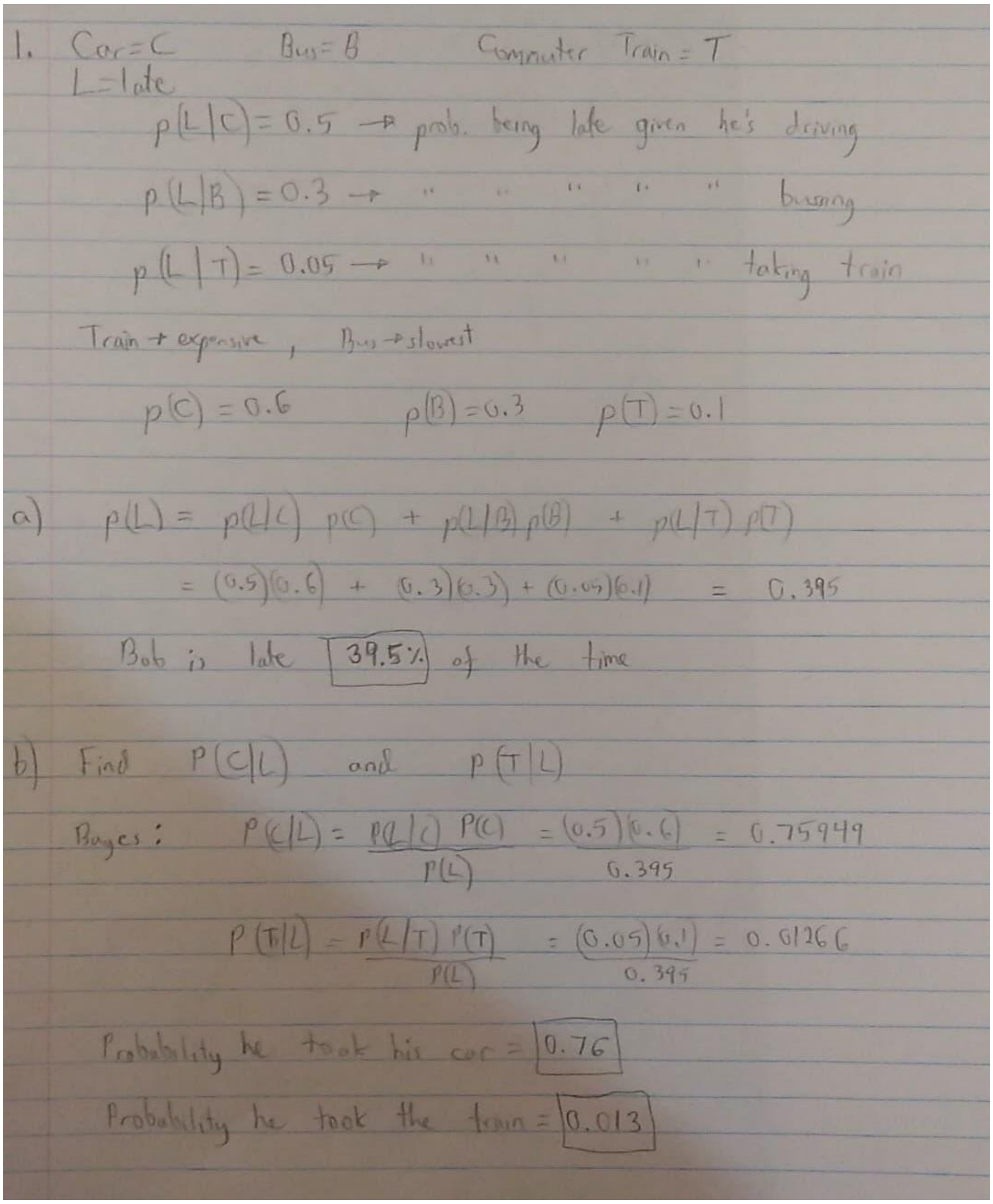
Q1.



1. Car = C Bus = B Commuter Train = T
L = late

$p(L|C) = 0.5 \rightarrow$ prob. being late given he's driving

$p(L|B) = 0.3 \rightarrow$ " " " " " busing

$p(L|T) = 0.05 \rightarrow$ " " " " " taking train

Train → expensive , Bus → slowest

$p(C) = 0.6$ $p(B) = 0.3$ $p(T) = 0.1$

a) $p(L) = p(L|C)\,p(C) + p(L|B)\,p(B) + p(L|T)\,p(T)$

$= (0.5)(0.6) + (0.3)(0.3) + (0.05)(0.1) = 0.395$

Bob is late $\boxed{39.5\%}$ of the time

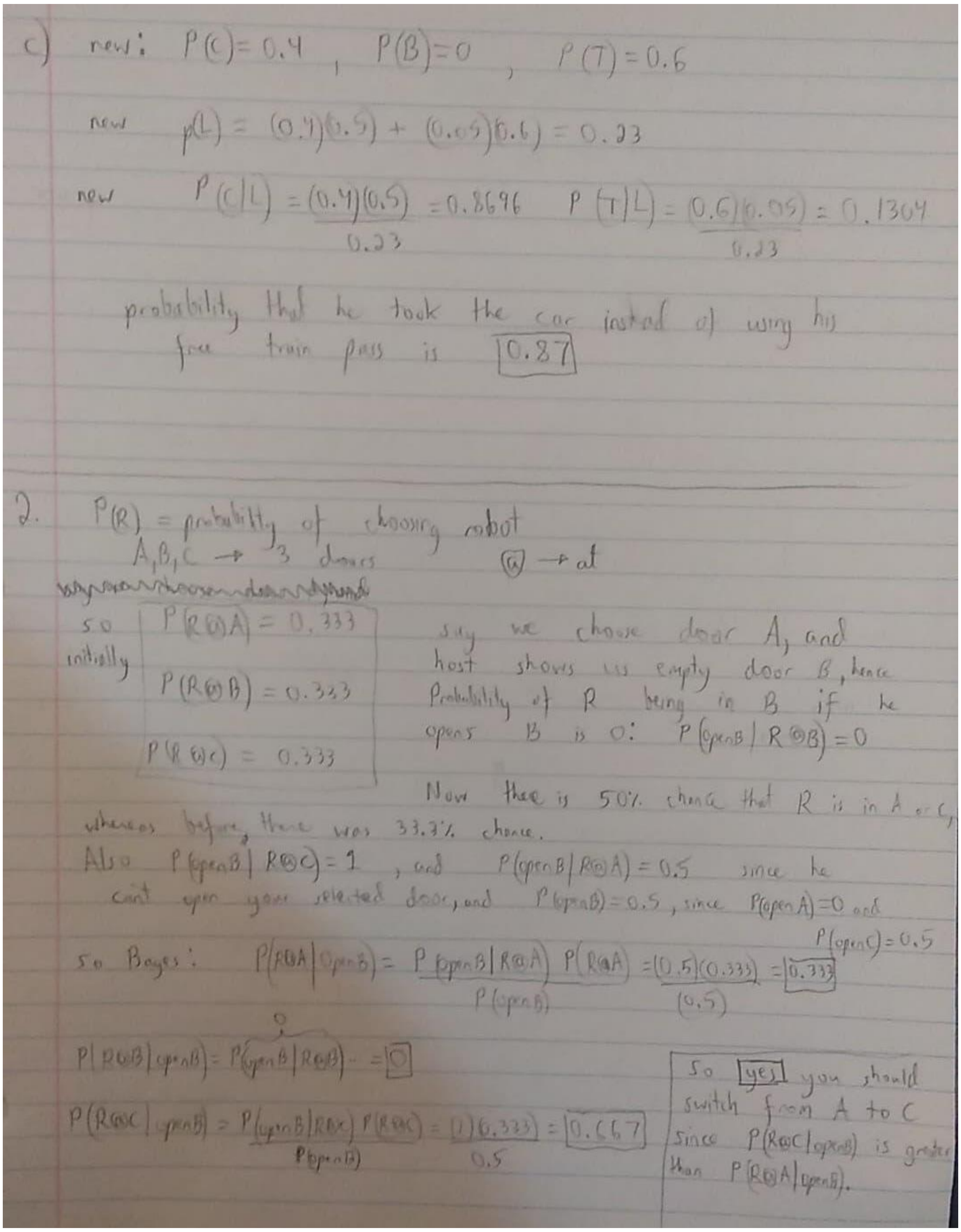
b) Find $P(C|L)$ and $p(T|L)$

Bayes: $P(C|L) = \frac{P(L|C)\,P(C)}{P(L)} = \frac{(0.5)(0.6)}{0.395} = 0.75949$

$P(T|L) = \frac{P(L|T)\,P(T)}{P(L)} = \frac{(0.05)(0.1)}{0.395} = 0.01266$

Probability he took his car = $\boxed{0.76}$

Probability he took the train = $\boxed{0.013}$



c) new: $P(C) = 0.4$, $P(B) = 0$, $P(T) = 0.6$

new $p(L) = (0.4)(0.5) + (0.05)(0.6) = 0.23$

new $P(C|L) = \frac{(0.4)(0.5)}{0.23} = 0.8696$ $P(T|L) = \frac{(0.6)(0.05)}{0.23} = 0.1304$

probability that he took the car instead of using his free train pass is $\boxed{0.87}$

2. $P(R)$ = probability of choosing robot
A, B, C → 3 doors $\quad$ @ → at

[illegible]

so initially

$P(R@A) = 0.333$

$P(R@B) = 0.333$

$P(R@C) = 0.333$

say we choose door A, and host shows us empty door B, hence Probability of R being in B if he opens B is 0: $P(\text{open}B \mid R@B) = 0$

Now there is 50% chance that R is in A or C, whereas before, there was 33.3% chance.

Also $P(\text{open}B \mid R@C) = 1$, and $P(\text{open}B \mid R@A) = 0.5$ since he can't open your selected door, and $P(\text{open}B) = 0.5$, since $P(\text{open}A) = 0$ and $P(\text{open}C) = 0.5$

So Bayes: $P(R@A \mid \text{open}B) = \frac{P(\text{open}B \mid R@A)\, P(R@A)}{P(\text{open}B)} = \frac{(0.5)(0.333)}{(0.5)} = \boxed{0.333}$

$P(R@B \mid \text{open}B) = \overbrace{P(\text{open}B \mid R@B)}^{0} \cdots = \boxed{0}$

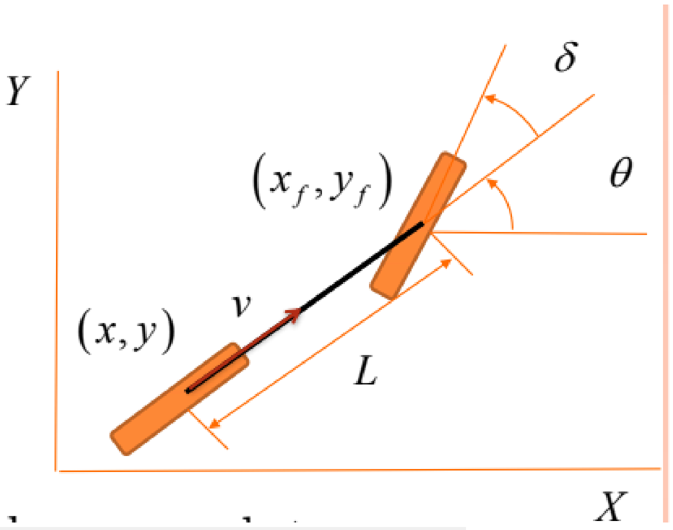
$P(R@C \mid \text{open}B) = \frac{P(\text{open}B \mid R@C)\, P(R@C)}{P(\text{open}B)} = \frac{(1)(0.333)}{0.5} = \boxed{0.667}$

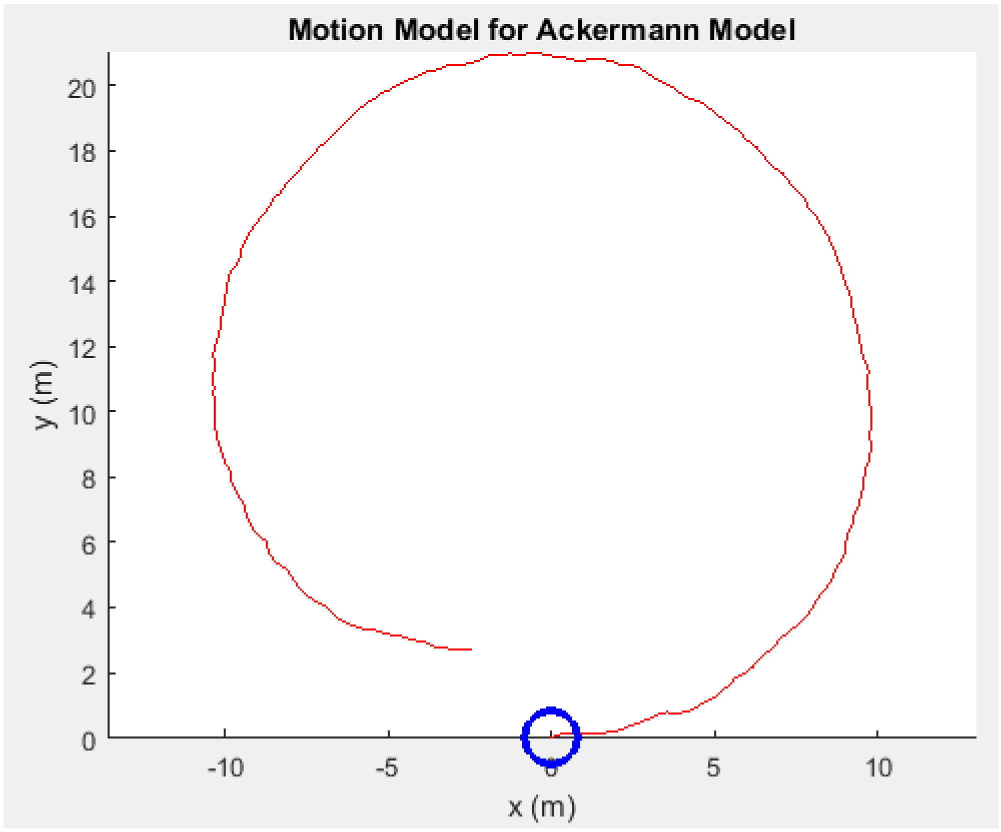
So $\boxed{\text{yes}}$ you should switch from A to C since $P(R@C \mid \text{open}B)$ is greater than $P(R@A \mid \text{open}B)$.

3. B
4. C
5. B
6. A
7. C

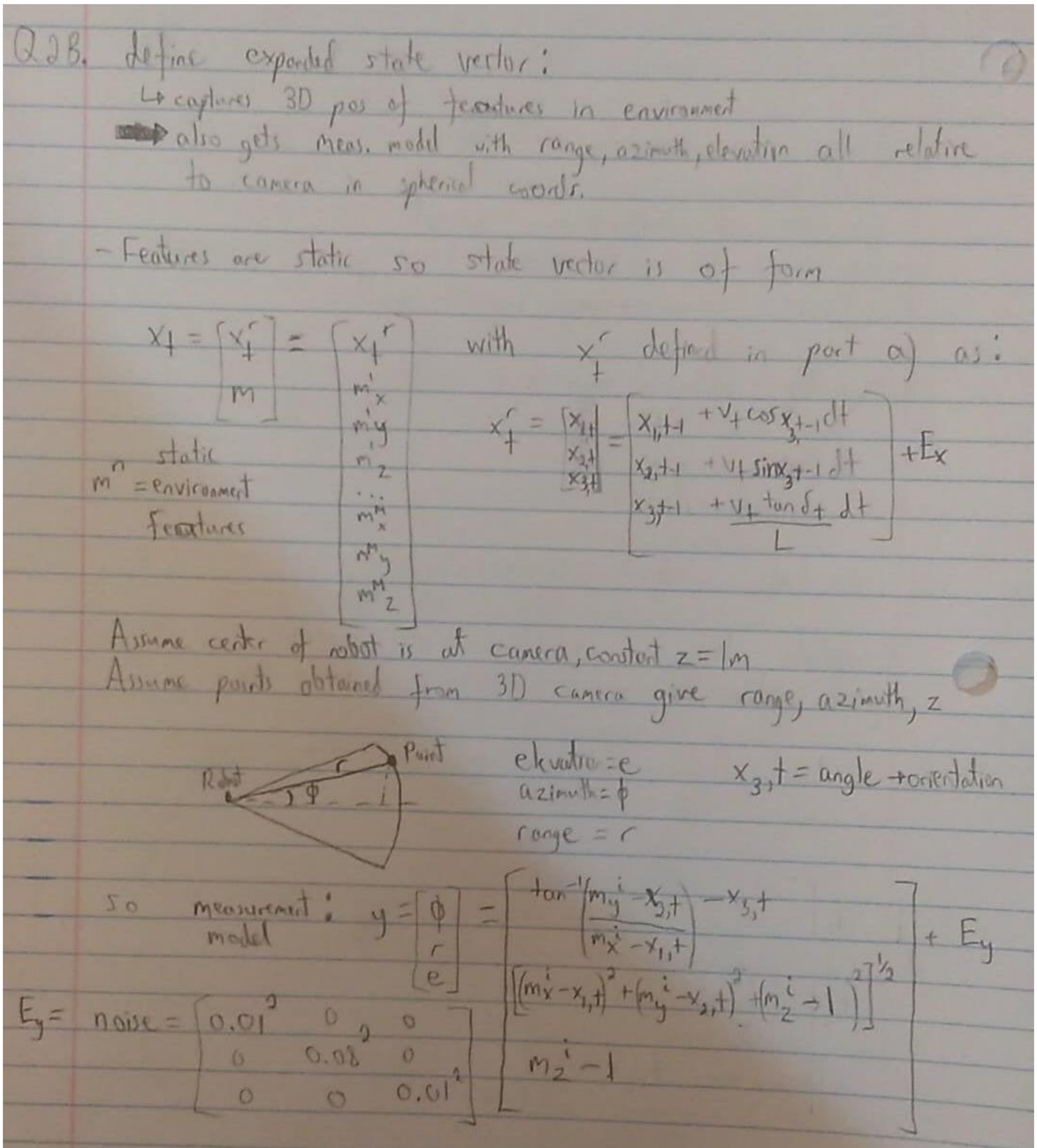
Q2A. See Ackermann bicycle robot model below (taken from lecture slides), and graph for 20 seconds from matlab (file included in in zip).

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + v_t \cos\theta_{t-1} dt \\ y_{t-1} + v_t \sin\theta_{t-1} dt \\ \theta_{t-1} + \dfrac{v_t \tan\delta_t}{L} dt \end{bmatrix}$$





Q2B.



Q2B. define expanded state vector:

↳ captures 3D pos of features in environment

➡ also gets meas. model with range, azimuth, elevation all relative to camera in spherical coords.

- Features are static so state vector is of form

$$x_t = \begin{bmatrix} x_t^r \\ M \end{bmatrix} = \begin{bmatrix} x_t^r \\ m_x^1 \\ m_y^1 \\ m_z^1 \\ \vdots \\ m_x^M \\ m_y^M \\ m_z^M \end{bmatrix}$$

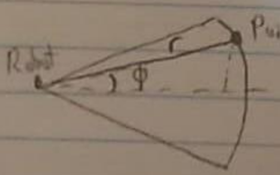
m^n = static environment features

with x_t^r defined in part a) as:

$$x_t^r = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \end{bmatrix} = \begin{bmatrix} x_{1,t-1} + v_t \cos x_{3,t-1} dt \\ x_{2,t-1} + v_t \sin x_{3,t-1} dt \\ x_{3,t-1} + \frac{v_t \tan \delta_t}{L} dt \end{bmatrix} + E_x$$

Assume center of robot is at camera, constant $z = 1m$

Assume points obtained from 3D camera give range, azimuth, z



elevation = e

azimuth = ϕ

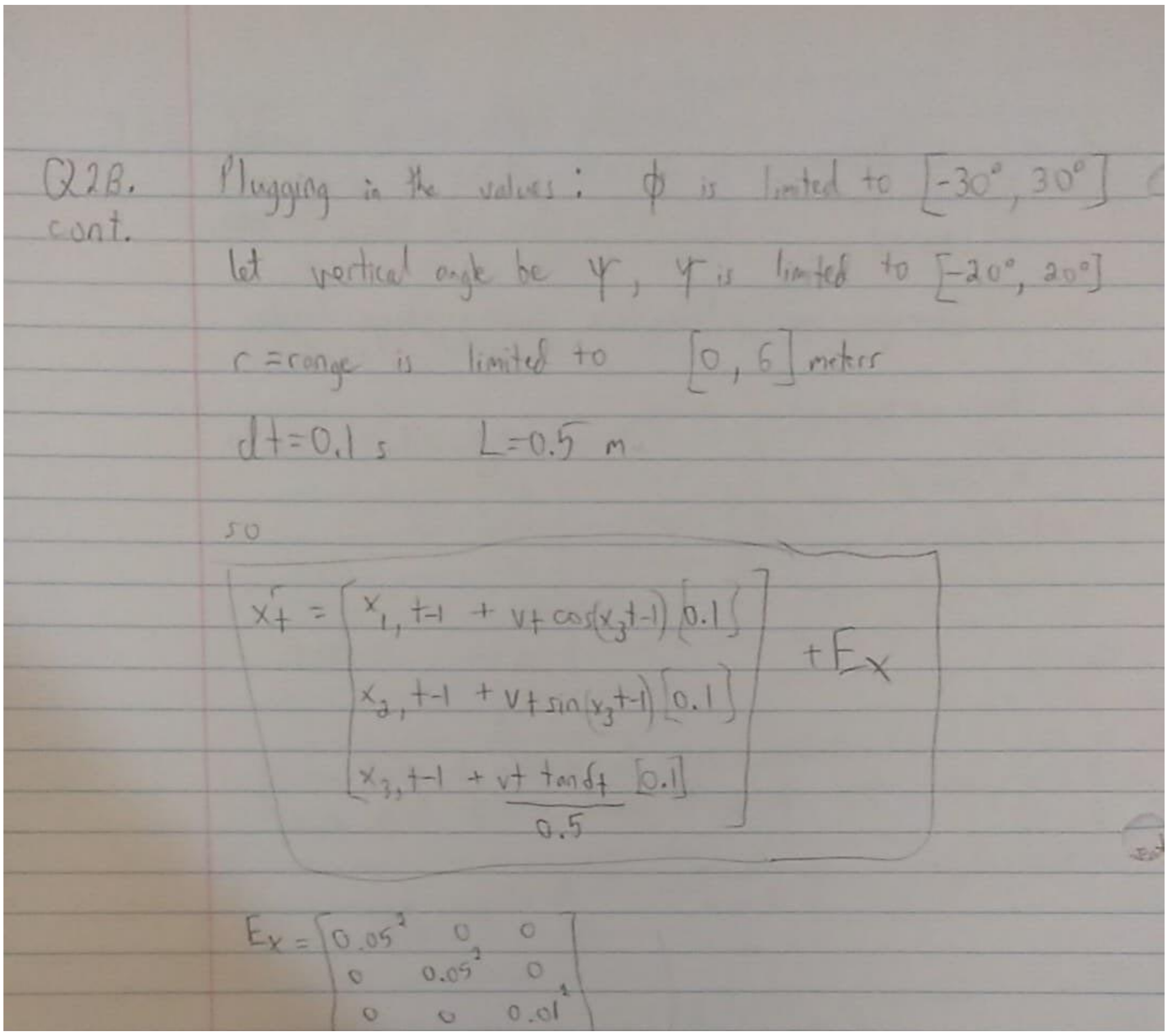
range = r

$x_{3,t}$ = angle + orientation

so measurement model:

$$y = \begin{bmatrix} \phi \\ r \\ e \end{bmatrix} = \begin{bmatrix} \tan^{-1}\left(\frac{m_y^i - x_{2,t}}{m_x^i - x_{1,t}}\right) - x_{3,t} \\ \left[(m_x^i - x_{1,t})^2 + (m_y^i - x_{2,t})^2 + (m_z^i - 1)^2\right]^{1/2} \\ m_z^i - 1 \end{bmatrix} + E_y$$

$$E_y = \text{noise} = \begin{bmatrix} 0.01^2 & 0 & 0 \\ 0 & 0.08^2 & 0 \\ 0 & 0 & 0.01^2 \end{bmatrix}$$



Q2B. cont.

Plugging in the values: ϕ is limited to $[-30°, 30°]$

let vertical angle be ψ, ψ is limited to $[-20°, 20°]$

r = range is limited to $[0, 6]$ meters

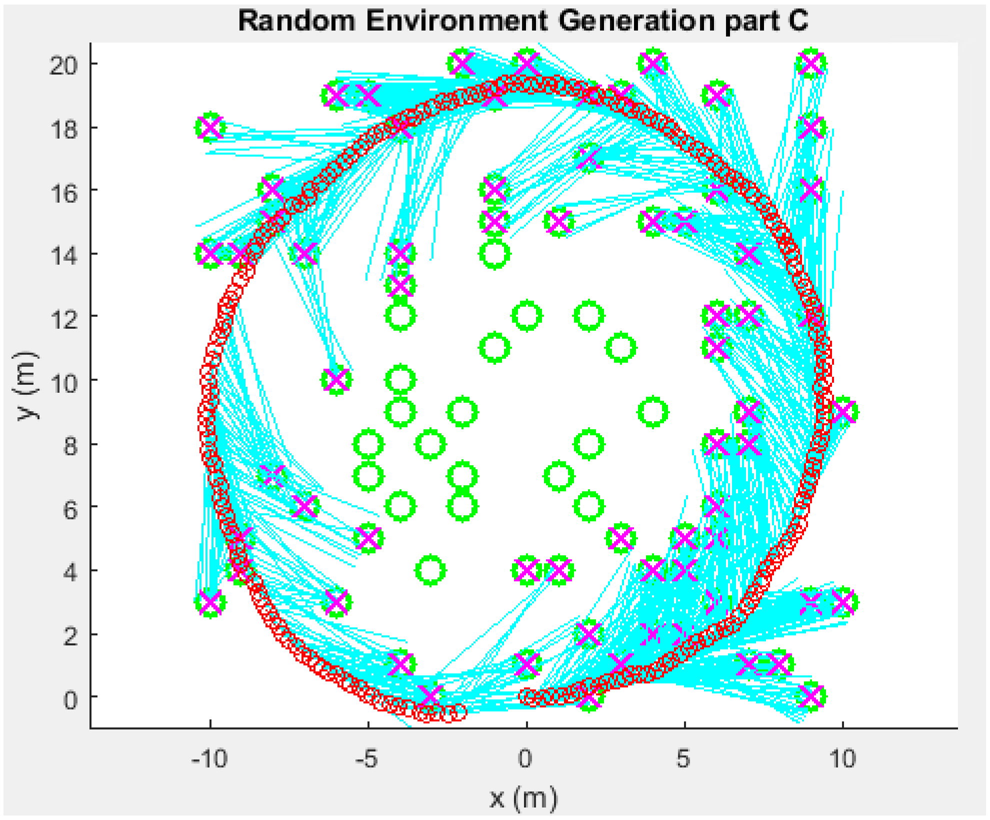
$dt = 0.1$ s $\quad L = 0.5$ m

so

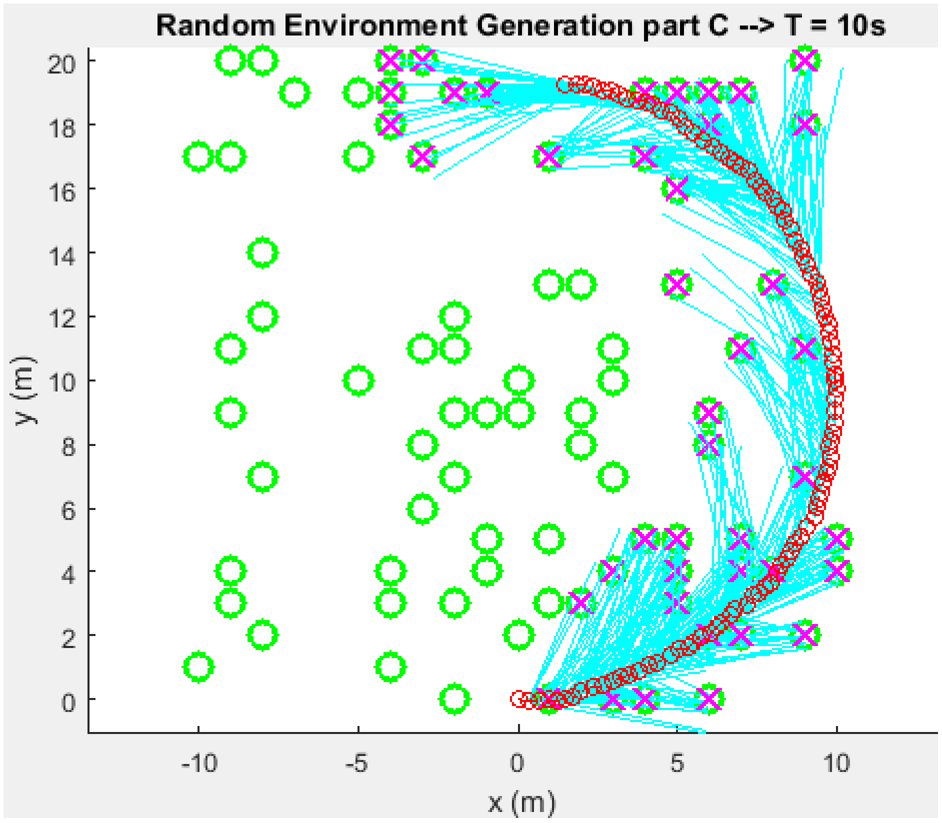
$$\boxed{\bar{x}_t = \begin{bmatrix} x_{1,t-1} + v_t \cos(x_{3,t-1})[0.1] \\ x_{2,t-1} + v_t \sin(x_{3,t-1})[0.1] \\ x_{3,t-1} + \frac{v_t \tan\delta_t}{0.5}[0.1] \end{bmatrix} + E_x}$$

$$E_x = \begin{bmatrix} 0.05^2 & 0 & 0 \\ 0 & 0.05^2 & 0 \\ 0 & 0 & 0.01^2 \end{bmatrix}$$

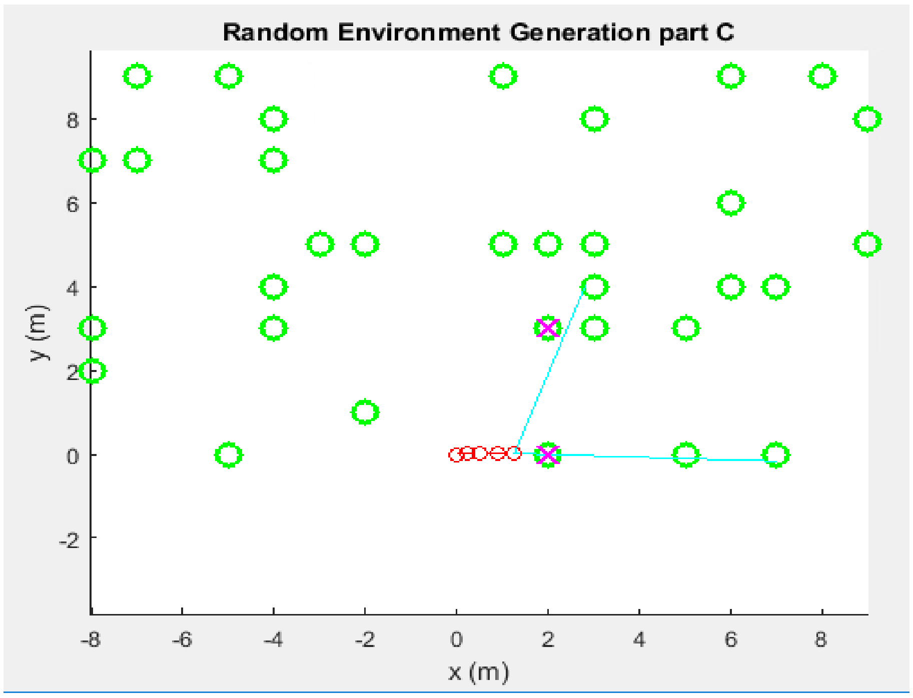
Q2C.


Random Environment Generation part C
20
18
16
14
12
10
8
6
4
2
0
y (m)
-10
-5
0
5
10
x (m)

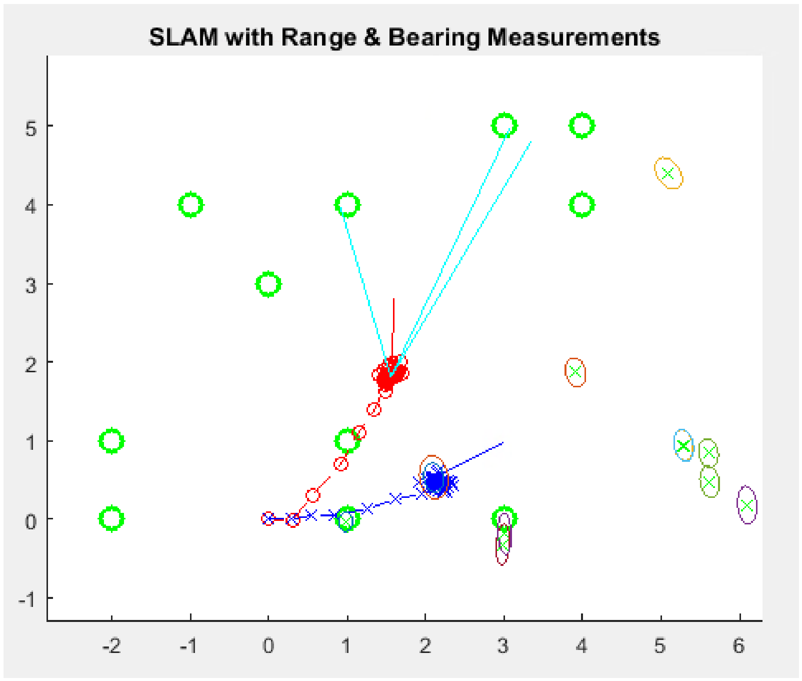
T = 20s


Random Environment Generation part C --> T = 10s
20
18
16
14
12
10
8
6
4
2
0
y (m)
-10
-5
0
5
10
x (m)

Individual blob:



Q2D. See code for EKF SLAM algorithm. Features are initialized randomly within the map space (same as in Q2C). All steps can be seen in the code regarding linearization and measurement step update. Note that the problem was coded in 2D since elevation isn't significant + I ran out of time...

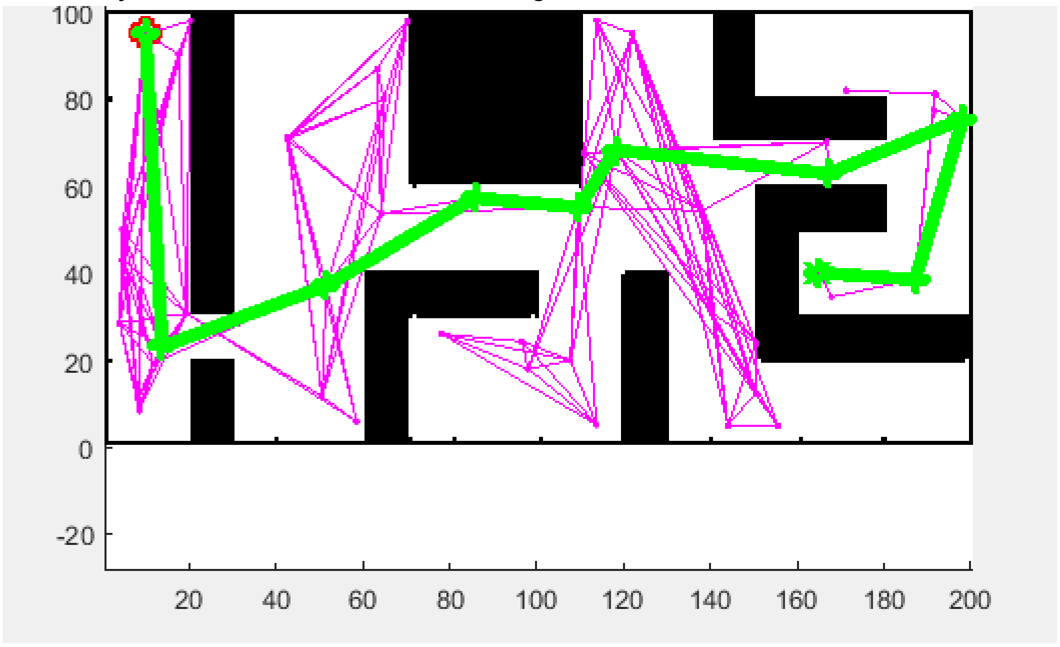


Q2E. Yes there should be a difference in estimation accuracy between 3 axes because they have diff Gaussians (noise), and the measurement model is not uniform for all 3 axes.

Q3A and Q3C. Explanation of sampling strategy, milestone connection strategy, and path identification method from my code is seen below. Also, see complete code labeled as Q3A. A map is generated using the MazeMap.m file.

First, create map of 0s (free) and 1s (occupied)that matches the maze file
PRM algo, to find path from start to end:
1. Sampling strategy sets a max limit of 400 randomly generated samples within the map boundaries. We create several samples that are located within the map boundaries.
2. Check if samples are in an occupied or free space, keep all the samples in free space, and call them milestones
3. Milestones are connected using simple Pythagorean Theorem edges (sample 1 connects to sample 2,3,4,5,etc... And repeat for all samples)
4. For each edge connection, check using Bresenham if any discretized point along the edge is in an occupied space. If so, disregard edge.
5. Once a closed path from start to end is found, find shortest path using A* search algorithm.
If closed path is not found, add more samples, and repeat algorithm. If number of samples exceeds 400, remove old samples to avoid clustering and lag
6. Finally plot the shortest path

It is possible to traverse an arbitrary path with the control options described above by assuming that the robot can physically travel from start to finish without altering the map structure. The robot will simply have to stop at each milestone along the chosen path, turn to the desired orientation, and then proceed. This slows down the path follower algorithm, but drastically decreases the chance of hitting obstacles.



Shortest path is seen in green

Q3B and Q3C.
5 motion primitives:

1. Drive forward in straight line for random amount of time
2. Turn 90 deg to the right (stationary)
3. Turn 90 deg to the left (stationary)
4. Turn 90 deg to the right using an arc of radius = 1m
5. Turn 90 deg to the left using an arc of radius = 1m

Define RRT algo that selects segments from these 5 motion primitives and makes a single tree from start to end node.

A possible method for completing the algorithm is done by creating two trees that grow at the same rate. One tree root is at the start position (Tree A), and the second tree root is at the goal position (Tree B). An overview of the general algorithm is seen below:

While path is not closed:

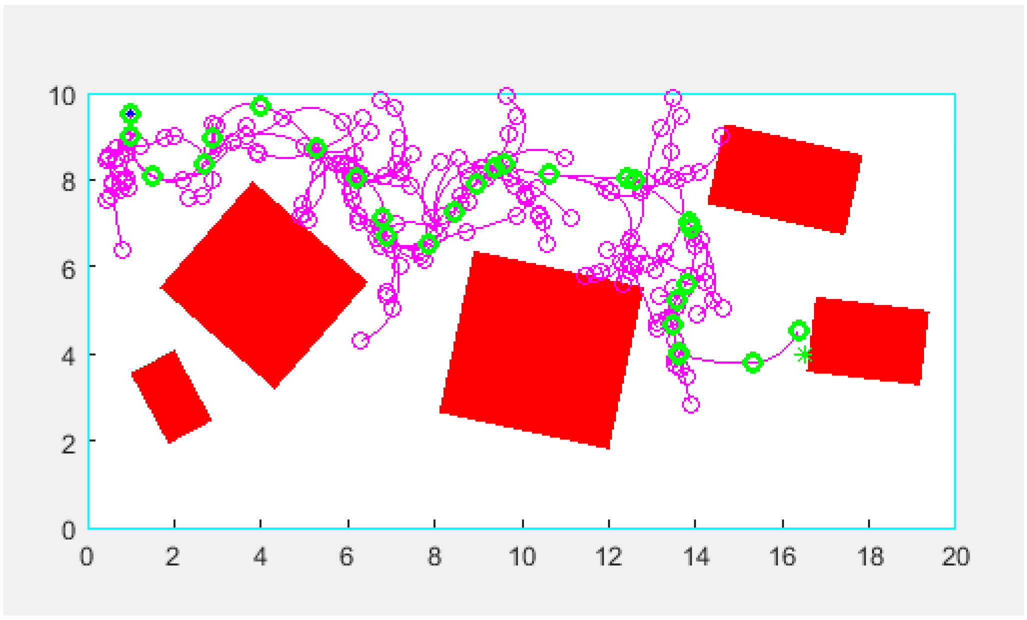
1. Extend Tree A by adding a new node (node x) within a specified distance, d
2. Find closest node (node y) in Tree B to node x
3. Check if you can connect the 2 bridges using the new node (with straight lines)
    a. If so, add edge between x and y, path is complete and thus we can run shortest path algorithm to end our algorithm
    b. Else, Swap Tree A with Tree B, and repeat process

NOTE: the dynamic constraints (motion primitives) define the allowable extension of the new node in step A.

The actual algorithm used was a single tree starting at the start node (as specified in the question). This means that the algo does not switch in between the two trees, are previously specified. However, this can take longer depending on the configuration of the start and end points. The core sequence remains the same:
1. Choose a node
2. Extend K nodes from chosen node
3. For each new node, if conditions (boundary and free space) are met, then we add a vertex and an edge to the new node.
4. Repeat for every node.
5. Once a path is complete (from end to start node), we find and plot final path through back tracing

Due to the primitive motion constraints, the orientation of the robot is required.
Hence, when a new node is created, it must be created either directly ahead, to the left, or to the right (90 degrees) d distance away from the current node.
The collision checking of the edge must be done with either straight lines or a curve of radius = 1m
Hence the max linear travel distance is set to d = sqrt(1^2 + 1^2) = 1.414. This is the distance that will be used for creating new nodes. Due to the constraints, each existing node will generate 5 expansion nodes with a distance d = 1.414 at relative angles of [-90,-45, 0, 45, 90] degrees. Each node will check if it is in a free space before any calculations on it are made. The more nodes that exist in the algorithm, the more will be added. This will have an exponential computational time. However, due to the dynamic constraints with our 1.414m

expansion distance, the node creation will not be as random as in most RRT algorithms. However, the dynamic constraints have no been implemented into the code due to a lack of time and energy.



However, since I couldn’t get the polygon world to work with the map you provided (struggled for 3 hours with it :)), I just generated an RRT algorithm without the dynamic constraints (but with the correct start and end position). Where the green dots are the path chosen for traversing the arbitrary obstacles.

Q3D. PRM was faster since I did it correctly. Couldn’t implement dynamic conditions for RRT nor the proper mapping background.
Based on the results from the simulation, the time required to run the PRM algorithm was 1.85 seconds which gave a path length of 31.3 meters.
The time required to run the RRT algorithm (on the wrong map) was significantly longer: 21.95 seconds with a path length of 22.8 meters (which isn’t the best comparison due to the difference in maps).
For simpler maps, PRM generally triumphs due to it speed, although it is suboptimal.