

# CS 228, Winter 2008

## Programming Assignment #2 - Robot Localization and Mapping

### Handout #14

---

This assignment is due at 11:59 PM on February 20.

## Introduction

In this assignment you will use probabilistic inference to track a robot through time and build a map of the world in which it lives. You will use the code that you developed for Programming Assignment #1 to perform inference and will see how observing aspects of the world through a measurement model can significantly improve your ability to localize compared to only using a motion model that captures the probabilistic motion of the robot.

Our robot is a minesweeper named Roxer, and she lives in a flat grid world composed of  $N \times N$  squares. We denote her position at time  $t$  by  $X^{(t)} \in [1, N] \times [1, N]$ , where the components of  $X$  index the grid position that she currently resides in. She moves through the world by taking a step in one of the 8 compass directions (North, Northeast, East, Southeast, etc.). We use  $H^{(t)}$  with values  $2, \dots, 9$  to denote the direction that she wishes to travel at time  $t$ , where  $H^{(t)} = 1$  means that Roxer decides not to move at all. Figure 1 shows a sample world and example location and heading of Roxer.

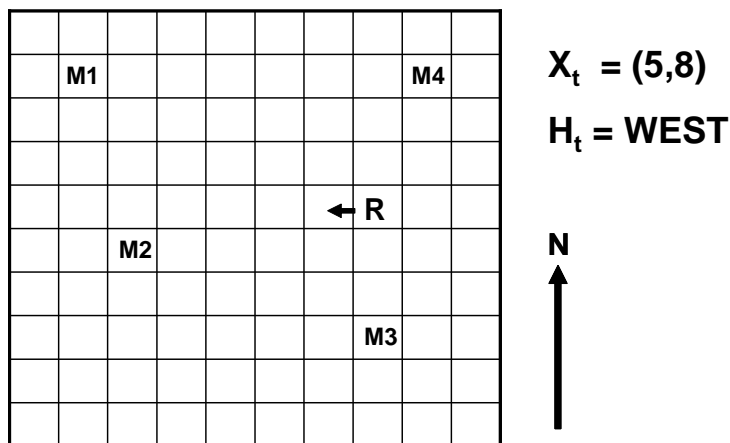


Figure 1: Schematic diagram of Roxer World.

However, because Roxer is getting old and her machinery doesn't work as well as it used to, she sometimes accidentally slips into the wrong square. Her movement is therefore probabilistic, according to a **transition model**:

$$P_{\text{Transition}}(X^{(t+1)} | X^{(t)}, H^{(t)})$$

Roxer's job is to locate mines in her world, so that people can safely avoid them when they pass through the grid. The world contains  $K$  mines, with locations denoted by  $\mathbf{M} = M_1 \dots M_K$ . We will refer to these locations as "landmarks". Roxer is designed so that she can safely stand on top of mines without detonating them, and is equipped with a sensor that can pick up the electromagnetic radiation emitted from the mines. Her sensor provides a signature that uniquely identifies the mine with no error, and also a noisy quantized distance to the mine. Hence, Roxer knows about all the  $K$  mines, and at each timestep  $t$ , her sensors give her noisy estimates  $D_k^{(t)}$  for the *distance* to each landmark from Roxer's current location. This amounts to a **sensor model** for each landmark:

$$P_{\text{Sensor}}(D_k^{(t)} \mid M_k, X^{(t)})$$

The probabilistic setup here induces a DBN as shown in Figure 2. You will use this model to simultaneously track Roxer in the world and localize the mines using different probabilistic algorithms.

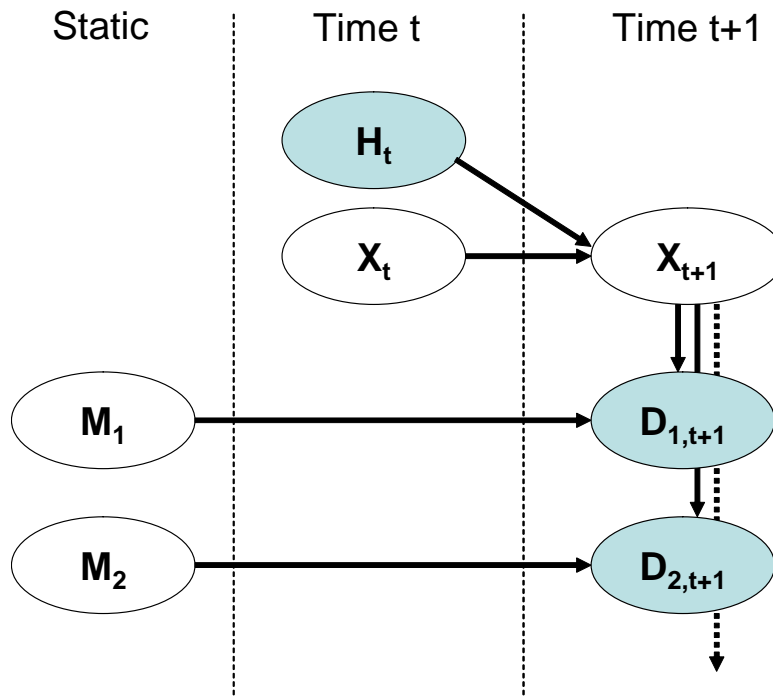


Figure 2: DBN for Roxer's world.

## Overview

You will implement the following different algorithms, which allow us to track the robot and determine the location of the landmarks in the world.

1. **Forward Inference (Tracking)** - The algorithm you will implement is precisely the DBN tracking inference task discussed in the course reader in Section 15.2.1. You are given an initial belief state  $\sigma^{(0)}$  over the locations of Roxer and the mines (for  $t = 0$ ), together with the headings  $H^{(t)}$  and sensor readings  $D_k^{(t)}$  for all times. For each time step  $t$ , you will use your current belief  $\sigma^{(t)}$  over  $X^{(t)}$  and  $\mathbf{M}$ , along with the transition model and the sensor model to obtain a belief  $\sigma^{(t+1)}$  over  $X^{(t+1)}$  and  $\mathbf{M}$ .
2. **Unrolled DBN Inference** - You will also unroll the DBN to produce a Bayes Net over the time window  $t = 1 \dots T$ , and perform approximate inference (using loopy belief propagation) in the unrolled Bayes Net to determine the beliefs over mine locations and Roxer's path.
3. **Sliding Window DBN Inference** - You will also implement a variant of the above method, where instead of unrolling the network over the entire range of observations  $1 \dots T$ , we only do it for a "window"  $W$ , so that the unrolled network is over the time steps  $1 \dots W$ . We run approximate inference (conditioned on the observations in the window) in the unrolled network, and store the posterior distribution of all variables. We then "slide" the network over, unroll it over the window  $2 \dots W + 1$ , and use the previously computed posteriors on  $X^{(2)}$  and  $M_k$  as priors on  $X^{(2)}$  and  $M_k$ . We repeat this process until our window reaches the last time step  $T$ .

In the following sections we will discuss the exact steps you need to take to implement the above algorithms.

## Part 1: Code (75 points)

All code for this assignment should be copied from

`/afs/ir/class/cs228/ProgrammingAssignments/PA2/assignment.`

Your job is to fill in any missing portions of code - marked with YOUR CODE HERE - in the files described below. You will not have to implement any factor operations or approximate inference: those are provided for you in the `Utils/` subdirectory.

### Tests

`TestPA2.m` contains a series of tests to evaluate the correctness of your implementation. Each test uses one of your completed functions, where sample inputs and outputs are loaded in, and the test passes if the output produced by your implementation matches the correct outputs. Note that when you submit your final implementation it will be tested with different inputs and outputs.

You are provided with two sets of tests, corresponding to two different worlds `world1.mat` and `world2.mat` and can switch between them to make sure your implementation is correct.

### [45 points] Forward Inference (Tracking)

Read section 15.2.1 in the course reader. Make sure you understand the difference between  $\sigma^{(t)}$  and  $\sigma^{(t)}$ . Note that in our case, the state variables are *both*  $X^{(t)}$  and  $\mathbf{M} = \{M_1, \dots, M_K\}$ . Since

the mines do not move, the variables  $M_k$  are static over time (i.e. we do *not* have variables  $M_k^{(t)}, M_k^{(t+1)}$ ).

As discussed in class, and in the reader, due to *DBN entanglement*, an exact representation of the belief state  $\sigma^{(t)}(X^{(t)}, \mathbf{M})$  can have no compact factored representation. Therefore, for efficiency, we will resort to an *approximate* representation of our belief state:  $\hat{\sigma}^{(t)}$ , which will assume the variables  $X$  and  $M_k$  are independent:

$$\hat{\sigma}^{(t)}(X^{(t)}, \mathbf{M}) = \hat{\sigma}^{(t)}(X^{(t)}) \cdot \hat{\sigma}^{(t)}(\mathbf{M}) = \hat{\sigma}^{(t)}(X^{(t)}) \prod_k \hat{\sigma}^{(t)}(M_k)$$

We can see that our approximate belief state  $\hat{\sigma}^{(t)}(X^{(t)})$  is a product of independent beliefs over the robot's location and each landmark location. Each one of those beliefs is a factor over (only) one variable, in contrast to the *exact* belief state representation, which would require a factor over all the variables.

1. Assuming an *exact* belief state representation, show how you can compute  $\sigma^{(t+1)}(X^{(t+1)}, \mathbf{M})$ . That is, write a formula  $F$  such that  $\sigma^{(t+1)}(X^{(t+1)}, \mathbf{M}) = F(\sigma^{(t)}(X^{(t)}, \mathbf{M}))$ .
2. Assuming an *exact* belief state representation, show how you can compute  $\sigma^{(t+1)}(X^{(t+1)}, \mathbf{M})$ . That is, write a formula  $G$  such that  $\sigma^{(t+1)}(X^{(t+1)}, \mathbf{M}) = G(\sigma^{(t+1)}(X^{(t+1)}, \mathbf{M}))$ .

Now we are going to see how these formulas change when we plug-in the *approximate* belief state:

3. Define  $\hat{\sigma}^{(t+1)}(X^{(t+1)}, \mathbf{M}) = F(\hat{\sigma}^{(t)}(X^{(t)}, \mathbf{M}))$ . Show that

$$\hat{\sigma}^{(t+1)}(X^{(t+1)}, \mathbf{M}) = \hat{\sigma}^{(t+1)}(X^{(t+1)}) \cdot \hat{\sigma}^{(t)}(\mathbf{M})$$

Write the formula for  $\hat{\sigma}^{(t+1)}(X^{(t+1)})$ .

4. Define  $\hat{\sigma}^{(t+1)}(X^{(t+1)}, \mathbf{M}) = G(\hat{\sigma}^{(t+1)}(X^{(t+1)}, \mathbf{M}))$ . Show how you can compute  $\hat{\sigma}^{(t+1)}(X^{(t+1)})$  and  $\hat{\sigma}^{(t+1)}(M_k)$ .

Write down your answers. You will use them in your answer to question 1 in the questions section, as well as in the implementation of the missing code in the following files:

1. **[10 points] RobotSigmaDotUpdate.m**: Calculates  $\hat{\sigma}^{(t+1)}(X^{(t+1)})$  as a function of  $\hat{\sigma}^{(t)}(X^{(t)})$  and the heading  $H^{(t)}$ , using the transition model.
2. **[10 points] RobotBeliefUpdate.m**: Calculates  $\hat{\sigma}^{(t+1)}(X^{(t+1)})$  as a function of  $\hat{\sigma}^{(t+1)}(X^{(t+1)})$ ,  $\hat{\sigma}^{(t)}(M_k)$  and the observations  $D_k^{(t+1)}$ , using the sensor model.
3. **[15 points] LandmarksBeliefUpdate.m** Calculates the same thing for  $\hat{\sigma}^{(t+1)}(M_k)$ .
4. **[10 points] ForwardInference.m** Using what you implemented in the above files, implement the full Forward Inference (tracking) algorithm.

### [15 points] Unrolled DBN Inference

**UnrollRoxerDBN.m** Implement the unrolling of the DBN. Most of the code is there for you, and your job is to set the factors properly. Since we provide the approximate inference code for you along with the function `UnrolledDBNInference.m` this is all you need to do to complete the unrolled DBN inference implementation.

## [15 points] Sliding Window DBN Inference

**UpdateUnrolledRoxerDBN.m** This function takes a previously unrolled DBN over a time window (i.e. not over all time steps) and advances the sliding window forward by eliminating the clusters including variables that move out of the window scope, and adds clusters corresponding to the new variables introduced by the sliding window. Your job is to set the factors correctly.

**Before submitting your code, make sure you update the Students.m file with information about your group members.**

### Details

Before starting with your implementation, make sure you understand the factor data structures from the Programming Assignment 1. In particular, you should be comfortable multiplying and marginalizing factors, and understand the role of the `.var` field when multiplying two factors. For example, if you have a transition model factor and wish to multiply it with a factor representing the beliefs over a state, make sure that you set your belief factor has the proper `.var` field to match up with the variable order of the transition model factor. In addition, review the graph data structure used in approximate inference from Programming Assignment 1.

At this point you might want to get yourself introduced with some real data. Check the file `TestPA2.m`. Run the code up to "Part 1" (not including). It will load the factors that make up the transition model and the observation model. The observed variables  $H^{(t)}$  and  $D_k^{(t)}$  are given in the variables `HEADING` and `OBSERVATION`. `HEADING(t)` is the direction the robot took at time step  $t$ , and `OBSERVATION(t,k)` is the sensor reading at time  $t$  for landmark  $k$ . The initial belief for robot location is in `ROBOT_BELIEF_0`, and the initial beliefs for landmark locations are in `LANDMARK_BELIEFS_0`.

It is important that you understand the return values of the following functions:

1. **TransitionModel.m** - This function takes in a desired heading, and returns a CPD that provides the probability of Roxer's next location given her current location (i.e., the distribution  $P(X^{(t+1)} | X^{(t)}, H^{(t)})$ , for a particular  $H^{(t)}$ ). Pay particular attention to the variable order of the returned factors. Instead of returning the full transition model in one factor, we return a cell array of conditional transition models which you can index using the `HEADING` the robot took at the previous time step.
2. **SensorModel.m** - This function either returns the full sensor model ( $P(D_k^{(t)} | X^{(t)}, M_k)$ ), or if you pass in a specific observed  $D$ , it returns a factor over the other two variables where the appropriate entry is selected from the full distribution. Pay particular attention to the variable order of the returned factors. Instead of using the full sensor model in one factor, we will use a cell array of conditional sensor transition models which you can index using the `OBSERVATION` array.

The factor values returned by the above functions might be hard to probe since they are flattened (`.val` is a one-dimensional vector). It is not necessary to un-flatten these values for this problem set. However, we provide the following two examples so that you can get a better feel for the models:

If Roxer's current location is [3,3] and she is heading north ( $H^{(t)} = 8$ ) then the distribution over her next location is given by:

```
Z = reshape(TRANSITION_MODELS{8}.val, N, N, N, N); squeeze(Z(3,3,:,:))
```

If Roxer is at location [3,3] and there is a mine over location [1,5], then the distribution over the reading from the sensor is given by:

```
Z = reshape(FULL_SENSOR.val, N, N, N, N, 10); squeeze(Z(3,3,1,5,:))
```

We also provide you with the true final (time step  $T$ ) robot and landmark locations in `TRUTH` and `TRUE_LANDMARK_LOCATIONS`, respectively. You can use the functions `AnalyzeResults` and `ViewState` to visualize the beliefs the various inference algorithms return, and compare them to the truth.

Pay careful attention to the comments in the code, as they are meant to make your job clearer and easier.

## Part 2: Questions (25 points)

Answer the following questions. Keep your answers concise.

1. [7 points] Given a current belief state over Roxer's location  $\hat{\sigma}^{(t)}(X^{(t)})$  and current estimates of the mine locations  $\hat{\sigma}^{(t)}(M_1) \dots \hat{\sigma}^{(t)}(M_K)$ , the current heading  $H^{(t)}$  and the observations from the next time step  $D_1^{(t+1)} \dots D_K^{(t+1)}$ , give the formulas you used in your Forward Inference algorithm to compute:  $\hat{\sigma}^{(t+1)}(X^{(t+1)})$ ,  $\hat{\sigma}^{(t+1)}(M_1), \dots, \hat{\sigma}^{(t+1)}(M_K)$ .

Please attempt to use reasonable notation in your README file (for example  $\text{\LaTeX}$  style notation would be great; anything else that can be parsed is also fine.)

2. [6 points] How do the (asymptotic) memory requirements vary between the unrolled inference and the forward-backward inference approaches? State your answer in terms of  $T$ ,  $N$  and  $K$ .
3. [6 points] Our implementation of Sliding Window inference was not optimized, and required many iterations of loopy belief propagation for each network. How could you reuse information from the calibrated cluster graph on the unrolled DBN over one window to speed up inference over the unrolled DBN over the next window of observations?
4. [6 points] Describe any scenario where you expect inference on the unrolled DBN to produce more accurate localization results than Forward Inference.

**You must include the answers to these questions in an ASCII text file called README when you submit your code.**