CS 747 : Assignment 3
Autonomous Car Control

Shivam Patel, 200070077

November 21, 2022

1 The Parking Lot Problem

1.1 Problem Description

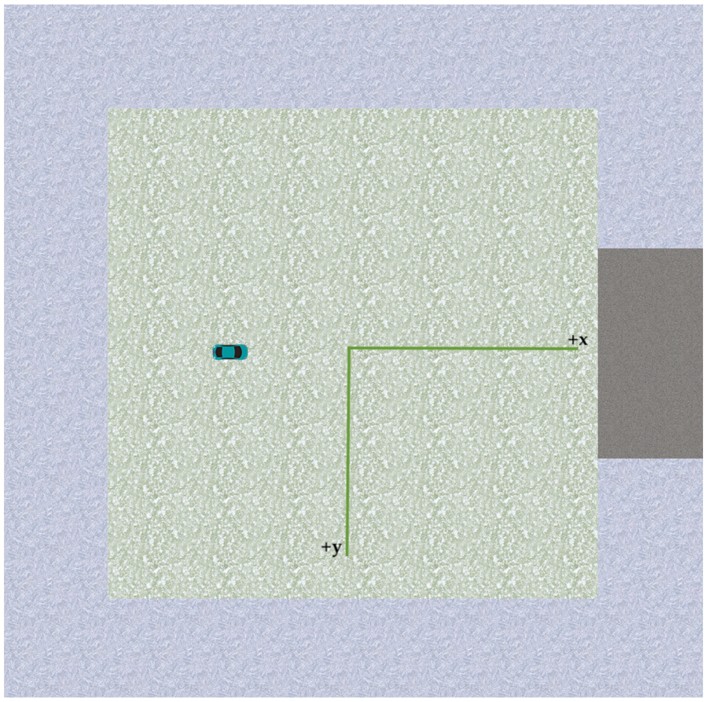

Figure 1: Ice Parking Lot Structure

Our task is to navigate your 50 x 25 car out of a 700 x 700 square grid. It must exit onto the road whose entrance has its centre located at (350, 0). This grid is centred at the origin with the coordinate axes shown below (screen size: 1000 x 1000). It is an episodic task that begins with your car initialised at a random position and orientation; the episode ends when you navigate out or bump into a wall (or pit in Task 2). We also cap episodes to a total of 1000 time steps. We have five commands for controlling the acceleration of the car, and three commands for controlling the steer.

1.2 Failed Attempts

For applying Reinforcement Learning (policy search), we need to define an MDP and states for the state of the car. Discretising the region and providing appropriate reward functions depending on the proximity to the target. the Learning Problem was ineffective and time consuming during training, as the reward for staying in ice was -1, irrespective of proximity to target. A successful park gave 100 points, and wandering out of the lot gave a -100 point penalty.

Due to the uniformity of rewards for each direction of motion, the algorithm takes a lot of time to learn about the direction of the parking lot. The final rewards are obtained only when the episode ends, i.e. when the car reaches the target or goes out of the parking space.

1.3 Present Approach

The present approach that I have implemented includes a rather straightforward brute-force method, which aligns the car initially in the direction of the target, and then moves in that direction with full acceleration. The angle is continuously being observed, and corrective steer is done when the deviation crosses a threshold.

Visually, this algorithm looks as if the car rotates on it's place towards the target and moves straight in that direction.

I have also accounted for the edge cases when the car is very close to the target axis (i.e. towards the boundary with the asphalt). In such cases, the vanilla algorithm fails, and some discretization needs to be done. I have defined regions beyond the points (300,150) and (300,-150) as 'edge-regions' in the first and fourth quadrants respectively. If a car is in these regions initially, then the target is the x axis, rather than the (350,0). So, the car moves straight towards the x axis, and has a low chance of wandering into the boundaries. Once the car moves out of these 'edge-regions', normal algorithm is used.

1.4 Code Snippet

```python
if angle >=180 :
    angle = angle-360    # converting angle to [-180, 179]

# to check if car has ever exited 'edge-regions'
# to avoid continous steering at inner boundary
if self.init_region == 'A' or self.init_region == 'B' :
    if self.init_region != curr_region:
        self.exited_corner_region_once = True

if self.init_region == 'A' and curr_region == 'A' and \
        self.exited_corner_region_once == False:
    theta_slope=90

elif self.init_region == 'B' and curr_region == 'B' and \
        self.exited_corner_region_once == False:
    theta_slope=-90

else: # standard case, control angle found
    theta_slope = np.degrees(np.arctan\
            ((y_target-y)/(x_target-x + 1e-12)))

# controlling mechanism, threshold of error tuned manually
# either steer, or full acceleration
if abs(angle-theta_slope)>5:
    if angle-theta_slope>0:
        action_steer = 0
        action_acc = 2
    else :
        action_steer = 2
        action_acc = 2
else:
    action_steer = 1
    action_acc = 4

return np.array([action_steer, action_acc])
```

2 The Parking Lot Problem Intensifies

2.1 Problem Statement

Building on the first task, our goal now is still to navigate your 50 x 25 car out of a 700 x 700 square grid. It must exit onto the road whose entrance has its centre located at (350, 0). However, this time, the road is narrower and the parking lot is filled with 4 randomly located pits of mud (with a size of 100 x 100 each). The parking lot is once again centred at the origin with the coordinate axes shown below (screen size: 1000 x 1000).

2.2 Failed Attempts

Again, as in the last part, the policy search seemed not so promising due to already mentioned problems.

2.3 Current Approach

The algorithm that I used was a smarter roundabout to the difficult-to-implement policy search algorithms. We had access to the centres of the mudpits in the *run_simulator.py* file, from the controller class. I have used these mudpit locations to define a boolean function which returns if there is a mudpit just below the car, w.r.t x-axis.

This helps me to direct a car to a region just beyond the mudpit, after which the car tries to reach the x-axis. As the puddles are localised in quadrants only, and not completely randomly, hence following the x-axis to the target at (350,0) does indeed work.

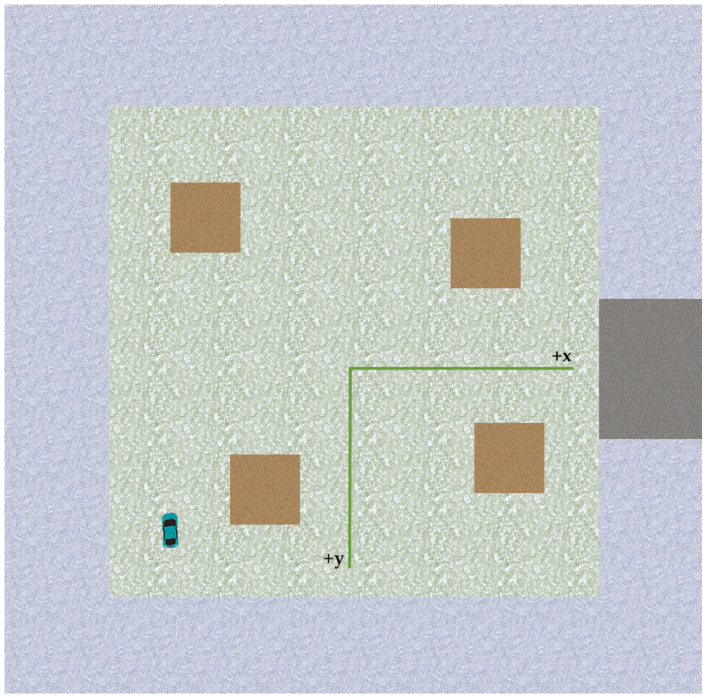

Figure 2: Ice Parking Lot with Mud Puddles

2.4 Code Snippet

```
if self.is_above_pit(state)==True:
    # go towards y axis from above the puddle
    if self.get_quadrant(state)=='1':
        theta_slope = 175
    elif self.get_quadrant(state)=='2':
        theta_slope = 0
    elif self.get_quadrant(state)=='3':
        theta_slope = 0
    elif self.get_quadrant(state)=='4':
        theta_slope = 175

else: # go towards the x axis or (350,0)
    if y>15:
        theta_slope = -90
    elif y<-15:
        theta_slope = 90
    else:
        x_target = 350
        y_target = 0
        theta_slope = np.degrees(np.arctan\
            ((y_target-y)/(x_target-x + 1e-12)))

# control actions, either steer, or full acc
if abs(angle-theta_slope)>2:
    if angle-theta_slope>0:
        action_steer = 0
        action_acc = 2
    else :
        action_steer = 2
        action_acc = 2
else:
    action_steer = 1
    action_acc = 4

action = np.array([action_steer, action_acc])

return action
```

3 References and Acknowledgements

This assignment and code is completely mine, and I have taken no direct code from anywhere.