Michael Ferrin: mi6034fe-s
Hector Cervantes: he3486ce-s
EDA132- Spring 2016

Assignment 2: Probabilistic Reasoning

Introduction

A robot navigates an empty room traveling one step at a time. The robot continues in a straight line with 70% probability until it hits a wall, in which case it always changes direction and continues indefinitely. Unable to see the robot's exact location, Hector and I attempt to pinpoint the exact location of the robot. The only information we are supplied with is a sensor which sometimes reports the precise location of the robot, but is often slightly inaccurate. This is the problem we created and solved, programming first the robot, then the noisy sensor, and lastly the tracking mechanism to guess, with some level of precision, the robot's actual location.

Implementation

Our room and robot are implemented in *game.c*, primarily using a matrix of arrays. The room is set to a 5x8 grid by default (this can be changed by changing the values of *n* and *m* at the top of the file) and the robot is stored as a coordinate pair that moves around the matrix one box at a time. Also in *game.c* is our sensor. Our sensor works by grabbing the location of the robot and, with given probabilities, outputs either the correct location, a nearby location, or nothing at all. It was difficult to make the sensor probabilities perfect because we didn't want the sensor to report coordinates outside of the room. Instead, if the sensor was going to report false coordinates, we had it report nothing. This results in a higher probability of getting nothing when the robot is near a wall or corner.

We began to work in reverse as we took the output of the sensor and used it in our tracking algorithm to remove the noise and locate the robot. Despite the fact that our sensor didn't report its information with precise probabilities, we assumed they were close enough and interpreted the sensor model as if it was perfect. This leaves some room for errors but was an assumption we needed to make. We created matrix to keep track of how likely the robot is to be at each coordinate location. For example, on the first turn the sensor might report (4,2) in which case we know that there is a 10% chance the robot is actually at (4,2), a 5% chance it is at (3,2) and so on. On the second turn we generated another matrix in the same way, and then overlapped these matrices to generate our transitions. We could get more accurate percentages based on the two different sensor readings in conjunction with what we knew about the robot's movements. For help with our transitions and forwarding we referenced the textbook often.

Our code is stored on my student computer account under *eda132/assignment2*, or more specifically */h/d9/r/mi6034fe-s/eda132/assignment2*. To run the code as a .c file, one can navigate into the proper folder, compile with "make game.c" and run the code with "./game" or alternatively just run the included executable version.

Result

Our efforts resulted in a fairly accurate robot tracker. By manhattan distance we were often a few steps off but generally closer than the noisy sensor, especially as time went on and our algorithm gained more information about the robot's behavior. There are often streaks where it is 100% accurate but then the robot will hit a corner or change direction unexpectedly and we will have to begin searching for it all over again.

If we were to continue with this project there are a couple things I would like to change in order to improve the accuracy of our algorithm. We assumed our scanner reported the noisy location with precise probabilities for simplicity, but if we took the time to make our scanner mathematically sound it would likely improve our manhattan accuracy. On the other hand, something that would improve how often we were 100% correct would be to keep a detailed log of every matrix we generated in order to estimate not only where the robot was at a given moment, but to recreate its entire path.