CS 7638: Artificial Intelligence for Robotics - Problem Set Descriptions

This document is a written description of the instructions for each problem set to be used as a reference. Please note the due dates of each as stated in the syllabus and fill out the respective ps#_answers.py file to submit to Gradescope.

Problem Set #1:

Question 1: Probability

1. Given $P(X) = 0.2$.
   What is $P(\neg X)$? _____
2. Given $P(X) = 0.2$, $P(Y) = 0.2$, and X, Y are independent.
   What is $P(X, Y)$? _____
3. Given $P(X) = 0.2$, $P(Y|X) = 0.6$, and $P(Y|\neg X) = 0.6$.
   What is $P(Y)$? _____

Question 2: Localization

Consider a robot in two-dimensional space whose position is specified by (x, y, θ), where x and y are the robot's location and θ is the direction the robot is facing. These variables are so-called *state variables*, in that they capture the robot's current state.

In three-dimensional space, we could represent a robot's position using $(x, y, z, roll, pitch, yaw)$. Note that the number of state variables has increased from three to six.

Recall that a *histogram filter* represents the possible states of the robot by dividing each dimension of the state space into a fixed set of buckets.

When we use a histogram filter, how does the memory required scale in the number of state variables? - Linearly - Quadratically - Exponentially - None of the Above

Question 3: Bayes' Rule

As a hypothetical scenario, say that you live in a house which you are worried is likely to catch fire. Every day while you are out, you call your neighbor to ask them whether your house is on fire. Your neighbor always responds with a yes or no, but sometimes they lie, saying that your house is on fire when it is not, or vice versa.

Define random variables to represent the events:

F = your house is actually on fire
B = your neighbor says your house is on fire
L = your neighbor lies

Assume that: $ P(F) = 0.001 $ and $ P(L) = 0.1 $, and that L is independent of F, that is, your neighbor is equally likely to lie whether or not your house is on fire.

First, compute the non-normalized probability distribution that your house is (or is not) on fire given that your neighbor says it is: (These values will be *proportional* to the true probability distribution, but not the same.)

$$\bar{P}(F|B) = \_\_\_\_\_$$

$$\bar{P}(\neg F|B) = \_\_\_\_\_$$

Next, normalize this distribution such that they form a valid probability distribution. (*Hint: in a valid distribution, all the probabilities sum to . . . ?*)

$P(F|B) =$ _____

$P(\neg F|B) =$ _____

Question 4: Localization Program

The localize function takes the following arguments:

- **colors**: 2D list, each entry either 'R' (for red cell) or 'G' (for green cell)
- **measurements**: list of measurements taken by the robot, each entry either 'R' or 'G'
- **motions**: list of actions taken by the robot, each entry of the form [dy,dx], where dx refers to the change in the x-direction (positive meaning movement to the right) and dy refers to the change in the y-direction (positive meaning movement downward) NOTE: the *first* coordinate is change in y; the *second* coordinate is change in x
- **sensor_right**: float between 0 and 1, giving the probability that any given measurement is correct; the probability that the measurement is incorrect is 1-sensor_right
- **p_move**:float between 0 and 1, giving the probability that any given movement command takes place; the probability that the movement command fails (and the robot remains still) is 1-p_move; the robot will NOT overshoot its destination in this exercise

The function should **RETURN** (not just show or print) a 2D list (of the same dimensions as colors) that gives the probabilities that the robot occupies each cell in the world.

Compute the probabilities by assuming the robot initially has a uniform probability of being in any cell.

Also assume that at each step, the robot: 1) first makes a movement, 2) then takes a measurement.

Motion: [0,0]-stay, [0,1]-right, [0,-1]-left, [1,0]-down, [-1,0]-up

```
def q4_localize(colors, measurements, motions, sensor_right, p_move):
    # initializes p to a uniform distribution over a grid of the same dimensions as colors
    pinit = 1.0 / float(len(colors)) / float(len(colors[0]))
    p = [[pinit for _col in range(len(colors[0]))] for _row in range(len(colors))]

    # TODO: >>> Insert your code here <<<

    return p
```

Example:

```
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R']
motions = [[0,0]]
sensor_right = 0.8
p_move = 1.0
p = localize(colors,measurements,motions,sensor_right,p_move)
correct_answer = (
    [[0.06667, 0.06667, 0.06667],
     [0.06667, 0.26667, 0.26667],
     [0.06667, 0.06667, 0.06667]])
```

Problem Set #2:

Question 1: Measurement Update

Figure 1: PS2 Question 1

After multiplying two original Gaussians, such as shown, what is the variance of the new Gaussian relative to the previous two?

- Smaller
- Larger
- the same as the original Gaussian

Question 2: New Variance

Given a prior Gaussian with mean, μ, and variance, σ^2. Our measurement is exactly the same mean and same variance. Suppose we multiply both, to get a new mean (which is the smae as the old mean) and a new variance ν^2. Express ν^2 as a multiple of σ^2.
$\nu^2 =$ ____σ^2

Question 3: Heavytail Gaussian

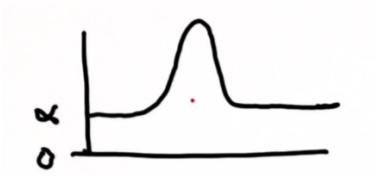


Figure 2: PS2 Question 3

Is it possible to represent this function as a Gaussian? More particularly, can you find a μ and σ^2 from which this function is obtained? _____

Question 4: How Many Dimensions

In class we addressed a 1 dimensional tracking problem where we estimated the location (x) and velocity ($\dot{x}$). What is the dimension of the state vector in the Kalman filter for an object moving in a 2 dimensional space? (How many variables are there?) _____

Question 5: State Transition Matrix

For the 1 dimension, we had that our state transition matrix, F, was equal to $\begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$ where $\Delta t = 0.1$.

What is the new F matrix for a Kalman filter in 2 dimensions? (Please use 0.1 for Δt.)

$$F = \begin{bmatrix} \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \end{bmatrix}$$

Question 6: Programming Exercise

This question requires NO CODING. Fill in the matrices P, F, H, R and I for a kalman filter with 4 state variables.

$$P = \begin{bmatrix} \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \end{bmatrix} \quad F = \begin{bmatrix} \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \end{bmatrix} \quad H = \begin{bmatrix} \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \end{bmatrix} \quad R = \begin{bmatrix} \_\_ & \_\_ \\ \_\_ & \_\_ \end{bmatrix}$$

$$I = \begin{bmatrix} \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \end{bmatrix}$$

Problem Set #3:

Question 1: Empty Cell

Consider the following world with 4 states:

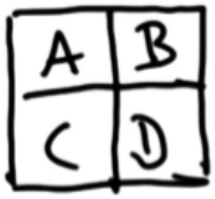


Figure 3: PS3 Question 1

For N uniform particles, what is the probability that 0 particles are in state A? Answer for $N = 1$: _____ , $N = 4$: _____, and $N = 10$: _____.

Question 2: Motion Question

Consider the same world as above with 4 states (A, B, C, D). We are given that there are 5 particles in A, 3 particles in B, 3 particles in C, and 1 particle in D. The rules of our robot is as follows:

1. 50% chance of moving horizontally.
2. 50% chance of moving vertically.
3. It never moves diagonally.
4. It never stays still in the same cell.

After one step, how many particles are in each state?

$A =$ _____, $B =$ _____, $C =$ _____, $D =$ _____

After infinite steps, how many particles are in each state?

$A =$ _____, $B =$ _____, $C =$ _____, $D =$ _____

Question 3: Single Particle

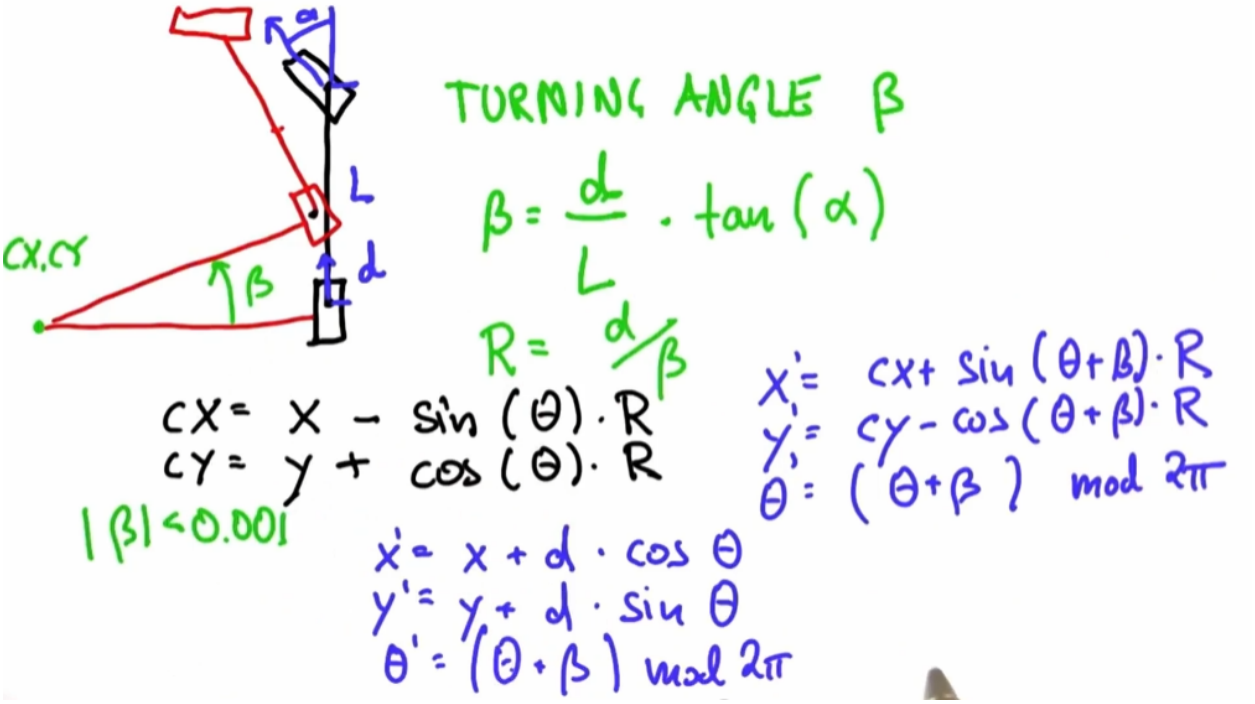
Suppose you run a particle filter with $N = 1$ particles, what do you expect will happen? (You may select multiple answers.)

- Works Fine
- Ignores Robot Measurements
- Ignores Robot Motion

- It Likely Fails
- None of Above

Question 4: Circular Motion

Given the following where the state of the robot is represented by position x, y and orientation θ, steering angle $= \alpha$, distance $= d$, and the length of the robot is represented with L. Use the following equations to create a function for executing robot movement given a list of motions.




```
def q4_move(self, motion):
    # You can replace the INSIDE of this function with the move function
    # you modified in the module quiz
    # ENTER CODE HERE
    return res # make sure your move function returns an instance
                # of the robot class with the correct coordinates.
```

Example:

```
# 1) The following code should print:
#       Robot:     [x=0.0 y=0.0 orient=0.0]
#       Robot:     [x=10.0 y=0.0 orient=0.0]
#       Robot:     [x=19.861 y=1.4333 orient=0.2886]
#       Robot:     [x=39.034 y=7.1270 orient=0.2886]

length = 20.
bearing_noise  = 0.0
steering_noise = 0.0
distance_noise = 0.0

myrobot = robot(length)
myrobot.set(0.0, 0.0, 0.0)
myrobot.set_noise(bearing_noise, steering_noise, distance_noise)

motions = [[0.0, 10.0], [pi / 6.0, 10], [0.0, 20.0]]
```

```
T = len(motions)

print ('Robot:    ', myrobot)
for t in range(T):
    myrobot = myrobot.move(motions[t])
    print ('Robot:    ', myrobot)
```

Question 5: Sensing

Create a sense function that calculates the bearing to each of the landmarks given the robot's current position and orientation as shown below.

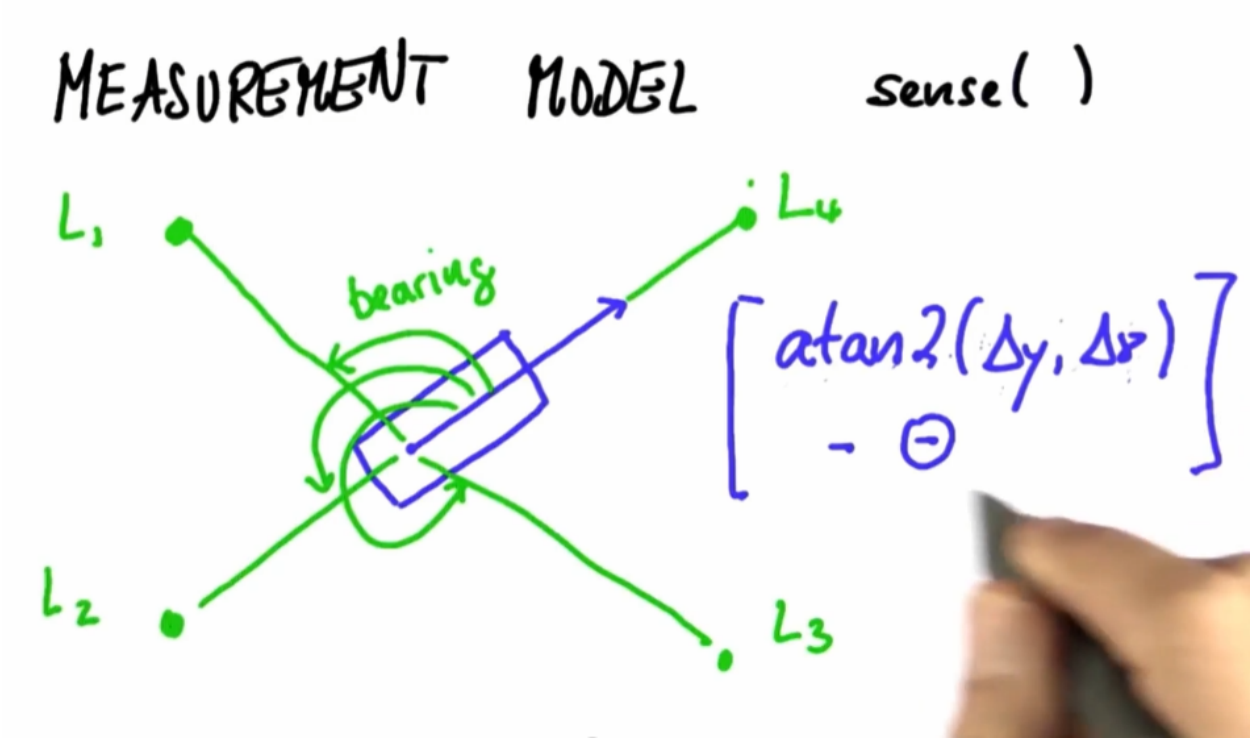


Figure 4: PS3 Question 5

```
def q5_sense(self, add_noise):
    # You can replace the INSIDE of this function with the sense function
    # you modified in the module quiz
    # You can ignore add_noise for Q5
    Z = []

    # ENTER CODE HERE
    # HINT: You will probably need to use the function atan2()

    return Z #Leave this line here. Return vector Z of 4 bearings.
```

Example:

```
# 1) The following code should print the list:
#    [6.004885648174475, 3.7295952571373605, 1.9295669970654687, 0.8519663271732721]
```

```
length = 20.
bearing_noise  = 0.0
steering_noise = 0.0
distance_noise = 0.0

myrobot = robot(length)
myrobot.set(30.0, 20.0, 0.0)
myrobot.set_noise(bearing_noise, steering_noise, distance_noise)

print ('Robot:        ', myrobot)
print( 'Measurements: ', myrobot.sense())
```

Question 6: Final Quiz

Putting it all together: modify the previous procedures to accomodate bearing noise, steering noise, and distance noise.

```
def q6_move(self, motion):
    # You can replace the INSIDE of this function with the move function
    # you modified in the module quiz
    # Note that you will need to handle motion noise inside your function accordingly

    return res    # return a new robot object that you created in the move function

def q6_sense(self, add_noise=1):
    # You can replace the INSIDE of this function with what you changed in the module quiz
    # Note the add_noise parameter is passed to sense()
    Z = []

    # ENTER CODE HERE
    # HINT: You will probably need to use the function atan2()

    return Z
```

Problem Set #4:

Question 1: Admissable Heuristic

Given the following where the goal is in the bottom right corner and an admissable heuristic is one such that $h(x) \leq cost\_to\_goal$.

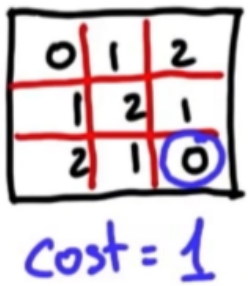


Figure 5: PS4 Question 1

Is the function shown here admissible?

- Yes or No

Question 2: Admissable Heuristic 2

Given the following:

Figure 6: PS4 Question 2

Is the function shown here admissible?

- Yes or No

Question 3: Bad Heuristic

What may happen if h is not admissible?

- A* finds an optimal path, always.
- A* MAY find a suboptimal path.
- A* MAY fail to find a path even if one exists.
- None of the above.

Question 4: Diagonal Motion

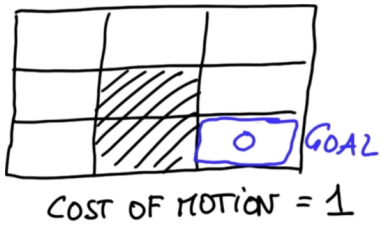


Figure 7: PS4 Question 4

We allow straight and diagonal motion. Fill in the value of each grid cell, if your robot may move straight or diagonally, *both* with a cost of 1.

$$\begin{bmatrix} \_\_ & \_\_ & \_\_ \\ \_\_ & \# & \_\_ \\ \_\_ & \# & 0 \end{bmatrix}$$

Question 5: Stochastic Motion

Model stochastic actions for the robot in the function below where success_prob is the probability of successfully executing an action. If the robot fails, it moves orthogonal to the intended action with equal probability. (For example: If the success_prob = 50% and the robot intended action is up, it may move left or right with 25% probability for either.)

```
def q5_stochastic_motion(grid, goal, cost_step, collision_cost, success_prob):
```

```
    # Be sure to fix or replace the following two initialization lines with the
    # correct initialization of value and policy
    value = [[n for col in range(len(grid[0]))] for row in range(len(grid))]
    policy = [[' ' for col in range(len(grid[0]))] for row in range(len(grid))]

    # ENTER CODE HERE

    # You will need to be sure to return the following
    return value, policy
```

Example:

```
grid = [[0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 1, 1, 0]]
goal = [0, len(grid[0])-1] # Goal is in top right corner
cost_step = 1
collision_cost = 1000
success_prob = 0.5

value,policy = stochastic_value(grid,goal,cost_step,collision_cost,success_prob)
for row in value:
    print(row)
for row in policy:
    print(row)

# Expected outputs:
#
#[471.9397246855924, 274.85364957758316, 161.5599867065471, 0],
#[334.05159958720344, 230.9574434590965, 183.69314862430264, 176.69517762501977],
#[398.3517867450282, 277.5898270101976, 246.09263437756917, 335.3944132514738],
#[700.1758933725141, 1000, 1000, 668.697206625737]
```

Problem Set #5:

Question 1: Missing Parameters

Make the appropriate selections below for the missing parameters. Each column should be selected only once.

Question 2: Cyclic Smoothing

Implement a cyclic smoothing algorithm. This algorithm should not fix the end points (as you did in the lesson modules). You should use the gradient descent equations that you used previously. Your function should return the newpath that it calculates. (Please make sure to update newpath[i][j] only once per iteration of i and j, instead of using two update statements per iteration.)

```
def smooth(path, weight_data = 0.1, weight_smooth = 0.1, tolerance = 0.00001):

    #
    # TODO: Enter code here
    #
    return newpath
```

Question 3: Constrained Smoothing

Now you will be incorporating fixed points into your smoother. You will need to use the equations from gradient descent AND the new equations presented in the previous lecture to implement smoothing with fixed points.

```
def smooth2(path, fix, weight_data = 0.0, weight_smooth = 0.1, tolerance = 0.00001):
    #
    # TODO: Enter code here.
    #
    return newpath
```

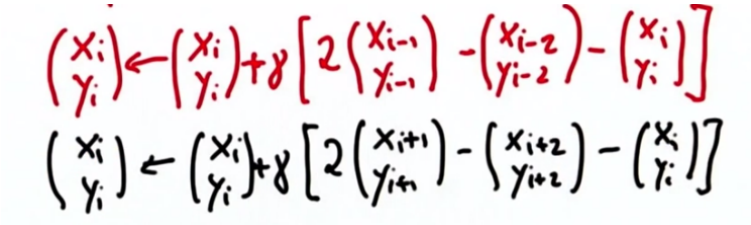


Figure 8: PS5 Question 3

Question 4: Racetrack Control

Define a function cte in the robot class that will compute the crosstrack error for a robot on a racetrack with a shape as described in the video and picture below.

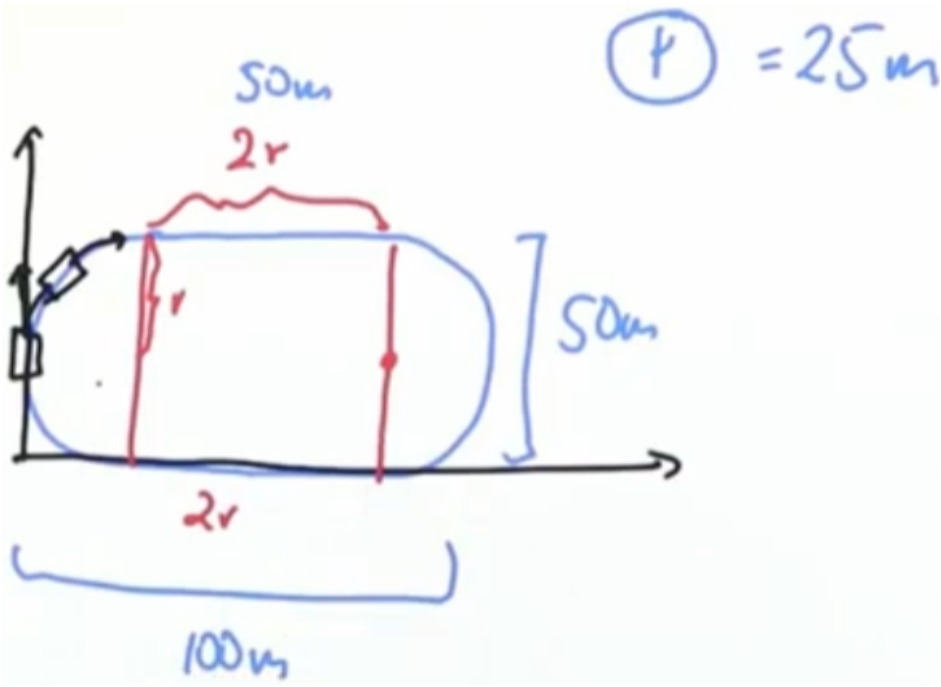


Figure 9: PS5 Question 4

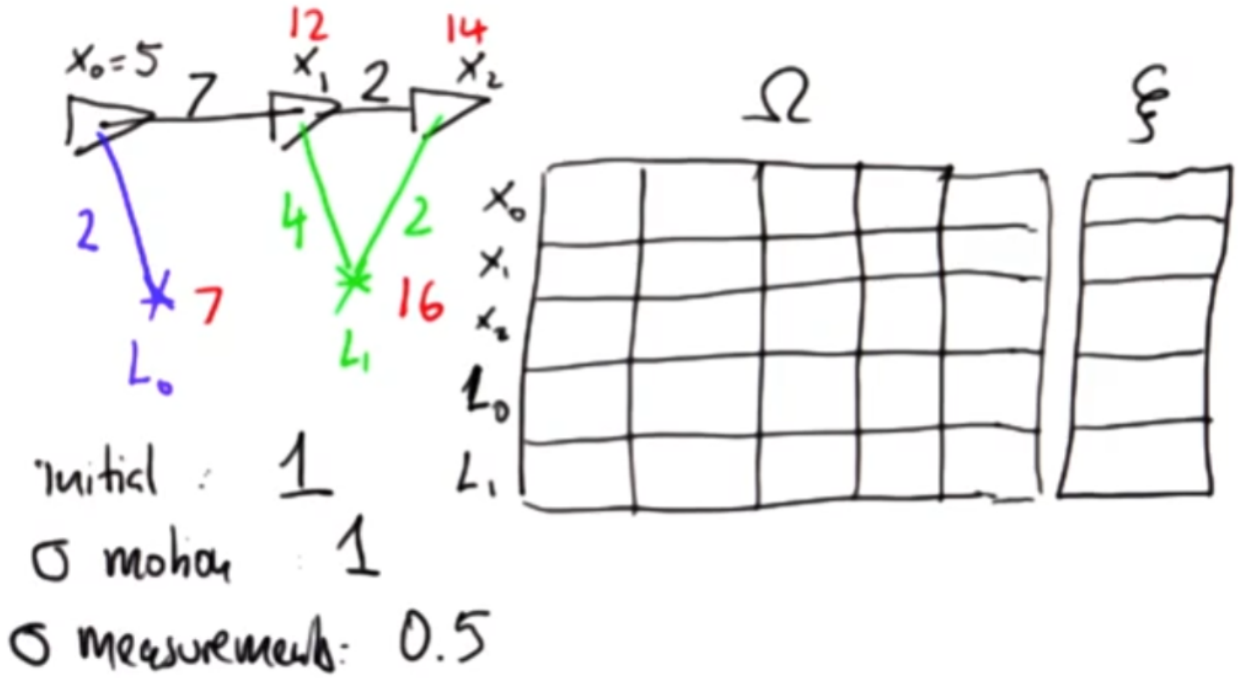
You will need to base your error calculation on the robot's location on the track. Remember that the robot will be traveling to the right on the upper straight segment and to the left on the lower straight segment.

```
def cte(self, radius):
    #
    cte = 5
    # TODO: Add code here
    #
    return cte
```

Problem Set #6:

Question 1: Matrix Fill In

Given the following, fill in the correct values for the Ω and ξ marices. Some values are filled in for you.



$$\Omega = \begin{bmatrix} 4 & \_\_ & \_\_ & \_\_ & \_\_ \\ -1 & \_\_ & \_\_ & \_\_ & \_\_ \\ 0 & \_\_ & \_\_ & \_\_ & \_\_ \\ -2 & \_\_ & \_\_ & \_\_ & \_\_ \\ 0 & -2 & -2 & \_\_ & 4 \end{bmatrix} \quad \xi = \begin{bmatrix} -6 \\ -3 \\ \_\_ \\ \_\_ \\ 12 \end{bmatrix}$$

Question 2: Online SLAM

In this problem you will implement a more manageable version of graph SLAM in 2 dimensions. Define a function, online_slam, that takes 5 inputs: data, N, num_landmarks, motion_noise, and measurement_noise–just as was done in the last programming assignment of unit 6. This function must return TWO matrices, mu and the final Omega.

```
def online_slam(data, N, num_landmarks, motion_noise, measurement_noise):
    #
    #ENTER CODE HERE
    #
    mu = matrix()
    Omega = matrix()

    return mu, Omega  # make sure you return both of these matrices to be marked correct.
```

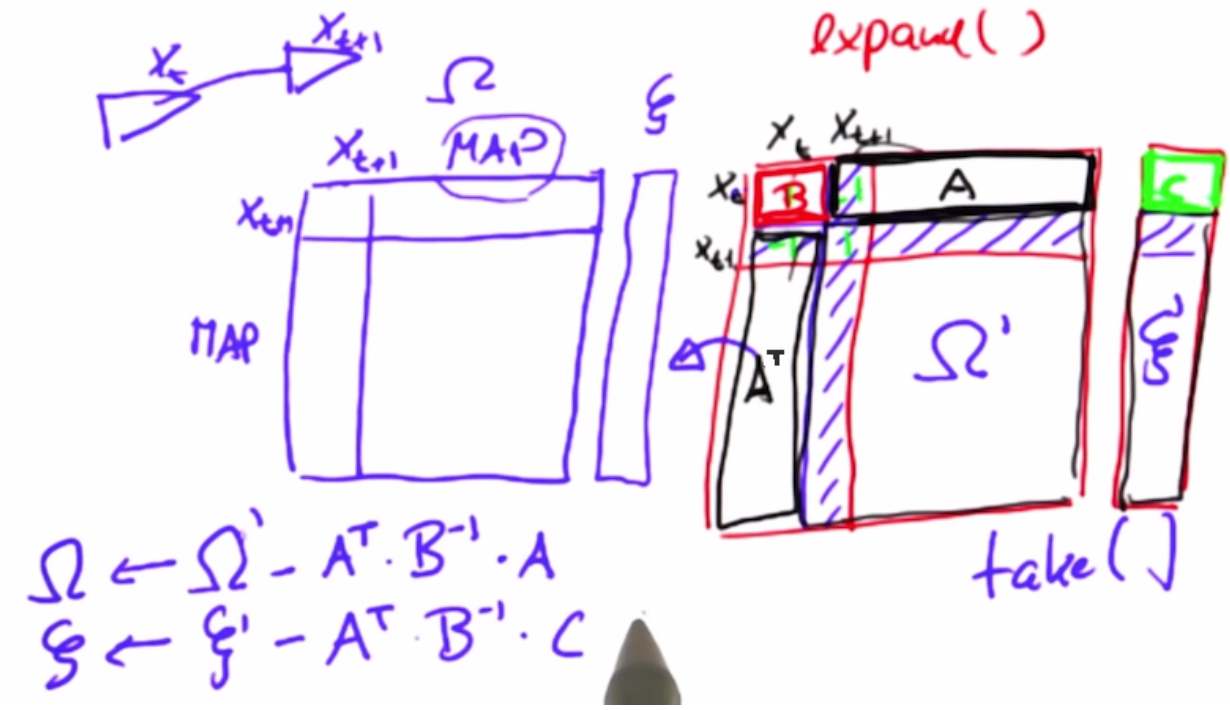


Figure 10: PS2 Question 2