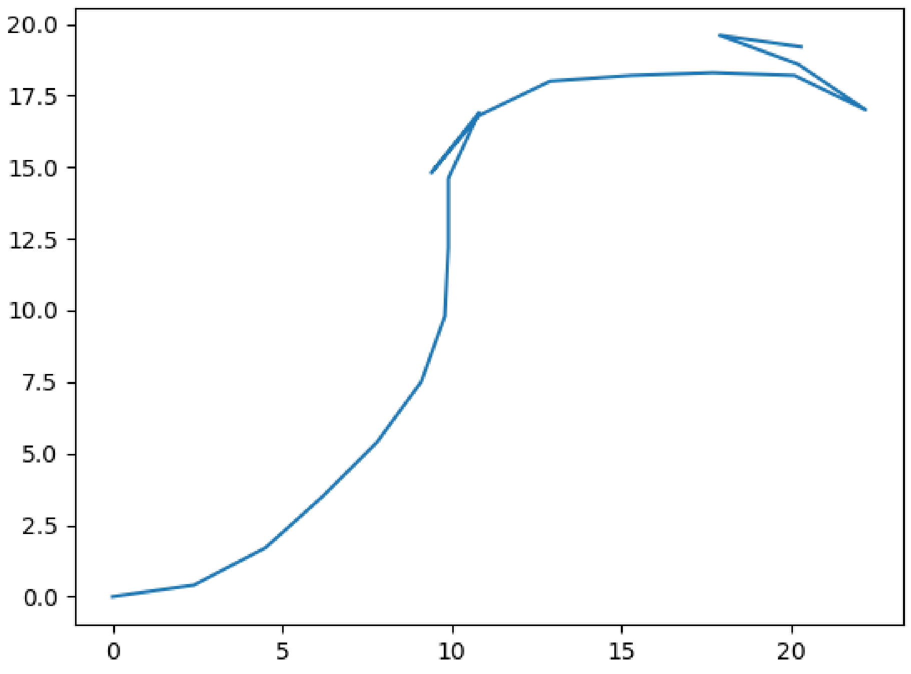
Final position:



Final position: 20.5, 20.37

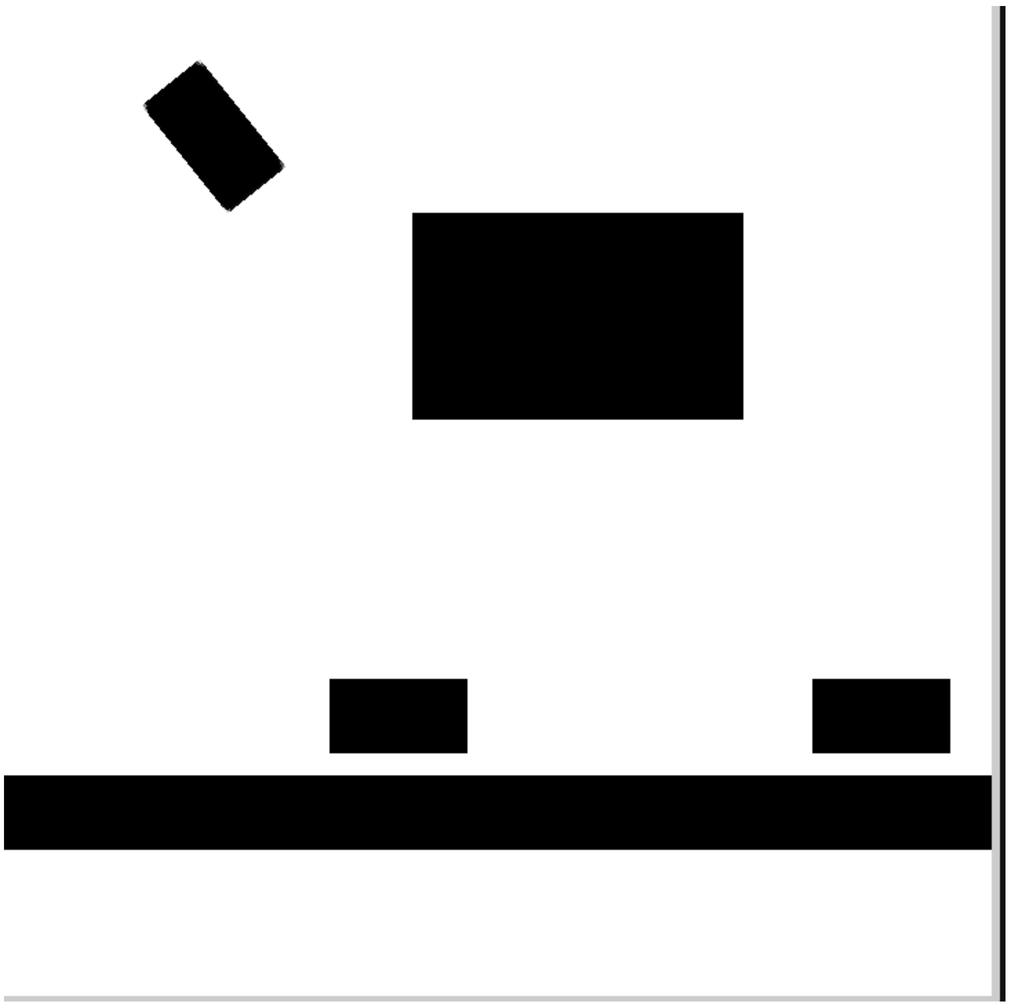
The Car:

Path generated :

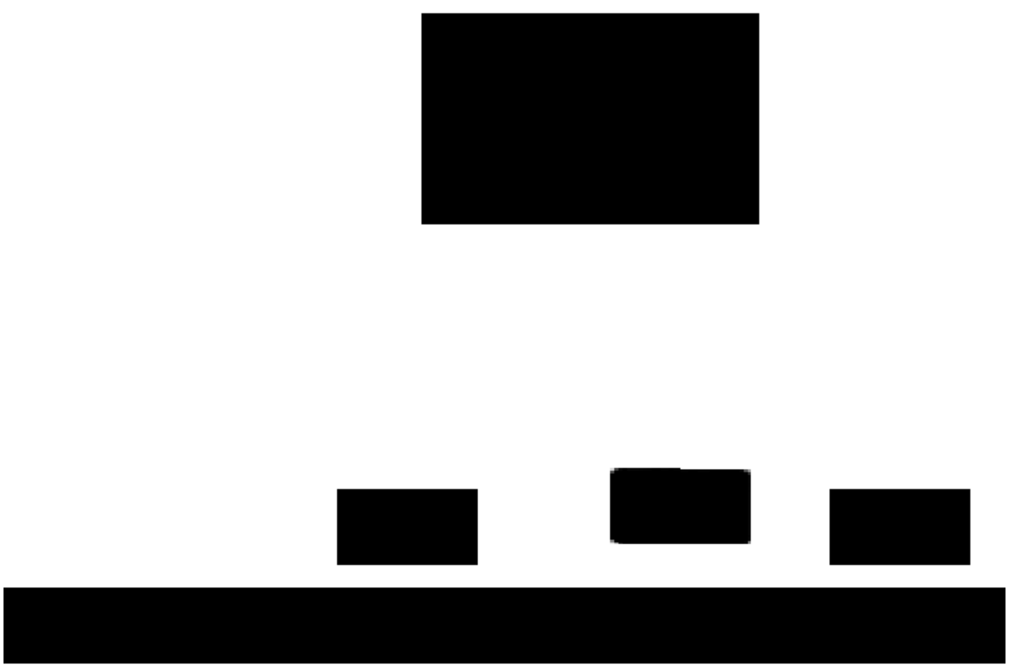


Time taken to generate the path: 6.6705427169799805s

Initial position:



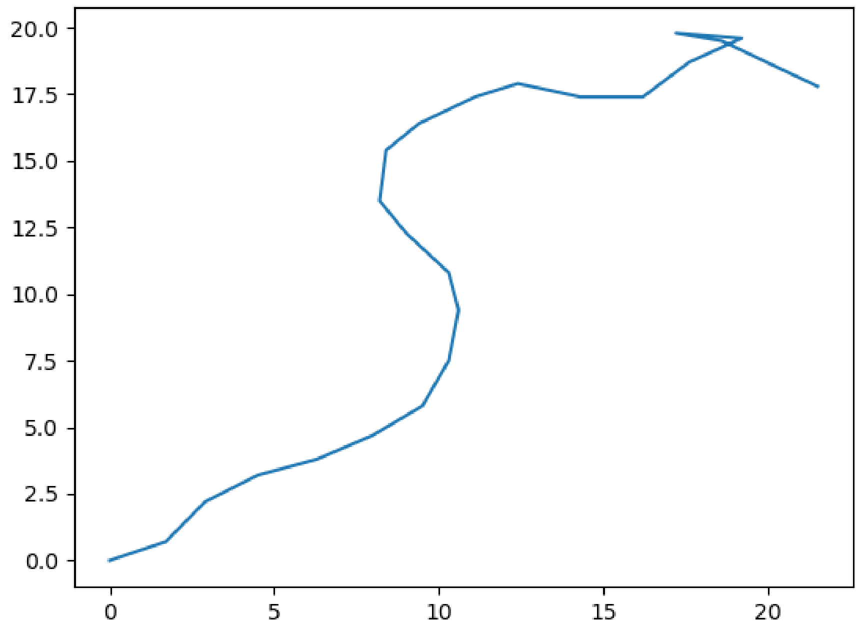
Final position:



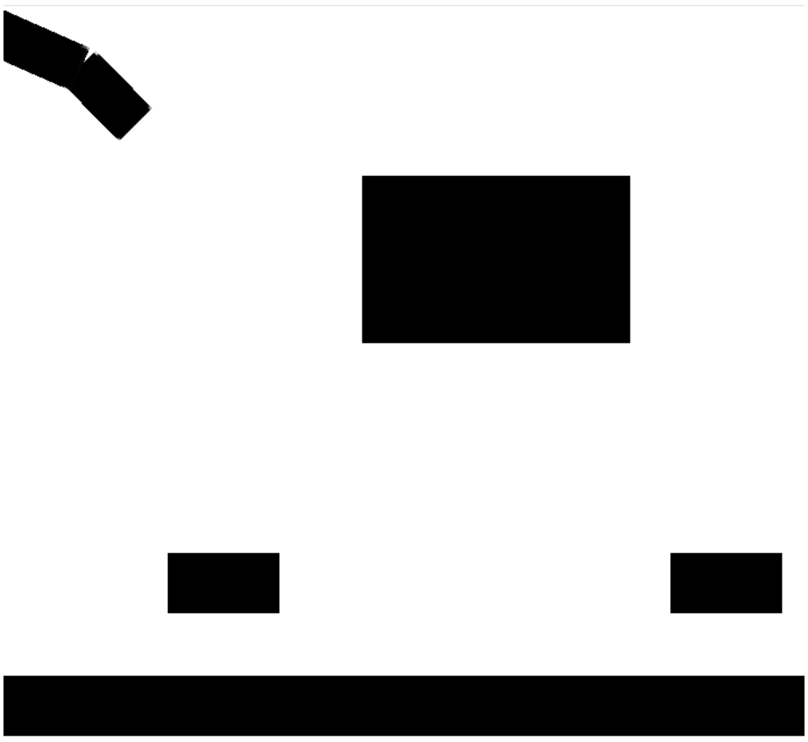
Final position: 20.1, 20.9

The Truck

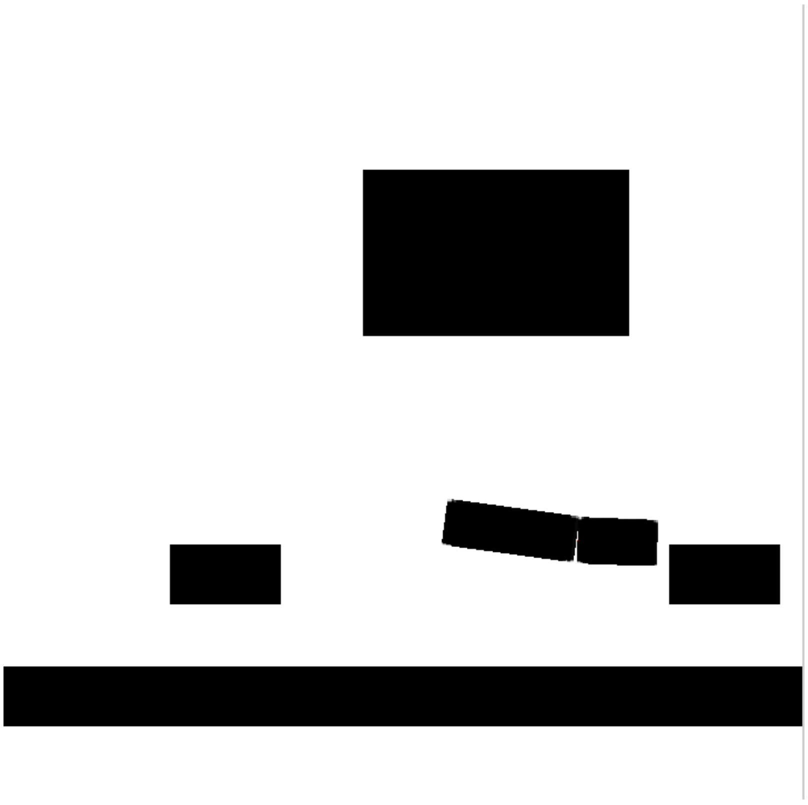
Path generated:



Time taken to generate the path: 116.97623085975647



Final position:



Final position: 21.5, 20.3

References:

1. https://github.com/ghliu/pyReedsShepp (library for Reeds Shepp)
2. http://lavalle.pl/planning/node822.html