# PROGRAMMING ASSIGNMENT 3

*An assignment report,*
*submitted to* **Prof. Shivaram Kalyanakrishnan**
*in the subject of Foundations of Intelligent and Learning Agents*

*by*

**Sandarbh Yadav**
**(22D0374)**


**INDIAN INSTITUTE OF TECHNOLOGY**
**BOMBAY**

**2022**

# TABLE OF CONTENTS

# Task 1

A controller has been constructed to navigate a car out of the parking lot on a winter night in icy surroundings. The car gets out of this situation and reaches the road safely.

## 1.1 Approach

The car gets randomly initialised in the parking lot. Initially, the controller will align the car in direction of centre of the road (350,0) by rotating clockwise or anticlockwise, whichever aligns it faster. This is achieved by steering the car over a series of time steps. A relaxation of 2° in alignment is provided on both sides. Once the car is aligned with centre of the road, it starts moving in a straight line towards centre of the road. This is achieved by accelerating the car without steering in the upcoming time steps, until it reaches the road. In this way, the car gets out of the parking lot safely without bumping into the walls. Fig. 1.1 and 1.2 depict a step by step illustration.

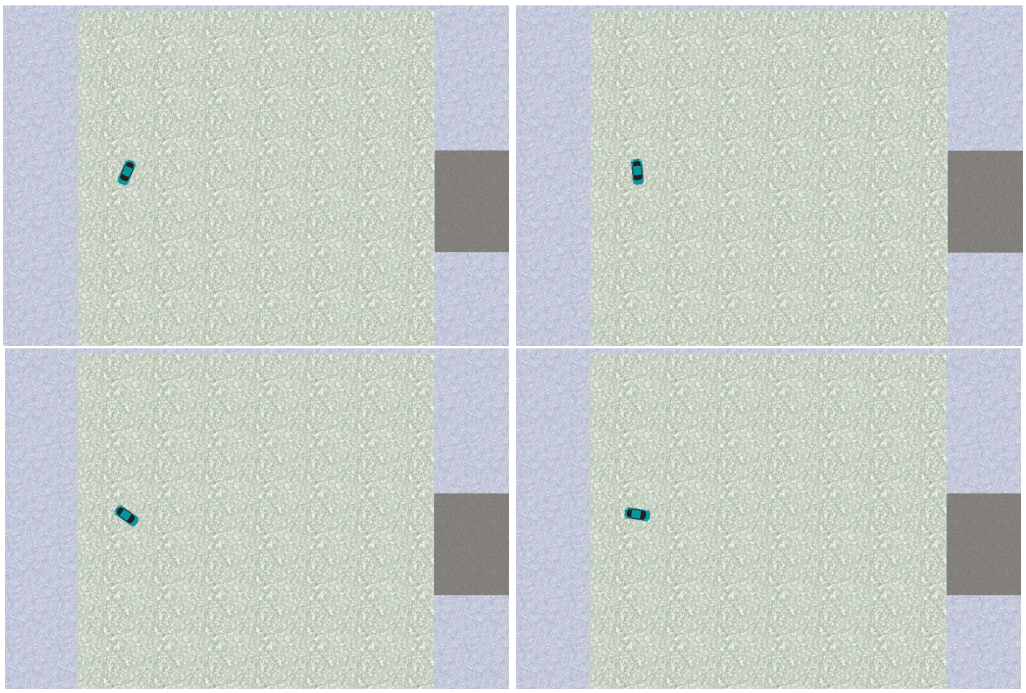

Figure 1.1: Initially, the car aligns itself with centre of the road. Here, it rotates in anticlockwise direction because it aligns the car faster compared to clockwise direction.

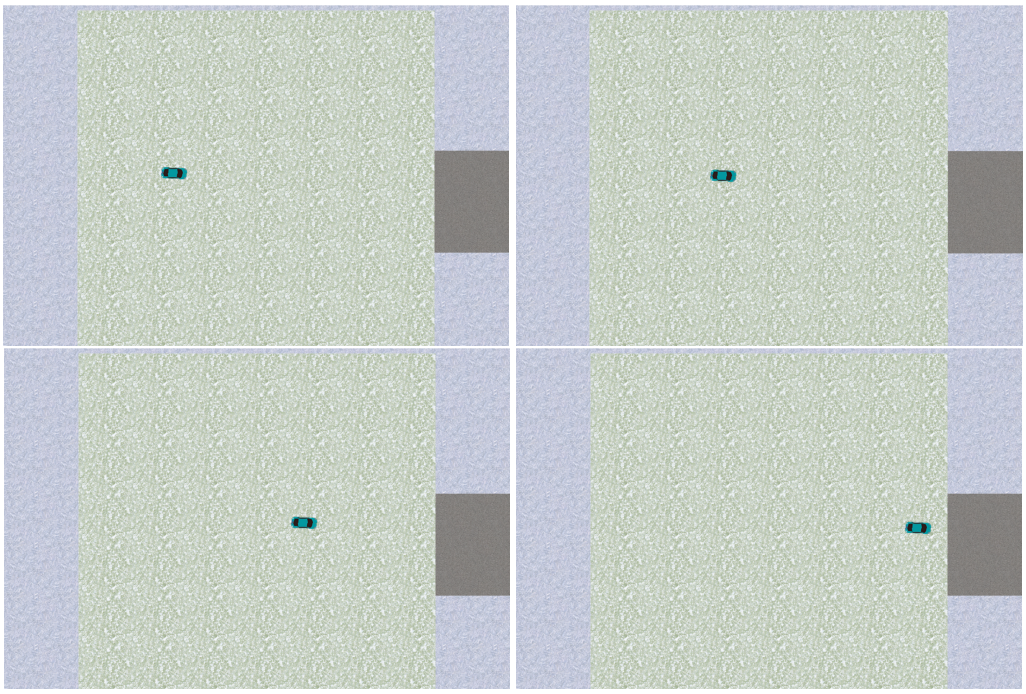

Figure 1.2: Once the car is aligned with centre of the road it starts accelerating in a straight line until it reaches the road.

## 1.2 Algorithm

**Algorithm 1:** The Parking Lot Problem

**Input:** Parameters of *next_action* function

s : array containing current coordinates, velocity and angle of car

$\theta$ : angle between centre of the road and initial coordinates of the car

**Output:** Output of *next_action* function

a : array containing actions in terms of acceleration and steer

$\phi$ = s[3] `// Current angle of the car`

`// If car is aligned with centre of the road within 2°`

**if** $abs(\theta - \phi) < 2$ **then**

| Move in a straight path towards road with acceleration 5 and steer 0

**end**

`// Else car is not aligned with centre of the road`

**else**

| Align the car with centre of the road with acceleration 0 and steer -3 or 3
| depending upon whether rotating anticlockwise or rotating clockwise
| aligns it faster

**end**

Algorithm 1 describes the *next_action* function. It takes the state array and $\theta$ as input and returns the action array. $\theta$ is the angle between centre of the road and initial coordinates of the car. It is determined in the *controller_task1* function at the beginning of the episode and is passed into the *next_action* function as a parameter. A relaxation of 2° is provided on both sides to accommodate the Gaussian noise added to the steer action. The *next_action* function determines the action to be taken in order to reach the next state while the *controller_task1* function integrates the sequence of actions to complete the entire navigation task.

## 1.3 Code Snippets

### 1.3.1 Determining $\theta$

The following snippet shows how the angle $\theta$ is calculated using arc tangent function on centre of the road and initial coordinates of the car.

```
x_init = state[0]
y_init = state[1]
theta_rad = math.atan2((0-y_init), (350-x_init))
theta_deg = np.rad2deg(theta_rad)
theta = theta_deg % 360
```

### 1.3.2 Actions taken when car is aligned with centre of the road

The following snippet shows the actions taken when the car is aligned with centre of the road. Steer is set to 0 (denoted by 1) and acceleration is set to the maximum possible value 5 (denoted by 4). This is done so that the car does not deviate from its path once aligned with centre of the road and reaches the road in a short duration of time.

```
# Car is aligned with centre of the road
if abs(theta-phi) < 2:
    # Straight
    action_steer = 1
    action_acc = 4
```

### 1.3.3 Actions taken when car is not aligned with centre of the road

The following snippet shows the actions taken when the car is not aligned with centre of the road. The car rotates clockwise with steer of 3 (denoted by 2) or anticlockwise with steer of -3 (denoted by 0), whichever aligns it faster with centre of the road. This is done so as to reduce the time taken in aligning. Acceleration is set to 0 (denoted by 2) as the car should not move forward while rotating.

```
# Car is not aligned with centre of the road
else:
    # Rotate
    if theta < 180:
        if theta < phi <= 180 + theta:
            # Anticlockwise
            action_steer = 0
            action_acc = 2
        else:
            # Clockwise
            action_steer = 2
            action_acc = 2
    else:
        if theta - 180 <= phi < theta:
            # Clockwise
            action_steer = 2
            action_acc = 2
        else:
            # Anticlockwise
            action_steer = 0
            action_acc = 2
```

# Task 2

A controller has been constructed to navigate a car out of a parking lot filled with pools of mud from a cold winter storm. The car dodges these obstacles and reaches the road safely.

## 2.1 Approach

The parking lot is divided into 3 different regions: R1, R2 and R3 as shown below. The dimensions and locations of these regions are not fixed and are influenced by coordinates of centres of the pits alongside other factors. The controller adopts a different approach depending on the region car is present in.

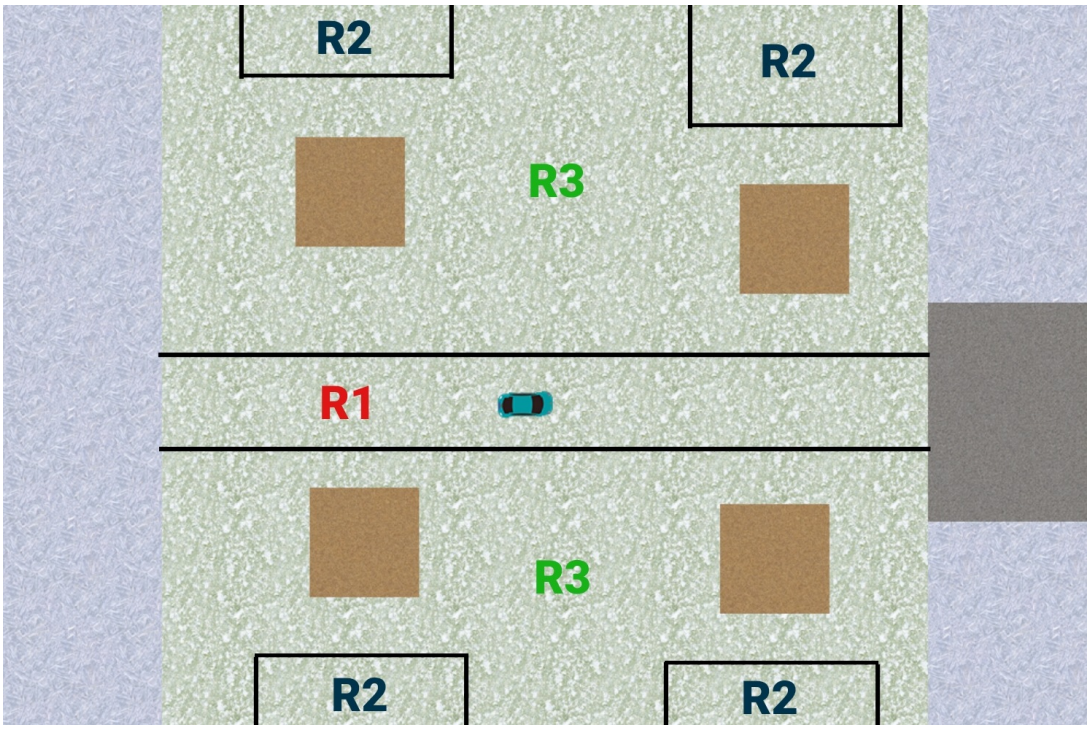


Figure 2.1: Division of parking lot into 3 different regions

R1 is the horizontal strip in front of the road. Its width is influenced by coordinates of centres of the pits and width of the road. In order to avoid collisions at the corner

of the road, R1 is limited to a maximum width of 50 units on both sides of the x-axis. However, it may become even thinner as it maintains a distance of 80 units from centres of the pits. This is done to avoid collision with pits.

R2 is the union of 4 different areas of the parking lot as shown in Fig. 2.1. Its dimensions and locations are influenced by coordinates of centres of the pits and walls of the parking lot. Vertically, the region starts from a distance of 80 units from centre of each pit and goes till the walls of parking lot. Horizontally, the region extends to a distance of 80 units on both sides of centre of each pit.

R3 comprises of remaining area of the parking lot, excluding R1, R2 and the pits.

The action taken by the controller depends on the region car is currently in. If the car is in R1, it will simply align itself horizontally and accelerate towards the road. If the car is in R3, it will align itself vertically and move towards R1. Lastly, if the car is in R2, it will align itself horizontally and move towards R3. Basically, the car moves from R2 to R3 and from R3 to R1 and once it is in R1, it races away towards the road. In this way, the car gets out of the parking lot and reaches the road safely without bumping into walls or pits. Fig. 2.2 to 2.4 show a step by step illustration of approach taken by the car when it is initialised in R2.

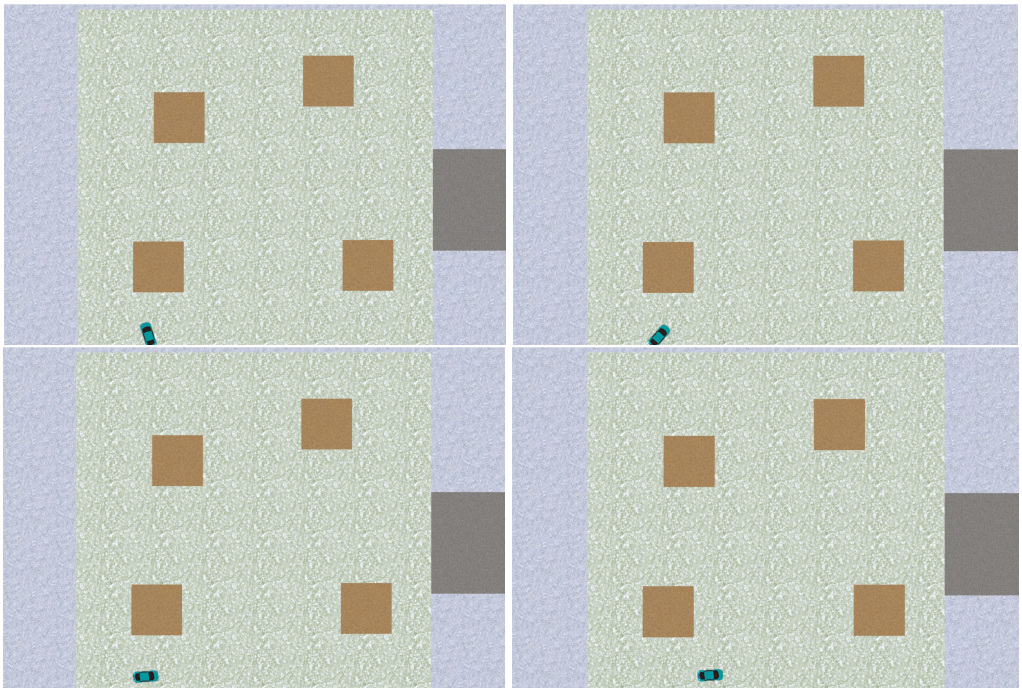

Figure 2.2: Initially the car is in R2. It aligns itself horizontally by rotating clockwise. Once aligned horizontally, the car moves in a straight path towards R3.

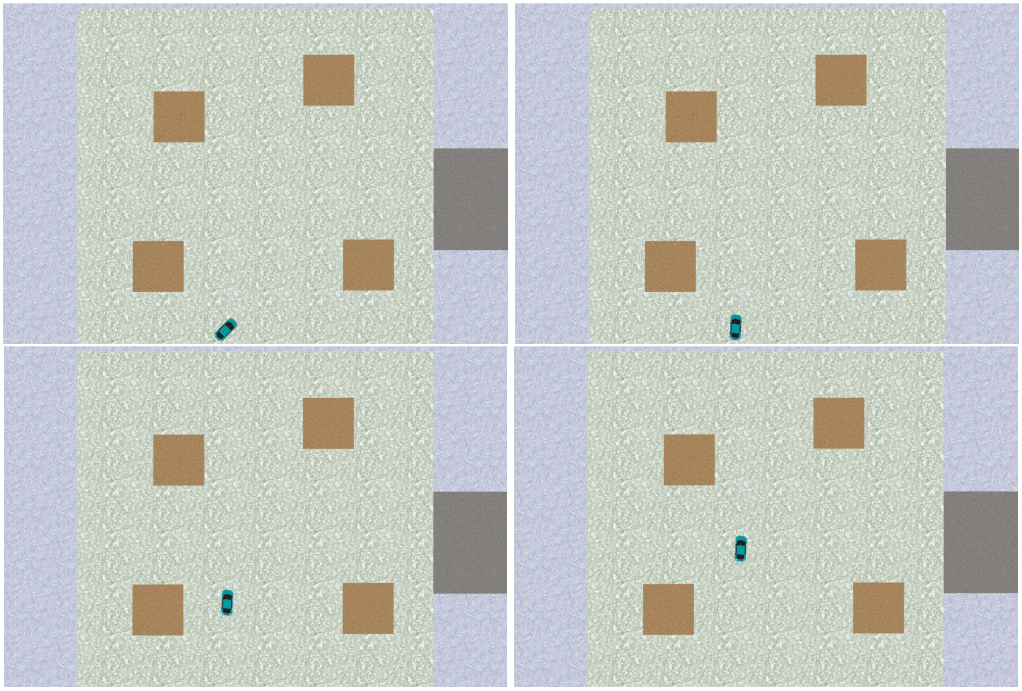

Figure 2.3: Once the car reaches R3, it aligns itself vertically and starts moving towards R1 in a straight path.

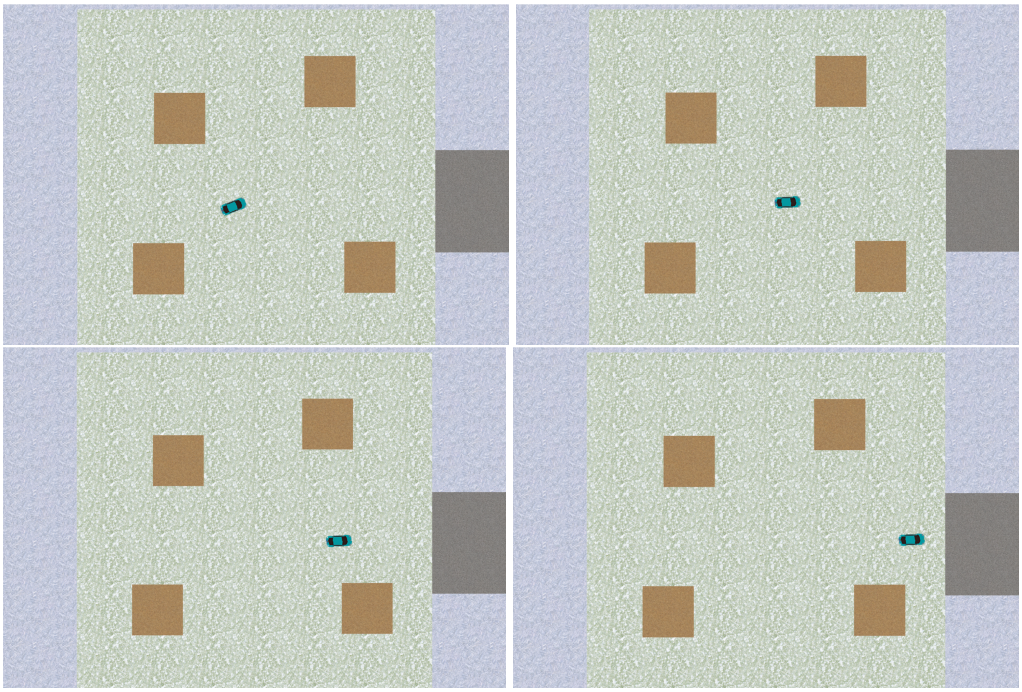

Figure 2.4: Finally, when the car reaches R1, it aligns itself horizontally and moves forward in a straight path until it reaches the road.

## 2.2 Algorithm

**Algorithm 2:** The Parking Lot Problem Intensifies

**Input:** Parameters of *next_action* function

s : array containing current coordinates, velocity and angle of car

ran_cen_list : list containing coordinates of centres of the pits

**Output:** Output of *next_action* function

a : array containing actions in terms of acceleration and steer

```
// s & ran_cen_list are used to determine the region in
   which car is present and accordingly actions are taken
// Region 1
```

**if** *car is in R1* **then**

Align the car horizontally with acceleration 0 and steer -3 or 3 depending upon whether rotating anticlockwise or rotating clockwise aligns it faster. Once the car is aligned horizontally, move in a straight path towards road with acceleration 5 and steer 0.

**end**

```
// Region 2
```

**else if** *car is in R2* **then**

Align the car horizontally with acceleration 0 and steer -3 or 3 depending upon whether rotating anticlockwise or rotating clockwise aligns it faster. Once the car is aligned horizontally, move in a straight path towards R3 with acceleration 5 and steer 0.

**end**

```
// Region 3
```

**else**

Align the car vertically with acceleration 0 and steer -3 or 3 depending upon whether rotating anticlockwise or rotating clockwise aligns it faster. Once the car is aligned vertically, move in a straight path towards R1 with acceleration 5 and steer 0.

**end**

Algorithm 2 describes the *next_action* function. It takes the state array and ran_cen_list as input and returns the action array. ran_cen_list is a list containing the coordinates of centres of the pits. It is passed into the *next_action* function as a parameter from the *controller_task2* function. The *next_action* function determines the action to be taken in order to reach the next state while the *controller_task2* function integrates the sequence of actions to complete the entire navigation task.

## 2.3 Code Snippets

### 2.3.1 Extracting state features

The following snippet shows extraction of state features. The current coordinates (x,y), velocity (v) and angle (a) is extracted from the state array. A relaxation of 4° is provided on both sides to accommodate the Gaussian noise added to the steer action. This is captured by the ang_tol variable.

```
# Extracting state features
x = state[0]
y = state[1]
v = state[2]
a = state[3]
a = a % 360
ang_tol = 4
```

### 2.3.2 Determining R1 and actions taken in it

The following snippet shows how R1 is determined and actions taken by the controller when car is in R1.

```
# Region 1
if max(ran_cen_list[1][1] + 80, ran_cen_list[3][1] + 80, -50) < y \
        < min(ran_cen_list[0][1] - 80, ran_cen_list[2][1] - 80, 50):
    if (a < ang_tol) or (360-ang_tol < a < 360):
        # Straight
        action_steer = 1
        action_acc = 4
    else:
        # Rotate
        if a <= 180:
            # Rotate anticlockwise
            action_steer = 0
            action_acc = 2
        else:
            # Rotate clockwise
            action_steer = 2
            action_acc = 2
```

R1 is enclosed by 2 lines parallel to x-axis whose distance from x-axis is influenced by centres of the pits. Also, R1 is limited to a maximum width of 50 units on both sides of the x-axis. This is done to avoid collision with pits and walls near the corners of the road. Once the car is in R1, it rotates clockwise with steer of 3 (denoted by 2) or anticlockwise with steer of -3 (denoted by 0), whichever aligns it faster horizontally. Acceleration is set to 0 (denoted by 2) during rotation. Once aligned horizontally, the car races towards the road with acceleration 5 (denoted by 4) and steer 0 (denoted by 1). This choice of actions reduces the time taken to reach the road while also avoiding any collision.

### 2.3.3 Determining R2 and actions taken in it

The following snippet shows how R2 is determined and actions taken by the controller when car is in R2.

```
# Region 2
elif ((ran_cen_list[0][0] - 80 < x < ran_cen_list[0][0] + 80) and (y > ran_cen_list[0][1] + 80))\
        or ((ran_cen_list[1][0] - 80 < x < ran_cen_list[1][0] + 80) and (y < ran_cen_list[1][1] - 80))\
        or ((ran_cen_list[2][0] - 80 < x < ran_cen_list[2][0] + 80) and (y > ran_cen_list[2][1] + 80))\
        or ((ran_cen_list[3][0] - 80 < x < ran_cen_list[3][0] + 80) and (y < ran_cen_list[3][1] - 80)):

    if (a < ang_tol) or (360 - ang_tol < a < 360):
        # Straight
        action_steer = 1
        action_acc = 4
    else:
        # Rotate
        if a <= 180:
            # Rotate anticlockwise
            action_steer = 0
            action_acc = 2
        else:
            # Rotate clockwise
            action_steer = 2
            action_acc = 2
```

R2 is the union of 4 different areas of the parking lot whose dimensions and locations are influenced by coordinates of centres of the pits and walls of the parking lot. Vertically, the region starts from a distance of 80 units from centre of each pit and goes till the walls of the parking lot. Horizontally, the region extends to a distance of 80 units on both sides of centre of each pit. Once the car is in R2, it rotates clockwise with steer of 3 (denoted by 2) or anticlockwise with steer of -3 (denoted by 0), whichever aligns it faster horizontally. Acceleration is set to 0 (denoted by 2) during rotation. Once aligned horizontally, the car races towards R3 with acceleration 5 (denoted by 4) and steer 0 (denoted by 1). This choice of actions reduces the time taken to reach R3 while also avoiding any collision.

### 2.3.4 Determining R3 and actions taken in it

The following snippet shows how R3 is determined and actions taken by the controller when car is in R3. R3 comprises of remaining area of the parking lot, excluding R1, R2 and the pits. Once the car is in R3, it rotates clockwise with steer of 3 (denoted by 2) or anticlockwise with steer of -3 (denoted by 0), whichever aligns it faster vertically. Acceleration is set to 0 (denoted by 2) during rotation. Once aligned vertically, the car races towards R1 with acceleration 5 (denoted by 4) and steer 0 (denoted by 1). This choice of actions reduces the time taken to reach R1 while also avoiding any collision.

```
# Region 3
else:
    if y > 0:
        if abs(a-270) < ang_tol:
            # Straight
            action_steer = 1
            action_acc = 4
        else:
            # Rotate
            if 90 < a < 270:
                # Clockwise
                action_steer = 2
                action_acc = 2
            else:
                # Anticlockwise
                action_steer = 0
                action_acc = 2
    else:
        if abs(a-90) < ang_tol:
            # Straight
            action_steer = 1
            action_acc = 4
        else:
            # Rotate
            if 90 < a < 270:
                # Anticlockwise
                action_steer = 0
                action_acc = 2
            else:
                # Clockwise
                action_steer = 2
                action_acc = 2
```