Localization

The most up-to-date localization code can be found on the SVN server in branches/Localization. In this folder are a number of Eclipse projects as follows:

AndroidDataCollector: This is the earliest project and provides an Android application to log sensor data to the SD card. This log data can then be processed using the ParticleFilterDR[…] and associated programs to run the particle filter calculations and visualize the data as well.

AndroidOfflinePF: This project is a small milestone in the port from offline calculations to full online calculations. This is just to compare the processing capabilities of the Android platform. Logging is still done via AndroidDataCollector, then the log file is then read in and processed with this application, and finally that output file is copied to the computer for visualization.

AndroidOnlinePF: This was the previous primary project for the localization. This code provides and Android application that handles online data collection and calculation. Visualization is still done offline on a PC after the output file from the application is copied over.

LogProcessor: These are PC Java programs that are used to process/visualize the PF data.

ServiceTester: This is a simple application designed to implement the LocalizationService and test it in real-time, but with only the service running. Visualization is done offline on a PC.

LocalizationLibrary: This is where all the fundamental localization code is contained. Outside of the PreNav project, all of this code is linked via the Android Library system. That is, LocalizationLibrary is marked as a library and other projects can simply designate it as an associated project and call it as if all the code was in the same project. This is so all ancillary projects can all reference the same, updated code without having to constantly update via copy/paste. A few noteworthy classes in this project:

LocalizationService: This is now the primary interface with which to access the localization system. It is currently designed as a local service and will only work provided it is in the codebase you want to use it in (as it currently is in PreNav). The code to implement this service in a project can be found in the PreNav project as well as in the LocalService example that is on the Android developer website. In summary, the service must first be bound to so that a local copy can be created and the data (particle filter object, compass calibration, and gyro calibration) can be set to be ready when the localization service is actually started. Communication of calculated particle states from the service to the activity occurs via an Android system broadcast and broadcast receiver. NOTE: gyro calibration must be done before binding to the service; compass calibration must occur after the service has been bound to.

In the edu.cmu.ri.rcommerce.filter package is OrientationFilter and the localization algorithms (PedestrianLocalization\_Gyro, etc). OrientationFilter takes in (from the localization algorithm) the Gyro, Accelerometer, and Magnetometer data, filters it (typically lowpass) and combines it to generate a more accurate heading. This heading is then fed back to the localization algorithm (primarily PedestrianLocalization) where it is used to compute whether the user is moving and then calls the broadcast method for the callback in the localization service.

In the edu.cmu.ri.rcommerce.particleFilter package is the ParticleFilter object, however, very little of the actual particle filter calculations happen here. This is more of a stub that outlines the process, but has very little of the actual code. All of this code is help in the Updater (dead reckoning that updates particle positions), Measurer (takes in wifi readings and reweights the particles), and Resampler (redistributes the particle positions based on the weights) objects. There are various versions of these objects, but the ones that are utilized in PreNav are the best.

In the edu.cmu.ri.rcommerce.render package is all the OpenGL rendering code that was used primarily in the HumanRobotInterface project done previously. To my knowledge, none of this is currently used.

In the edu.cmu.ri.rcommerce.sensor package is code that relates to sensor calibration, as well as reading and interpretation. This is where most of the WiFi calculation code is contained.

Additionally, the PreNav project has two activities important to the localization. One is for calibrating the gyro (by averaging the values over 10 seconds to calculate a good steady state) and the other is for calibrating the compass (the offset from the robot map hallway is added to the orientation measured and then compared with the current value for magnetic north).

When using the PreNav application, it is often important to do things in a certain way for optimal localization results.

#Calibrate Gyro must be done first for data handling purposes.

#Calibrate compass works best after rotating the phone about its axis’ a few times to reset the magnetometer. This needs to be done after a phone reboot as the magnetometer tends to get stuck in weird states until this spinning process gets it “unstuck”.

#When walking with the phone, localization works best if the phone is held at about waist level, as parallel to the ground as possible. If you were to define a coordinate system for your body with the origin at your waist, positive X going out in front of you and parallel to the ground, positive Y to your right and parallel to the ground, and positive Z up through your head, the localization works best if held along the X axis in the XY plane with the screen facing up.

#Also when walking, it is important not to try to offset the walking motion you would otherwise impose on the phone. The localization is designed to detect this motion so it knows you are moving. Holding the phone still will cause it to think you have stopped. If you notice the localization lagging behind and not closely observing your motions, this is likely the culprit.