Scott Pena
October 29, 2024
Valet Report

Introduction

This assignment required the development of a kinematic planner under nonholonomic constraints to control and park several vehicles. Additionally, a virtual environment should be created to demonstrate the performance and result of the planner, accounting for any collisions between the vehicle and any obstacles within the environment.

Creating the Environment

The first task completed for this assignment was to create a virtual environment to test the kinematic planners. To accomplish this, I utilized pygame to create polygons that represent surfaces and obstacles within a GUI. By using pygame, obstacles became very simple to add, modify, and remove to customize the environment. By importing an image, I can easily modify its position and rotation, therefore acting as the robot within the environment. Pygame also has built-in functionality to determine if two rectangles are colliding, which is used to help the kinematic planner generate a configuration space by determining if the robot will collide with an obstacle at various positions and rotations.

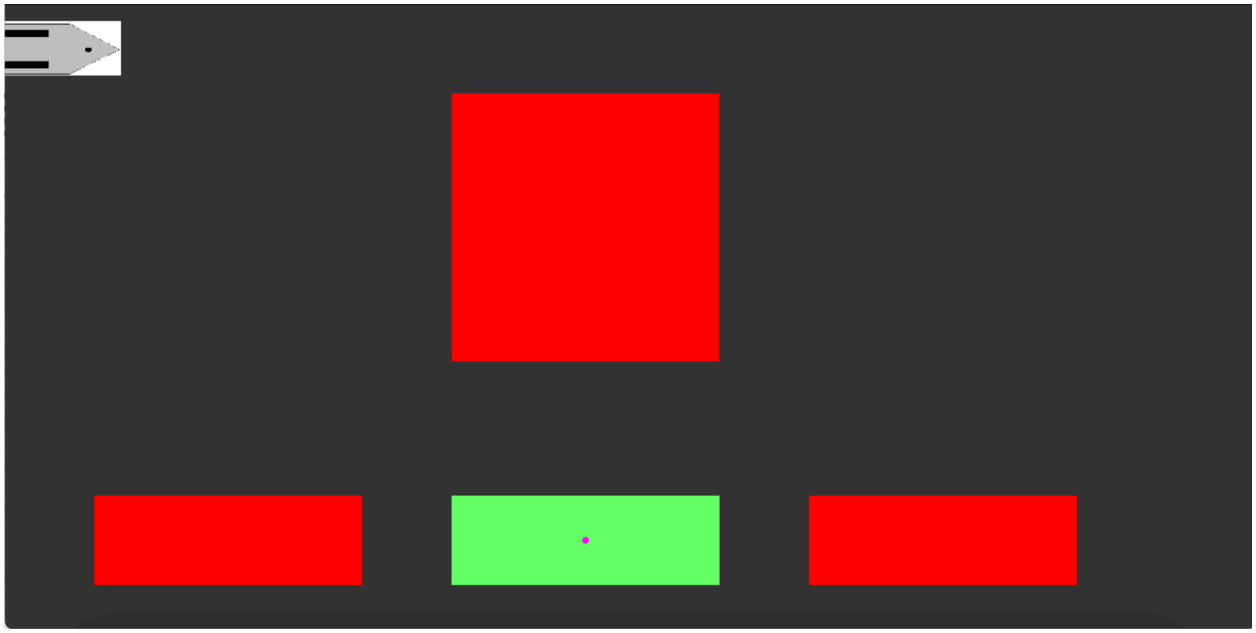


Figure 1: Environment built in pygame to perform kinematic planning upon. Red rectangles denote obstacles, the green rectangle is the goal, and the image in the top left corner is the robot to be controlled.

Kinematic Planner

The main deliverable for this assignment is the kinematic planner. The planner operates by using a priority queue to store robot states to explore. When a state is popped from the list, the planner first checks if the robot state is the goal node by checking if the position of the state matches the goal point. If the state is not the goal, the planner continues its search. The planner

then generates a list of robot velocities and steering angles and parses through each pairing to calculate a new robot state from these control inputs via the robot's kinematic equations. If the control input sends the robot to an already visited pixel or causes the robot to collide with an obstacle, then the state is not added to the priority queue. Otherwise, the new state is added to the priority queue with its calculated cost. The cost of each state depends on the control input and its euclidean distance from the robot state's position to the goal position. The control input cost is calculated based on if the control input (robot velocity and steering angle) changes from the previous state to the newly calculated state.

Delivery Bot Kinematics

The food delivery bot is a simple skid-steer drive robot with diwheel kinematics. To stay consistent with the other two vehicles, I decided to have the control inputs be the robot's velocity and the steering angle of the robot. Since the wheels on the delivery bot do not turn, the steering angle calculates the speed of both the left and right wheels to achieve the turn radius that would be defined by the steering angle. Below are the kinematic equations used in the kinematic planner.

$$R = \frac{L}{2 * \tan \psi}$$
$$\omega = \frac{v}{R}$$
$$v_r = v - \frac{L\omega}{2}$$
$$v_l = v + \frac{L\omega}{2}$$

Equations to calculate each wheel's velocity based on the control inputs: robot velocity (v) and steering angle (psi)

$$\theta' = \theta_0 + dt\omega$$
$$x' = x_0 + dt(v \cos \theta_0)$$
$$y' = y_0 + dt(v \sin \theta_0)$$

Kinematic equations to calculate the new angle and position of the delivery robot.

Police Car Kinematics

The old police car is a four wheeled robot that is controlled with Ackerman steering. The control inputs are the car's velocity and the acceleration of the steering angle of the front two wheels. Since the ackerman steering limits the turning radius of the car, the implementation limits the steering angle to -20 and 20 degrees. Below are the kinematic equations.

$$\psi' = \psi_0 + dt\omega$$
$$\theta' = \theta_0 + dt(v\frac{\tan(\psi_0)}{L})$$
$$x' = x_0 + dt(v\cos\theta_0)$$
$$y' = y_0 + dt(v\sin\theta_0)$$

Kinematic equations to determine the robot's steering angle, orientation, and position from the control inputs.

Trailer Kinematics

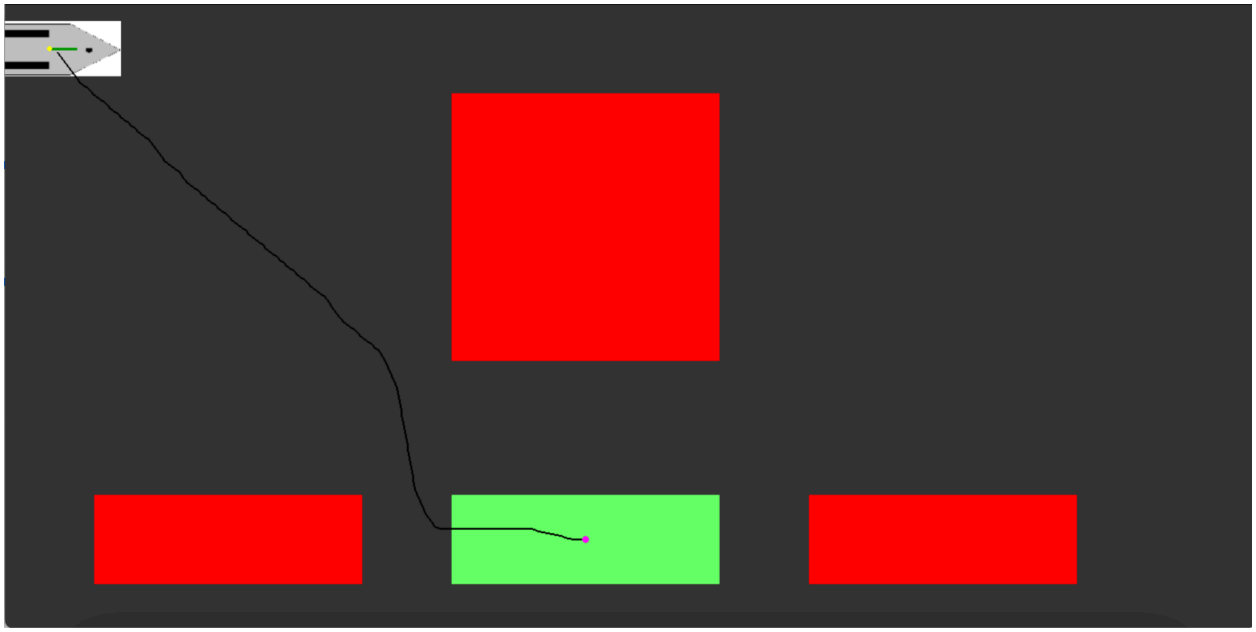
The trailer is a combination of an Ackerman steering robot and a diwheeled robot attached 5.0 meters away. After doing some research, I was able to find a resource to help compute the relationship between the truck and the angle of the trailer [1]. After calculating the angle of the trailer relative to the truck, we can use simple trigonometry to calculate the trailer's position, since the length of the trailer hitch is constant (5.0 meters). Below are the kinematic equations:

$$\dot{x} = s\cos\theta_0$$
$$\dot{y} = s\sin\theta_0$$
$$\dot{\theta}_0 = \frac{s}{L}\tan\phi$$
$$\dot{\theta}_1 = \frac{s}{d_1}\sin(\theta_0 - \theta_1)$$

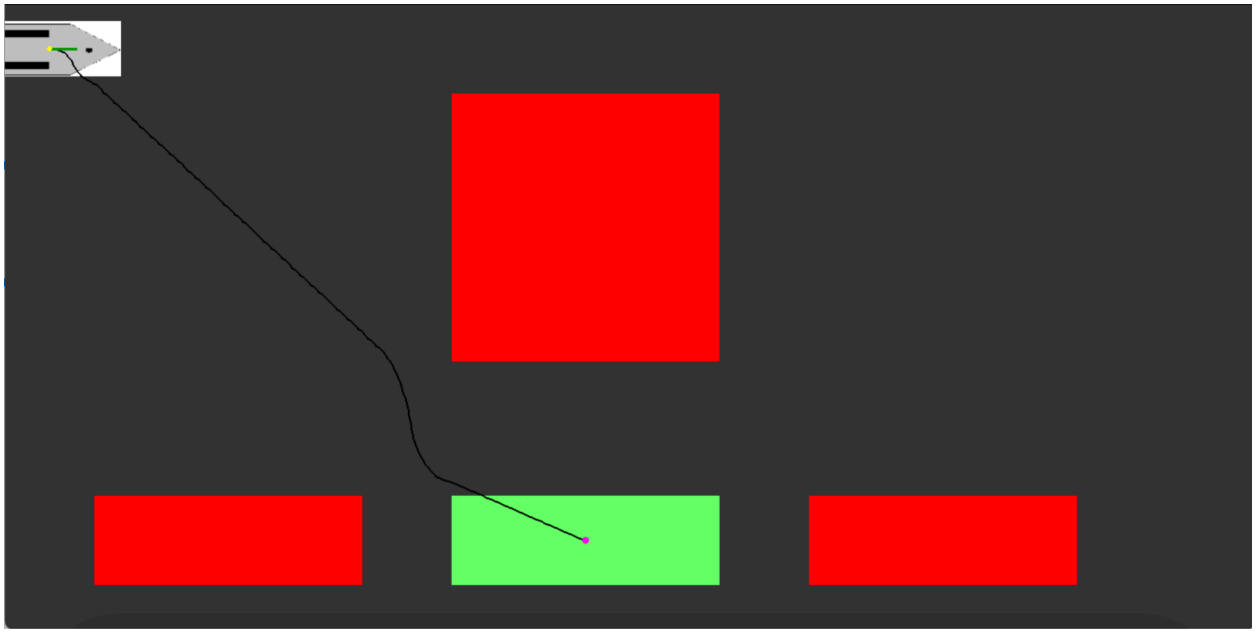
Kinematic equations to calculate the orientation and position of the trailer behind the truck. The truck itself uses the kinematic equations shown for the police car, as both are controlled using Ackerman steering.

Results

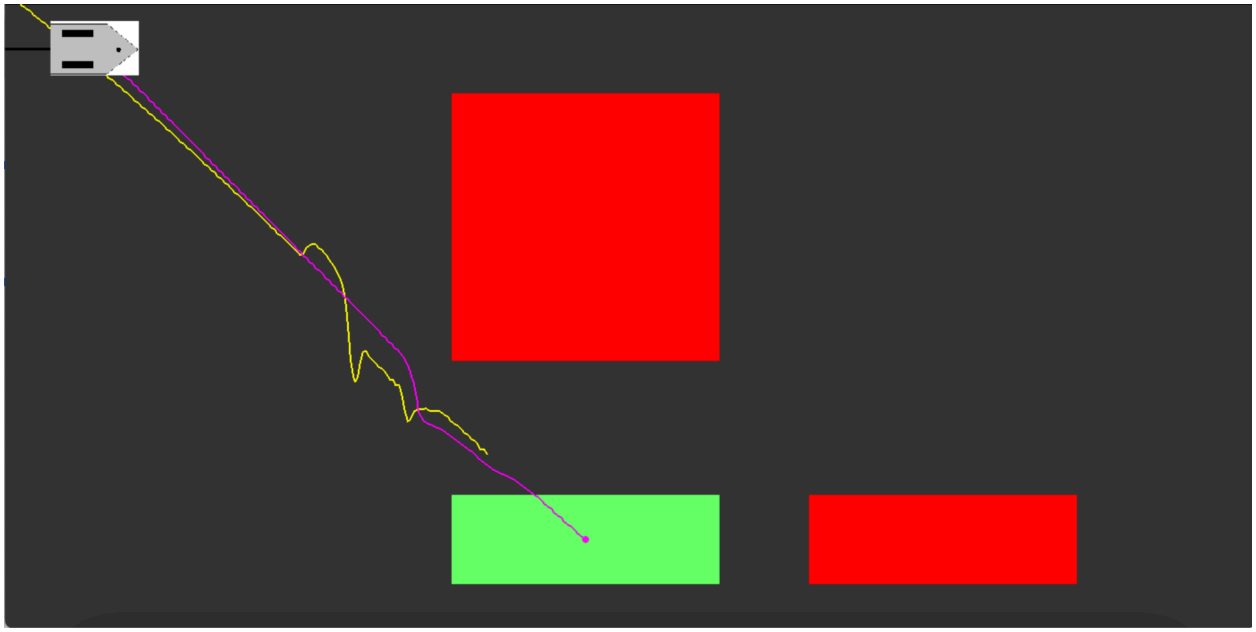
After creating the virtual environment in pygame and implementing the kinematic calculations for each vehicle in appropriate Python classes, I was able to plot all three motion paths to park each vehicle.



Path generated for the delivery bot.



Path generated for the police car.



Path generated for the truck and trailer. The purple path depicts the path of the truck, and the yellow path depicts the path of the trailer. This planner does require more tuning and troubleshooting to correct any bugs for possible kinematic errors or algorithm bugs. Unfortunately, I ran out of time to fully develop this planner before the October 28 deadline. Personally, I feel motivated to continue to debug this planner to have it operate smoothly (maybe a winter break project?)

Appendix A: Flowchart of the kinematic planner