Probabilistic Robotics
BAIPR6, Fall 2009
Lab Assignment : Error Variance

Arnoud Visser

September 14, 2009

Introduction

A robot is a system in interaction with the world. The robot can acquire information about its environment using its sensors. A good example of such sensor is a Global Positioning System (GPS), a sensor which gives an absolute position estimate based the signals of at least 4 GPS-satellites. GPS sensors are used in a wide variety of militairy and civil applications, such as car navigation systems and mobile phones. However, sensors are noisy, and the position estimate of the GPS is no exception.

The goal of this assignment is to estimate the variance of the GPS as modelled in USARsim.

Step 1: Get Your Robot

Each group of students will get its own branch of the UsarCommander to develop. Build UsarCommander, and get a feeling of your robot by tele-operating it. To build UsarCommander, follow the following steps:

1. Claim two `pc-np` c.q. `pc-ow` machines next to each other.
2. Use `TortoiseSvn` to get a copy from the repository svn://u013154.science.uva.nl/Roboresc/2009/group$*$ on one of the two machines.
3. Open the solution `UvA Rescue.sln` and look into the `Solution Explorer`.
4. There are several projects. Set `UsarCommander` as `StartUp Project` by giving it a right click.
5. Build and Run the project by clicking `F5`.

If you are successful, a blue screen with a black frame will appear. This is UsarCommander, an environment to visualize measurements on a map. Initially, the map is empty, indicated in blue. On the right side you can configure a team of robots. We will start with a single robot. Follow the following steps:

1. Add a robot to team by hitting the green '+' button.
2. Configure the robot by hitting the grey wheel next to the yellow arrows.
3. Load a configuration by hitting the directory button at the upper left.
4. Select the configuration `TheStreet.cfg` from the directory `../../../configs`.
5. Modify the location to `-8.59,46.79,-3.7` and the orientation to `0,0,2`.
6. Hit OK
7. Configure the network settings by hitting the blue globe at the bottom.
8. Use the IP-number of your current machine as `Localhost IP`
9. Use the IP-number of the other machine for the three `Servers`
10. Give this configuration a name (top right), and hit OK
11. You are to go, so hit the green `V` at the bottom.
12. Wait a moment before you hit Start.

Now, we should start the `UsarSim server` on the other machine, which will simulate the physics of the world.

1. Go to the directory C:\UT2004\Tools\ImageSrv\Release.
2. Start `ImageSrv.exe`.
3. Specify as UT Map DM-compWorldDay1_250 and press Start.
4. Select `UT2004.exe` instead of `UT2003.exe`.
5. An Unreal window will start up, which will disappear after 20 seconds.
6. Press the button 'Show UT' in the Image Server.
7. You will see a street with two police cars and a fire-brigade truck.
8. Fly with your arrow buttons over the fire-brigade truck. Look at the grey car on the cross-walk. The robot will appear in front of this car.
9. Go back to other machine.

.

Time for a spawn test:

1. Hit the Start button
2. Spawn the robot by hitting the yellow lightning button.
3. Go back to the other machine.

Switch from spectator view to camera view:

1. Fly your spectator view towards the male victim at the end of the police blockade.
2. You can see a small robot, the robot you just spawned.
3. Switch from camera view by hitting the left button.
4. Stop hitting when you see the victim from a short distance, you have found the robot's camera view.
5. Go back to the other machine.

Time for a ride:

1. Activate the Layers 'Free Space', 'Safe Space', 'Victims', 'Obstacles' and 'Agents' (on top of the blue field).
2. Hit the arrow button above the spawn button.
3. You can now control the robot. Drive the robot around the victim.

Look at your robot. This is a Zerg, a small and fast robot[1] designed and build at the University of Freiburg. This robot can be equiped with several sensors, such as a GPS and Inertial Navigation System (INS).

Step 2: Switch to GPS

In the configuration as loaded from `theStreet.cfg`, the position estimate was based on the INS sensor. This sensor measures the accelerations around several axes and integrates this information in velocity and position estimates. It is possible to measure very small accelerations, which allow a very high accurracy position estimates. Unfortunatelly, the precision is not that high. All inertial navigation systems suffer from integration drift: small errors in the measurement of acceleration and angular velocity are integrated into progressively larger errors in velocity, which is compounded into still greater errors in position.

There natural counterpart are GPS sensors, which have low accurracy but high precision. Look what happens if you use GPS for the position estimate.

1. Hide the teleoperation control buttons by clicking the red '$\emptyset$' button.
2. Stop the current run by clicking 'Done' twice.
3. Click 'Reset'.
4. Configure the robot by hitting the grey wheel next to the yellow arrows.
5. Switch the selection of the 'Pose Sensor' from INS to GPS.
6. Close the configuration screen and spawn the robot.

As you can see, the Zerg seems to be jumping around, indicating that the GPS makes large errors in its position estimate.

Your job is to drive a straight line and to measure the position estimate perpendicular to that line for both the INS and GPS. Plot the error against the GroundTruth. When you save you data as a comma seperated file, you can easily load it into for instance MatLab for further processing.

[1] http://www.informatik.uni-freiburg.de/~kleiner/video/fastZerg.avi

Can the error be modeled with a Gaussian? What are the parameters of this Gaussian? What is the probability that the INS reports a value 10 centimeter from the GroundTruth after 3 minutes?

Step 3: Correct the measurements

An autonomous mobile robot most determine its actions through the measured signals. The raw sensor values could be tied directly to robot behavior. Yet, much can be gained when some intermediate steps are introduced between the behavior and the raw measurements. This can lead to a complete pipeline of perceptual interpretation. It is now your task to create some initial filters.

Correct the lateral error by averaging the GPS-values perpendicular to the driving direction. How long do you have to average for a reasonable estimate?

Correct the longitudinal error by averaging the GPS-values along to the driving direction. Use the drive commands to estimate the baseline for the average. How long do you have to average for a reasonable estimate?

Hand-In

When you have completed the assignment, commit your modifications to code and mail me (a.visser@uva.nl) one file: labbook_errorvariation.pdf.

The body of the labbook should be to the point. Do not elaborate on the code, but show some highlights where you are proud of. Concentrate on the essence of your approach.

In case of doubt, summerize your quest for the first steps, and elaborate on the last step.

Do not attempt to perform an experiment that covers all possibilities, but make clear what the limits of your experiment are and how those limits are encountered.

Acknowledgements

The model of the GPS-sensor is recently included in USARSim by Ben Balaguer and the drifting of the INS sensor is modelled by Stefano Carpin (both University of California, Merced). Usar-Commander is designed and build by Bayu Slamet. Several other students of the Universiteit van Amsterdam have added components into his framework. Julian van de Hoog (University of Oxford) made the most recent modifications and heavilly tested the environment during the RoboCup.

Hints *(Please read this!)*

UsarCommander is programmed in Visual Basic. This has some downsides, but concentrate on the benefits:

- Visual Design

  In Visual Studio, it is easy to add buttons, textboxes, layers and other things to the User Interface, and link it with your code. Easy visualization of processed information can help experiments tremendously.

- Auto completion and instantaneously code checking

  When you type in your code, Visual Studio can show you from each object the applicable methods. You can jump from one object to another by 'Go To Definition' and 'Find All References'. When you make a mistake (for instance a missing bracket), directly an error report will appear in a lower window. Learn to make use of these advanced options.

- Debugging Facilities

  In debugging mode, you can inspect the value of all active variables and objects (also complex ones that contain a lot of data). Learn to make use of these advanced facilities.

- Extensive Help

  Microsoft ships its Studio with extensive help, interactive reference manuals, developer networks, etc, etc. Many professional developers use Studio, so on many independent forums additional information is available. Make use of this information, but be fair to give the proper credits and references.

And, if this is not enough, you can also ask for help.

Have fun!