Assignment 5: Particle Filter Localization
CS 393R: Robotics

Due Date: Thursday, November 3, 2011

Your task: Write the sensor update and re-sampling steps of a particle filter. Demonstrate the accuracy of your particle filter by adding functionality for your robot to walk to a point.

This assignment is to be done **individually**. You will share a robot with your assignment 1-3 partner.

In this assignment, we will be using the humanoid Nao robots and the UTNaoTool (a tool developed by UT Austin Villa to develop and debug code on the Nao robots). You will be using UT Austin Villa's Nao code base and filling in missing parts of the particle filter. Specifically, you will implement the sensor update and re-sampling steps of the particle filter.

Various sections of our Nao code base web page are referenced in this document. Visit the page at http://www.cs.utexas.edu/~pstone/Courses/393Rfall11/resources/nao.html.

We've updated parts of the code base since assignment 4. Follow the instructions in the 'Preparing for Assignment 5' section of our Nao code base web page to set up your copy of the code base for this assignment.

In this assignment, you will be implementing vision updates for distinct observations. Distinct observations are observations where there can only be one landmark that matches the observation. For these observations, you can simply update the particle's probability based on the expected and the observed readings. You should be updating based on both bearing and distance.

You will need to fill in the resampleParticles method and the updateParticleFromObservations method in core/localization/particleFilter/PFLocalization.cpp. Both of these methods are marked by TODOs. There are two other methods marked by TODOs that might be helpful to implement, but they are not used anywhere so only implement them if you want to use them in implementing the two required methods. See the documentation section near the end of this document for additional information that should help you.

Finally, you will need to write some code in core/behavior/BehaviorModule.cpp to get the robot to walk to point 0,0 (the center of the field). Hint: You can get the robot's position (determined by localization) as worldObjects->objects_[robotState->WO_SELF].loc.

We are not providing you with any localization logs, so you should take your own. See the 'How to Take a Log' (updated for localization logs) and 'How to Use a Log in the Localization Assignment' sections of our Nao code base web page for information on taking and using localization logs. Note that logs for this assignment must be taken with the Localization box checked in the log select window of the UTNaoTool, so any logs you have from assignment 4 will not be usable for this assignment. Also, note that your robot will not seem to walk where you think it should in the logs until your localization code is sane.

Checklist: (Scores will be out of 10)

[____] (2 points) Implement particle vision updates from known observations (i.e. the six beacons).

[____] (2 points) Implement and demonstrate particle re-sampling.

[____] (4 points) Demonstrate that your robot can localize to 0,0 (the center of the field). Once reaching the center of the field, it should continue to localize (and hence, walk in place). I will kidnap the robot a few times, and kidnappings may occur at any point (including once the robot is at the center of the field). On each kidnapping, I will place the robot facing any direction I want and on any part of the field that I want (it will be able to see atleast one beacon if it scans). After each kidnapping, the robot should relocalize and walk to the center of the field. I may take a log of your robot attempting to localize if it does not perform well.

[____] (2 points) Clarity and quality of your memo. Email it - along with any code files you changed - to Peter and Katie by class time on November 3.

[____] (1 points) **Extra Credit:** Implement clustering or some other method to get the robot's pose instead of weighted averaging of the particles (in estimateRobotPose()).

Documentation:

- Particles
  - These are defined in core/localization/Particle.*
  - You can get the particle's probability, position, or orientation through: getProbability(), getPosition(), getAngle().
  - You can set the particle's probability, position, or orientation through: setProbability(float), setPosition(Point2D, Rect), setAngle(AngRad).
  - You can also "degrade" the particle's probability using: degradeProbability(float). degradeProbability multiplies the given factor times the current probability to get the new probability. If the current probability is 1, then the new probability will be the probability you passed in to degradeProbability. The reason we're doing it this way is that if you are only resampling every n frames instead of every frame, then we want to keep multiplying out the particles probabilities until it's time to resample again. So the probability from the last step is multiplied by the one for the observations this step, etc.
  - You can determine the expected bearing and distance to an object from a particular particle position using: getDistanceToPoint(Point2D) and getBearingToPoint(Point2D).
  - We have one non-traditional thing in the code: we randomly move each particle based on its probability. This causes the particles to spread out, as the lower probability particles will move out more. You may find you localize better without this - feel free to comment out the randomWalkParticles() call in PFLocalization.cpp's processFrame method.
- World Objects
  - These are defined in core/common/WorldObject.*
  - These objects describe where everything is on the field, as well as tell you if something was seen and where it was seen.
  - World Objects 56-61 are the beacons that the robot might see
  - What the robot sees:
    - If you get the world object, the seen boolean tells if the robot saw that object that frame.
    - visionDistance and visionBearing tell you the actual distance and bearing at which the robot saw the object.
    - The object's loc variable (a Point2D) tells you where the object is located on the field.
- Geometry, Point2D
  - You may want to use some of the existing code in Point2D and Geometry. This is located in core/math/Geometry.*
- Particle Filter Parameters
  - You can set the parameters at the top of the file in core/localization/PFLocalization.cpp.
  - Some of these parameters won't be used in your code.
  - But others you can use to define how often resampling occurs, etc.
- Debug
  - You can add print statements that will show up in the tool's 'Log' window when you change the module to 'Particle filter' by using pfLog. See an example in the updateParticlesFromOdometry method in PFLocalization.cpp.
  - The log window will display all messages from the current frame, and in the bottom left you can set the range of message levels you want to see.