Project 2.2 (CSCI 360, Spring 2018)

3 Points

Due date: March 30st, 2018, 11:59PM PST

1 Introduction

In this project, we will consider introducing sensor and actuation *noise* into Wheelbot's environment. Sensor noise means that Wheelbot's sensors are not completely reliable. With some probability, they will provide Wheelbot with incorrect values. Actuation noise means that Wheelbot no longer knows the state resulting from taking an action with certainty. It may "slip" and end up in an incorrect state. The tools of probability theory allow us to reason about this uncertainty in a principled manner. In this project, you will define *sensor* and *transition* models (in the form of probability distributions) for Wheelbot in a nondeterministic environment with only hidden obstacles, which now act as walls that prevent Wheelbot's movement rather than killing Wheelbot.

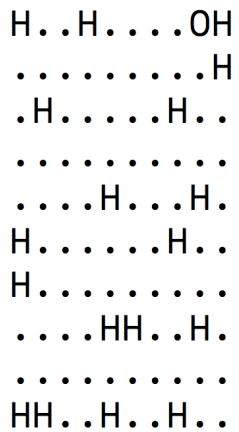

Figure 1: A screenshot of the text-based simulator for project 2.

2 Simulator

2.1 Wheelbot

For this assignment, Wheelbot has the following actions and sensors:

2.1.1 Wheelbot Actions

1. **MOVE_UP**: Moves Wheelbot one grid space up (North)
2. **MOVE_DOWN**: Moves Wheelbot one grid space down (South)
3. **MOVE_LEFT**: Moves Wheelbot one grid space left (West)
4. **MOVE_RIGHT**: Moves Wheelbot one grid space right (East)

2.1.2 Wheelbot Sensors

1. **Local Wall Sensor**: Returns a noisy estimate of the number of hidden obstacles immediately adjacent to Wheelbot's current grid cell location (in the directions up, down, left, right). To use this sensor, call this→simulator.getCurrentNumWalls(), which returns an integer value (an observation, o) in the set $O = \{0, 1, 2, 3, 4\}$.

2.1.3 Other functionality

The following important functionality is required for this project:

1. You can get the dimensions of the grid by calling this→simulator.getWidth() and this→simulator.getHeight().
2. You can get a list of all the hidden obstacles (needed for probability calculations) surrounding grid cell x by calling this→simulator.getLocalObstacleLocations(x).
3. In main.cpp, *manualControl* can be set to *true* in order to control Wheelbot's actions manually using the keyboard for debugging.
4. In main.cpp, *numHiddenObstacles* controls the number of obstacles in Wheelbot's environment. **All obstacles are hidden in this project, even though they are visualized with #.**
5. In main.cpp, set *display* to *true* to render the environment and robot at each step. Set it to $false$ to suppress rendering. This speeds up the validation process, which requires millions of steps for good estimates.

6. In main.cpp, *alpha* controls the noise in Wheelbot's transitions, while *epsilon* controls the noise in Wheelbot's observations.

3 Programming Portion

3.1 Theory

In this project, Wheelbot exists in a nondeterministic environment in which its actions and observations are noisy. The noise in its actions is controlled by the parameter α, which indicates the probability of performing the correct (intended) action in the environment. The remaining probability mass is divided equally amongst other possible transitions. For example, at location $(5, 5)$ of the 10×10 grid illustrated in Figure 1 there are no adjacent walls (grid boundaries or obstacles). If Wheelbot executes the action MOVE_UP, it will move up one grid cell to $(4, 5)$ with probability α. With probability $(1-\alpha)/3.0$, it will move down to grid cell $(6, 5)$. With probability $(1-\alpha)/3.0$, it will move left one grid cell to $(5, 4)$. Finally, with probability $(1 - \alpha)/3.0$, it will move right one grid cell to $(5, 6)$. Observe that $3((1 - \alpha)/3.0) + \alpha = 1$, so the total probability of making some transition is 1, as required for a valid probability distribution. Self-transitions are only possible when obstacles or environment boundaries are immediately adjacent to Wheelbot. For example, if Wheelbot is at the location of the 0 depicted in Figure 1, $(0, 8)$, executing the action MOVE_RIGHT keeps Wheelbot in $(0, 8)$ with probability $\alpha + (1 - \alpha)/3$. This is because the most likely transition (made with probability α) will smack Wheelbot into the obstacle at $(0, 9)$, leaving it at $(0, 8)$. If the robot slips, due to transition noise, and tries to go up one grid cell, it will smack into the top wall of the environment, again keeping Wheelbot in $(0, 8)$. This happens with probability $(1 - \alpha)/3$. With probability $(1-\alpha)/3$, Wheelbot might slip down to grid cell $(1, 8)$. Similarly, with probability $(1 - \alpha)/3$, Wheelbot might slip left to grid cell $(0, 7)$. $\alpha + (1 - \alpha)/3 + 2((1 - \alpha)/3) = 1$, as required.

The noise in Wheelbot's observations is controlled by the parameter ϵ. In this project, Wheelbot has a single sensor that tells it (noisily) how many walls – formed by obstacles – are surrounding it at its current position. **For simplicity, environment boundaries are *not* counted as walls**. With probability ϵ, this sensor reports the correct value. With probability $(1 - \epsilon)/4.0$, it will report one of the 4 possible incorrect values of this reading. For example, if Wheelbot is in cell $(0, 8)$ as before, the sensor $this \rightarrow$simulator.getCurrentNumWalls() will report 1 wall with probability ϵ. However, with probability $(1 - \epsilon)/4.0$, it will report 0. With probability

$(1-\epsilon)/4.0$ it will report 2. With probability $(1-\epsilon)/4.0$ it will report 3. With probability $(1-\epsilon)/4.0$ it will report 4. We see that $\epsilon + 4((1-\epsilon)/4) = 1$ as required for this to be a valid probability distribution. This sensor and transition noise has been programmed into the simulator for you already.

Let X denote the set of grid cells in the grid environment in which Wheelbot finds itself. Let A ={MOVE_UP, MOVE_DOWN, MOVE_LEFT, MOVE_RIGHT} denote Wheelbot's action set. Per the above information, it is possible to specify, for all $x \in X$, $a \in A$, and $x' \in X$:

$$P(x'|x,a) \tag{1}$$

This represents the probability of transitioning from grid cell $x \in X$ to grid cell $x' \in X$ under the action a. This is called Wheelbot's **transition model**. It encodes the information that Wheelbot needs to make informed predictions about the results of executing its actions. These probabilities are *time-invariant*, meaning that $P(x'|x,a)$ does not depend on the time t in which Wheelbot executes the transition between x and x' under action a. Clearly, since the transitions must be valid probability distributions: $\sum_{x' \in X} P(x'|x,a) = 1$ for all x, a pairs in $X \times A$.

Using the above information, we can also specify, for all $x \in X$ and $o \in O$:

$$P(o|x) \tag{2}$$

This represents the probability of seeing observation o in grid cell $x \in X$. This is Wheelbot's **sensor model**, which encodes its time-invariant uncertainty over the observations it will see in particular states. These must also be valid probability distributions, so $\sum_{o \in O} P(o|x) = 1$ for all x.

3.2 Implementation

Your task, in this project, is to implement Wheelbot's sensor and transition models **given any random environment (map)** and then to verify, via experimentation, that the frequencies of Wheelbot's transitions and observations match your computed probabilities closely. This will require you to fill out several functions in Project.cpp. **You must NOT modify the function validate(), which computes the error between the actual probabilities you calculated and the ones estimated by your experimentation procedure**. You should modify the following functions:

1. *getProbOfObsInState(int o, Point2D c)*: This function takes in an observation, $o \in O = \{0, 1, 2, 3, 4\}$, and a Point2D grid cell c. It

should compute and return the actual probability of observing o in grid cell c. Hint: see point 2. in section 2.1.3.

2. *getProbOfTrans(RobotAction a, Point2D c, Point2D n)*: In this function, you should compute and return the actual probability of taking the transition from the state *current* to the state *next* (both Point2D grid cells) under the action a.

3. *updateCounters(RobotAction a, int o, Point2D c, Point2D n)*: At each step in the simulation, the robot will take an action a from state c, resulting in new state n and observation o. You should keep counters for these events in order to estimate the frequencies of the transitions and observations Wheelbot makes (see 4. and 5.) in order to compare these frequencies with the probabilities computed in 1. and 2.

4. *getEstProbOfTrans(RobotAction a, Point2D c, Point2D n)*: After Wheelbot takes a large number of random steps, this function will be called by the *validate()* function and should return:

$$\frac{\#\{\text{transitions from grid cell } c \text{ to grid cell } n \text{ under action } a\}}{\#\{\text{transitions from grid cell } c \text{ under action } a\}} \tag{3}$$

5. *getEstObsProb(int o, Point2D c)*: After Wheelbot takes a large number of random steps, this function will be called by the *validate()* function and should return:

$$\frac{\#\{\text{observations of } o \text{ in grid cell } c\}}{\#\{\text{times Wheelbot was in grid cell } c\}} \tag{4}$$

3.3 Validation

To run the validation procedure, set $display \leftarrow false$ and $manualControl \leftarrow false$. *numSteps* controls the number of simulation steps Wheelbot uses to estimate equations 3 and 4. This will automatically call the function *validate()* after *numSteps* have been taken in the environment. If you have implemented the sensor and transition models correctly, the *validate()* procedure should report very low error values. The reference implementation achieves transition and observation error rates on the order of approximately 5.0×10^{-7} using 20 million steps to estimate equations 3, 4 with $\alpha = \epsilon = 0.95$.

4 Submission

For this project, please submit a zip archive of your modified code via Blackboard by the date and time listed at the top of this document.