Programming Assignment 2: Probabilistic State Estimation

Your project's report is due Tuesday April 10. You will be asked to illustrate how your inference algorithm and particle filter approaches work in a short 15-minute meeting on Monday April 16. The standard 5% bonus will be awarded for reports typesetted in LaTeX.

Part A: Bayesian Networks

Every year, credit card companies lose millions of dollars due to frauds resulting from lost or stolen cards. The financial industry has turned to AI for solutions to the fraud detection problem. Intuitively, credit card holders tend to make purchases following a certain pattern. A fraud is likely to happen when this pattern is broken. In the first part of this assignment you will implement a simple fraud detection system that employs Bayesian Networks. Computations in your Bayesian network should be done both manually and with the assistance of an approximate inference algorithm.

Suppose you are working for a financial institution and you are asked to implement a fraud detection system. You plan to use the following information:

- When the card holder is traveling abroad, fraudulent transactions are more likely since tourists are prime targets for thieves. More precisely, 1% of transactions are fraudulent when the card holder is traveling, where as only 0.4% of the transactions are fraudulent when she is not traveling. On average, 5% of all transactions happen while the card holder is traveling. If a transaction is fraudulent, then the likelihood of a foreign purchase increases, unless the card holder happens to be traveling. More precisely, when the card holder is not traveling, 10% of the fraudulent transactions are foreign purchases where as only 1% of the legitimate transactions are foreign purchases. On the other hand, when the card holder is traveling, then 90% of the transactions are foreign purchases regardless of the legitimacy of the transactions.
- Purchases made over the internet are more likely to be fraudulent. This is especially true for card holders who don't own any computer. Currently, 60% of the population owns a computer or smart phone and for those card holders, 1% of their legitimate transactions are done over the internet, however this percentage increases to 2% for fraudulent transactions. For those who don't own any computer or smart phone, a mere 0.1% of their legitimate transactions is done over the internet, but that number increases to 1.1% for fraudulent transactions. Unfortunately, the credit card company doesn't know whether a card holder owns a computer or smart phone, however it can usually guess by verifying whether any of the recent transactions involve the purchase of computer related accessories. In any given week, 10% of those who own a computer or smart phone purchase (with their credit card) at least one computer related item as opposed to just 0.1% of those who don't own any computer or smart phone.

Question 1: Construct a Bayesian Network to identify fraudulent transactions.

What to hand in: Show the graph defining the network and the Conditional Probability Tables associated with each node in the graph. This network should encode the information stated above. Your network should contain six nodes corresponding to the following binary random variables:

- OC: card holder owns a computer or smart phone.
- $Fraud$: current transaction is fraudulent.
- $Trav$: card holder is currently traveling.

- *FP*: current transaction is a foreign purchase.
- *IP*: current purchase is an internet purchase.
- *CRP*: a computer related purchase was made in the past week.

The arcs defining your Bayesian Network should accurately capture the probabilistic dependencies between these variables.
[10 points]

Question 2: What is the prior probability (i.e., before we search for previous computer related purchases and before we verify whether it is a foreign and/or an internet purchase) that the current transaction is a fraud? What is the probability that the current transaction is a fraud once we have verified that it is a foreign transaction, but not an internet purchase and that the card holder purchased computer related accessories in the past week?

Compute these probabilities both manually and using your implementation of an approximate inference algorithm. You can use the programming language of your preference to implement. Show how the output of the approximate inference algorithm approaches the correct solution as the number of samples increases. It should be possible to answer any prior or conditional probability with your approximate inference algorithm by providing the query and evidence variables to your program.

What to hand in: Indicate what queries (i.e., $Pr(variables|evidence)$) you used to compute those probabilities. Note that a maximum of half the marks are earned if you answer correctly the question by doing the computations only by hand or only using your program.
[15 points]

Question 3: After computing those probabilities, the fraud detection system raises a flag and recommends that the card holder be called to confirm the transaction. An agent calls at the home of the card holder but she is not home. Her spouse confirms that she is currently out of town on a business trip. How does the probability of a fraud changes based on this new piece of information? Answer similar to question 2.
[5 points]

Question 4: Suppose you are not a very honest employee and you just stole a credit card. You know that the fraud detection system uses the Bayesian network designed earlier but you still want to make an important purchase over the internet. What can you do prior to your internet purchase to reduce the risk that the transaction will be rejected as a possible fraud? Answer similar to previous questions. Report the action taken and indicate by how much the probability of a fraud gets reduced.
[5 points]

Part B: Particle Filters

In this part of the project you are asked to implement a particle filter algorithm to locate the position and orientation of a moving agent, such as a robot, in a known map of the environment. You must run a set of experiments and report your results and conclusions. You can use the programming language of your preference to implement and visualize the results of your algorithms.

Please, make an effort to work in pairs for this assignment.

Question 1: Create a simple interface to visualize an indoor 2D environment. Use maps of size at least 500x500 pixels in your work. Note, that the underlying world is assumed to be continuous, even though that you will have to discretize it in order to visualize it.

Your interface should output a file (e.g., “map.txt”) that corresponds to the map of the environment. The map is discretized at a pixel level. The map file should look as follows:
n
$x_1 y_1$
$x_2 y_2$
...
$x_n y_n$
where n is the number of landmarks in the world (points recognizable by the robot inside the world), and x_i, y_i are the coordinates of the i-th landmark. Note again, that although the map is *discretized for representation purposes*, the underlying world is assumed to be *continuous*.

Use black points to represent landmarks in your environment. The white space will correspond to accessible space for the robot. Using your interface define potential paths for an agent in the accessible space. The paths correspond to sequences of straight line segments. For example, you can use your interface to click on a series of points in the environment. Then the path is the sequence of segments that connect these points.

These paths correspond to the “ground truth” for your problem; the motion actually executed by the agent in the environment. Given such paths, the robot either moves forward along a straight-line segment or rotates in place when switching between straight-line segments. The agent’s motion is continuous. For each straight line segment assume that the agent is executing it with a constant velocity (e.g., 2 pixels per second). For each rotational movement assume that the agent is executing it with a constant rotational velocity (e.g., 1 radian per second). After completing a rotation the agent is able to achieve instantaneously the forward velocity for the next segment. Similarly, after completing a straight line segment, the agent is able to achieve instantaneously the rotational velocity for the next rotation.

Your interface should then output a file (e.g., “path.txt”) that represents the paths been created as a sequence of points and orientations (configurations $< x, y, \theta >$) along the path. These configurations must occur every second assuming the agent executes the defined path with your predefined constant velocities. Make sure that the sequence of configurations takes into account the fact that the agent has to rotate between consecutive straight-line motions. The file should look as follows:
N
$x_0\ y_0\ \theta_0$
$x_1\ y_1\ \theta_1$
...
$x_N\ y_N\ \theta_N$
where N is the number configurations stored in the file.

For the following questions, experiment with different maps and paths. Submit two maps and corresponding paths for these environments that appear interesting to you and useful to address the following questions. Include the maps/paths to your report and address the consecutive problems for them. Unless specified otherwise, assume that the agent’s initial location is always the same at coordinates (10,10) and orientation 0 (looking right - clockwise direction being the positive direction: meaning that at orientation $\frac{\pi}{2}$ the agent moves up).
[5 points]

Question 2: Once you have the interface for creating the “ground truth” for your experiments, you need to simulate the sensing input available to your agent. The sensing input available to the agent at each step i is: (a) the agent’s change in orientation [rot_i], (b) the displacement from the previous measurement [$disp_i$] (as if your agent was using an odometer), (c) the landmarks within a visibility radius of the robot (e.g., 40 for a 500x500 map - experiment with different values) and (d) for each landmark the relative angle between the robot’s orientation and the direction to the

landmark (e.g., if the robot looks up and has orientation $\frac{\pi}{2}$ and a landmark is at orientation $\frac{\pi}{4}$ relative to the robot, then the relative angle is $-\frac{\pi}{2}$ - imagine that the robot carries a panoramic camera, it can identify landmarks and compute the angle to them relative to its orientation).

Your program must be able to receive a path file (a "path.txt" file from the previous question) and output a file that represents the sensing input, which could look like the following description:
N
0 0 2 [3 ($-\frac{pi}{2}$)] [2 $\frac{pi}{4}$]
$disp_1$ rot_1 2 [3 ($-\frac{pi}{4}$)] [2 $\frac{pi}{2}$]
$disp_2$ rot_2 1 [3 ($-\frac{pi}{8}$)]
. . .
where N is the number of sensing inputs available (equal to the number N of configurations in the input "path.txt"). Then each line i in the file contains:

(i) $disp_{i,j}$: the displacement between configurations $i-1$ and i (note that for the very first measurement we get a 0 displacement and rotation),

(ii) rot_i: the rotation of the agent as measured by the agent's sensors at step i,

(iii) the number of landmarks visible by the agent's sensors within the visibility range,

(iv) and for each landmark a pair of data: its identity and its relative angle. For instance, in the above example and in the first time step the agent can sense two landmarks, the landmark with identity 3 at $-\frac{pi}{2}$ angle and the landmark 2 with angle $\frac{pi}{4}$.

If there is no noise in the sensing measurements then the perfect measurements $disp_j^*$ and rot_i^* can be computed given configurations $< x_i, y_i, \theta_i >$ from an input path file as follows:

$$disp_j^* = \sqrt{(x_{i-1} - x_i) * (x_{i-1} - x_i) + (y_{i-1} - y_i) * (y_{i-1} - y_i)}$$

$$rot_i^* = \text{angular\_difference}(\theta_i, \theta_{i-1})$$

Real sensors, however, suffer from noise. We will assume in this project that the noise is Gaussian. For a given input path file "path.txt", your program should output five different sensing files, each for a different sensing model (call the output files "sensing_0.txt", "sensing_1.txt", etc.):

1. A zero noise / perfect sensor model, which returns the displacement and rotation parameters computed by the above expressions.
2. A model that involves Gaussian noise only in the displacement parameter with standard deviation: 0.25 pixels and zero mean.
3. A model that involves Gaussian noise only in the rotation measurement with zero mean and standard deviation: 0.01 radians.
4. A model that combines both of the above Gaussian models .
5. A model similar like the previous one but with higher standard deviations: 0.5 pixels for displacement and 0.03 radians for rotations.

In order to compute the Gaussian noise you have to compute random numbers that follow the Gaussian distribution (for 0 mean):

$$\frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{x^2}{2\sigma^2})$$

where σ is the standard deviation. The webpage, under the "Deliverable" link, provides code (C++ but easily transferable to Java) that produces random numbers that follow a Gaussian probability density function for a given mean and standard deviation.

The assumption is that there is no noise in the identification of landmarks. All of the landmarks within the visibility range are identified correctly. There can be, however, noise in the measurement of the relative angle to landmarks. For the perfect sensing model assume that there is no such noise. For the following three models assume Gaussian noise with zero mean and standard deviation of 0.01 radians. For the last model assume standard deviation of 0.03 radians.
[10 points]

Question 3: Use your interface to integrate all the sensing inputs and visualize the corresponding paths on the map (use the "map.txt" and the "sensing_?.txt" files). Compare the resulting paths with the "ground truth" path on the same map in your report and report the average error in distance and orientation over all the configurations (10 comparisons total: 2 map/path pairs and 5 sensing models).

This means that you have to reconstruct the agent's position x_i, y_i and orientation θ_i from the sensing input $disp_{i-1,i}$ and rot_i. If the agent at time step i is at state $X_i =< x_i,\ y_i,\ \theta_i >$, then on the next time step it should be at:

$$X_{i+1} =< x_i + disp_{i+1} * cos(\theta_i),\ y_i + disp_{i+1} * sin(\theta_i),\ \text{angular\_summation}(\theta_i, rot_i) > . \quad (1)$$

The perfect sensor model should get you a path that is very close to the "ground truth" path (the one specified in the initial "path.txt" file). Nevertheless, discretizations can cause discrepancies even in the case of the perfect sensor models. For debugging purposes it might be beneficial to have at least one path that contains line segments and rotations with a duration of an integer number of seconds (this means the line segments must have a length of k*2 pixels and the rotations an angle of l radians, where k and l are integers). For such paths, the reconstructed paths from the perfect sensor model should not suffer as much from discretization errors.
[10 points]

Question 4: In the previous question we have not used the fact that the sensing input also contains information about the landmarks visible from the agent. Here we will attempt to utilize this information in conjunction with a filtering algorithm to improve the localization estimate.

The algorithm you will implement to achieve this is a particle filter. A particle filter algorithm is a sampling-based approach to implement the basic Bayesian equation for solving the filtering problem:

$$P(X_{i+1}|e_{1:i+1}) = \alpha P(e_{i+1}|X_{i+1}) \sum_{X_i} P(X_{i+1}|X_i) P(X_i|e_{1:i})$$

In the above equation a state X_{i+1} corresponds to a configuration $< x_{i+1},\ y_{i+1},\ \theta_{i+1} >$, while an evidence e_{i+1} corresponds to the landmarks that are visible at time step $i+1$ and the relative angles to them. As the above equation specifies, the approach is an iterative algorithm where the new belief state distribution $P(X_{i+1}|e_{1:i+1})$ after reading the $i+1$-th sensing input depends upon the previous belief distribution $P(X_i|e_{1:i})$. To implement Bayesian filtering we need:

- The initial (prior) belief distribution $P(X_0)$. For this question we assume that we know the initial position of the agent (as specified in part A). Consequently, the initial distribution has a probability 1 for the true initial state and 0 for every other state.
- A transition model: $P(X_{i+1}|X_i)$. We will use part of the sensing input to define the transition model, in particular the parameters $disp_{i,i+1}$ and rot_{i+1}. If the agent at time step i is at state $X_i =< x_i,\ y_i,\ \theta_i >$, then we must compute the state X_{i+1}. In the previous question, we gave equation 1 that allows us to compute X_{i+1} assuming that $disp_{i,i+1}$ and rot_{i+1} are perfect measurements. But the transition model is a probability distribution and we know that the $disp_{i,i+1}$ and rot_{i+1} parameters are noisy measurements.

  To draw states X_{i+1} from the probability distribution $P(X_{i+1}|X_i)$ do the following. Given the

sensing input $disp_{i+1}$, draw a random number from the Gaussian distribution

$$e_{disp} = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{x^2}{2\sigma^2})$$

where the σ is the standard deviation defined in the fourth sensor model in problem B (0.25 pixels). Add e_{disp} to the displacement parameter: $disp^+_{i+1} = disp_{i+1} + e_{disp}$, to get a noisy measurement. Do the same for the direction parameter (standard deviation: 0.01 radians) to compute a noisy direction parameter rot^+_{i+1}. Then a state X_{i+1} drawn from the probability distribution $P(X_{i+1}|X_i)$ is:

$$X_{i+1} =< x_i + disp^+_{i+1} * cos(\theta_i),\ y_i + disp^+_{i+1} * sin(\theta_i),\ \text{angular\_summation}(\theta_i, rot^+_i) > .$$

- An observation model: $P(e_{i+1}|X_{i+1})$. As we mentioned above, we use the landmark information as the evidence for the filtering process. Given a state X_{i+1} we can compute from the map the set of landmarks we expect to observe from this location. Build a probability distribution that expresses an appropriate observation model and explain how it works in your report.

Given the definition of the above three parameters we can execute the particle filter algorithm. In particular the steps that you have to implement are the following:

1. Initialize a population of P particles according to the initial (prior) belief distribution (all the particles in the known initial location). Each particle corresponds to a path estimate (initially just the known state) and a weight.
2. Set $i = 0$.
3. Read the next sensing input ($disp_{i-1,i},\ rot_i,\ l,\ \ldots$).
4. For each particle p:
   4.1. Read the state X^p_{i-1} from particle p
   4.2. Use the process described above to compute X^p_i given X^p_{i-1} according to the transition model $P(X^p_i|X^p_{i-1})$.
   4.3. Compute the landmarks observable from state X^p_i according to the map.
   4.4. Assign weights to the particles given the observation model you proposed.
5. Resample a new population of particles: Do not sample particles with weight 0, sample among all the particles with positive weight and according to their weight to create a new population with P particles. If there are no particles with weight 1, then retain all the particles.
6. As long as there are sensing inputs return to line 3.

Implement and test the above algorithm for the 2 pairs of maps/paths and 5 sensing models. [30 points]

Question 5: Follow an object-oriented approach in your implementation. Include classes for:

- states (configuration $< x, y, \theta >$),
- observations (displacement $disp$, rotation rot, number of landmarks l, landmark ids and angles)
- landmarks (location and id),
- the map,

- paths (a sequence of states),
- particles (a path, a weight, and the iteration index) and
- a particle filter.

The functions for the transition model and the observation model should be clearly identifiable in your code. Mention in your report where these functions can be found and briefly describe the object-oriented nature of your implementation.

You should probably use the same code base for all questions of part B of the project (creating the ground truth, simulating the sensing input and running the particle filter); avoid creating three completely different programs (although you might want to create three different executables).

Visualize the particles at each iteration of the algorithm. This will also be beneficial for debugging purposes.
[10 points]

Question 6: Specify in your report for which number of particles was the algorithm able to relatively keep track of the agent's true location (if able to do so). Remember that the higher the number of particles the more accurate the approach will be. At the same time it will be computationally more expensive as the number of particles increases. Furthermore address the following issues in your report:

- Describe your experience from your experiments about the effectiveness of the algorithm in your report. In which cases does the algorithm work better?
- Do you have any ideas how the algorithm can be improved? What kind of additional information would assist the algorithm in becoming more effective and accurate? How would you change the transition or the observation model in that case?
- What if the environment contained obstacles? How would you need to change the transition and observation model?

Provide some graphical examples in your report of the paths estimated by the algorithm. A path estimated by the algorithm corresponds to the history of the particle in the last iteration of the algorithm which has the highest weight. Report the average error of the particle filter's best estimation.

Experiment with different parameters and provide statistics (e.g., study the effects of the visibility range, the density of landmarks, the noise in the sensing models, or the noise introduced in the transition and observation model relative to the noise of the sensing model, etc.)
[up to 20 points depending on the amount of statistics provided and conclusions drawn - for a total of 120 points]