Introduction to Mobile Robots 2017

Homework #3

Kalman Filtering and SLAM

Note to myself - This file has conditional text formatting to hide the answers.

Generally, I'd prefer if you submit all homeworks entirely in electronic form. PDF files are best. See the web page for how to get PDF tools from Adobe for free.

You should do this and all homeworks alone without any undue help from your fellow students.

Grades in this course are based mostly on homeworks. Therefore, don't make the mistake of ignoring any questions asked. Answer each one explicitly. The docs asked for that are "not exceeding" a certain size do not need to be that size. A lot less is fine. Just tell me what you learned, and help me understand your code so I can grade it.

This part examines the formulation and performance of the Kalman filtering approach to the robot localization problem. You will implement the simple 2D Kalman filter given in the notes. Submit your results for this question in files named KalmanFilterYourAndrewId.java etc., and ExternalDocsYourAndrewId.pdf (or doc or whatever). You can also submit paper for the external documentation. Make sure to read all the supplied material on the SMRDE environment, Eclipse, and Java and get it all working before continuing.

1. Introduction and Software Design

The software implements a fairly sophisticated simulation of a mobile robot whose physics are close to reality and whose sensor nonidealities precisely match the requirements for a general Kalman filter as well as the noise specifications you provide in your filter. In other words, whatever noise values you pick in your filter, the simulator will corrupt your sensors by exactly that amount of unbiased Gaussian noise.

1.1 Ground Truth and Simulation

The robot is a MarsRoverRobotSprite. It has its body frame x axis pointing forward. The sensor has its y axis pointing forward. Both are consistent with the simple 2D filter in the class notes. This robot has a gyro, and speed sensor and a ladar (range) sensor. The ladar can be configured in field of view, maximum range, number of pixels, and rotation rate.

The simulator is completely set up to use. The robot is placed on a path in an environment with square landmarks of known dimensions which are visible to it. A pure pursuit controller moves the robot in the left window to follow the path as precisely as it can given the wheel to ground interactions and the dynamics of the body and actuators. This design is intended to make sure the simulated robot does not jump around in a way that would invalidate your Kalman filter system model.

The range sensor on the robot measures, for a number of rays, the range to the first reflecting surface in the environment. Pixels are regularly spaced across the field of view so the pixel index also provides a measurement of bearing (Figure 1). This sensor is used at times as a “bearing only” sensor by ignoring the range values it produces.

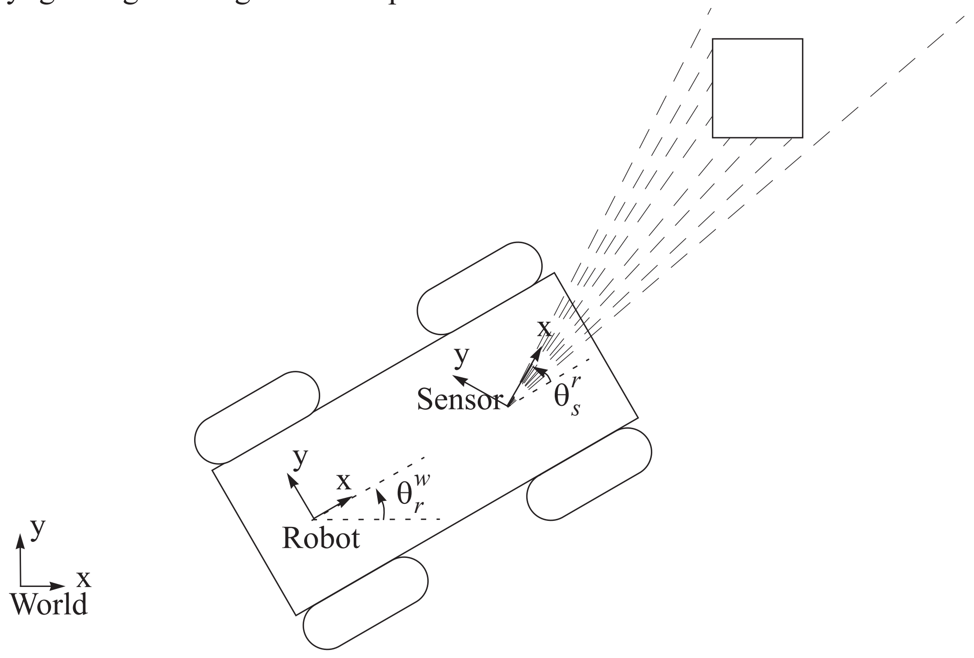

Figure 1: Coordinate Frames and Range Sensor. The range sensor casts a number of rays to determine the range to the first reflecting surface in the environment. Note that the ends of the pixels may not land exactly on the corners of the square landmarks.

Landmarks are simply small squares that reflect ladar beams. Of course, the center of the landmark is not the position of the corner that the ladar sees and there are four corners - each in a different location.

Landmarks are kept so small that you will not need to process the range imagery to fit lines to find the corners. Any range reading at all can be assumed to be pretty close to the corner which is facing the robot. Also, the number of landmarks is fixed at initialization time so that it is not necessary to automatically detect their existence.

The graphics have been set up to draw both the ground truth in one DrawingArea and your estimates of what is going on in a second DrawingArea. Things have been set up so you should not have to make any significant changes to the logical flow of any of the code. Take care to ensure that you are not cheating by accessing the ground truth data in the robot RigidBodySprite. In some cases, this could even make things worse.

1.2 First Run

The file applicationHW0KalmanFilter.java is provided in source form. It contains the main driver routines with all of the parameters, world creation, graphics etc.. It configures and then drives the simulator in the Simple Mobile Robot Development Environment (“smrde”) to produce fake sensor readings for a robot moving around in a room with walls that reflect laser rangefinder emissions.

Near the top of this file, find the following variables and make sure they are set like so:

```
boolean do2DDeadReckoning = true;
boolean doKalmanFilter = false;
boolean doLandmarks = false;
boolean doSLAM = false;
boolean doFixTwoLandmarks = true;
```

The variable `doSimplePath` can be turned on later if you want to use a simple square path to help in your debugging. The variable `doDrawGroundTruthRobot` can be turned on the see the pose of the ground truth robot in the right `DrawingArea`.

Once you have Eclipse and SMRDE installed, and the demos are running, you should also be able to run the Kalman filter application. From the top level menu, choose Homeworks and then the Kalman filter option to run the code. Press Go and you should see a robot moving in both windows.

Near the top of the `applicationHW0KalmanFilter.java` file, you will also find:

```
final double sigmaGG = 0.001*0.001;
final double alphaV  = 0.01;
final double sigmaRR = 0.01*0.01;
```

```
UncertaintyContext uncContext = new UncertaintyContext(sigmaGG,alphaV,sigmaRR);
```

These numbers control the variances of the gyro, the speed encoder, and range sensor respectively. If you increase the noise amplitudes, the data gets noisier.

The gyro is created so that its integral (the heading) will be a random walk error process. For that to happen, the effective variance of the gyro readings must be a constant and we would like that effective constant to be independent of how often the gyro is read. Therefore, the simulator corrupts the gyro measurements by a random amount whose variance is:

$$\Delta\sigma_{\omega\omega} = \sigma_{gg}\Delta t \tag{1}$$

where Δt is the time that has elapsed since the last time the gyro was read and σ_{gg} represents the effective gyro variance at 1Hz sampling and $\Delta\sigma_{\omega\omega}$ represents the noise amplitude of a single reading. Hence, regardless of how frequently you read the gyro, the variance of the sum of all the errors will be proportional to the time elapsed.

Similarly, the speed encoder has been created so that its integral (the distance travelled) will be a distance dependent random walk process. The simulator corrupts the speed measurements by a random amount whose variance is:

$$\Delta\sigma_{VV} = \alpha|V|\Delta t \tag{2}$$

where Δt is the time that has elapsed since the last time the encoder was read and $\alpha|V|$ represents the effective speed encoder variance at 1 Hz sampling. Therefore, α represents the effective speed encoder variance at 1 m/sec speed. Hence, regardless of how frequently you read the encoder, the variance of the sum of all the errors will be proportional to the product of the speed and the time elapsed - that is, the distance travelled.

The laser rangefinder has a constant error variance (independent of range, time, or distance) in its range measurements, the angles of the laser beams are known perfectly (no noise) but that does not mean the angles to the landmarks corner will be known perfectly. See Figure 1 and notice the precise positions of the endpoints of the laser beams.

The left robot is the simulated ground truth world that is used to ensure that robot stays on the path. Because the right robot gets noisy sensors, it will not stay on the path. Temporarily set:

```
final double sigmaGG = 0.1*0.1;
```

and run the program for one cycle of the path just to see how bad it gets if your gyro is not a good one. You should now see two robots in the right window - they were on top of each other before. The estimated robot is the one with the little square for the sensor in front of it. By the way, notice what happens to the error when the robot returns to the start point.

Now, **change it back** to its original value.

This is a good time to learn about <ctrl-Z> and <ctrl-Y>. The former undoes your edits one at a time, so you don't have to remember what the code started like. The latter lets you go forward through remembered changes in case you went back too far.

1.3 Code Overview

Javadocs are provided for the entire system. It is almost impossible to proceed without figuring out how to access the javadocs. The simplest way is to read them with an HTML browser. The best way is to configure Eclipse so that it integrates them with the source code editor as outlined in the "How to get started in Java" document on the web. Doing this will save you hours and allow you to see the javadocs on anything including the run-time library by hovering the mouse over the variable of interest. Understanding all of the SMRDE environment will take more time than you have, so don't spend too much time with that. Read what you need when you need it.

Figure 2 shows the main files of interest for this assignment.

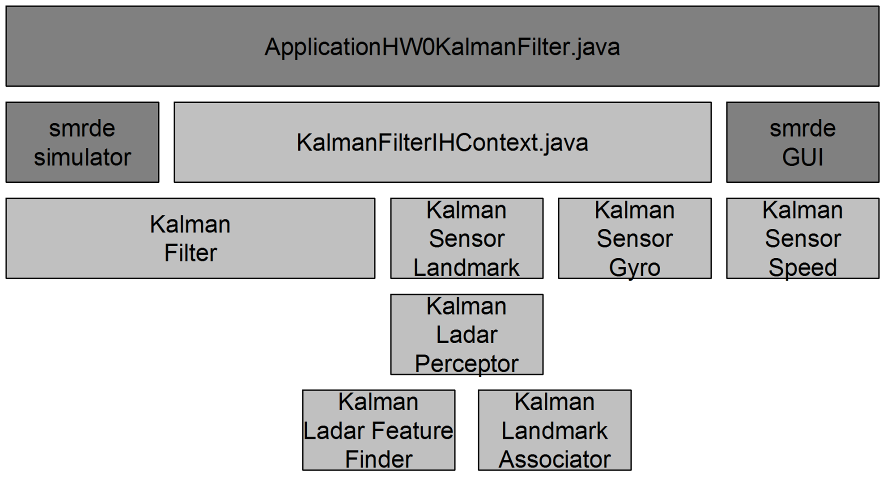

Figure 2: Rough Invocation Hierarchy for the Software for this Assignment.

- `applications.homeworks.ApplicationHW0KalmanFilter` - the main program tied to the KalmanFilter application button. Most of the code that is related to setting up the graphics and simulation is in here. You should need to do nothing in here except tweak some numbers and flags.

- `applications.homeworks.unlocked.hw0`....
  - `KalmanFilterIHContext.java`. A layer between the generic Kalman filter components and the GUI. This code knows what the sensors are and tells the GUI about the internals of the filter. You will want to adjust the flags at the top of this file from time to time.
  - `KalmanFilterIH.java`. The top level Kalman Filter function. Knows the structure of the state vector. Literally implements the classical system model and measurement processing equations of the Kalman filter. Owner of the state vector and its covariance.
  - `KalmanSensorSpeed.java`. The interface between a speed sensor and a Kalman filter. Provided so that the filter code is independent of the sensors used. Computes the innovation, the measurement Jacobian and the measurement covariance.
  - `KalmanSensorGyro.java`. The interface between a gyro and a Kalman filter. Provided so that the filter code is independent of the sensors used. Computes the innovation, the measurement Jacobian and the measurement covariance.
  - `KalmanSensorLandmark.java`. The interface between a landmark sensor and a Kalman filter. Provided so that the filter code is independent of the sensors used. Computes the innovation, the measurement Jacobian and the measurement covariance. This sensor is much more complicated than the other two, but the complexities are placed in classes external to this one as outlined below.
  - `KalmanLadarPerceptor.java`. The top level perception class that finds features, associates them with a map, and generates the derived info necessary for use by a Kalman filter.
  - `KalmanLadarFeatureFinder.java`. A class that processes ladar scans to look for either corners or reflections. The former makes sense for large landmarks (with enough range hits on the sides to make corner detection possible. The latter makes sense for small landmarks where any hit at all can be regarded as a reflection from a point landmark. We will be using reflections.
  - `KalmanLandmarkAssociator.java`. A class to determine the index of the closest map landmark to a feature detected in a range image.

Also read the Javadocs for `smrde.Math` and learn about `Matrix` - especially the methods `setElement()`, `getElement()`, `setAll()`, `setDiagonal()`, `print()`, `multiply()`, `multiply3()`. This class can invert a matrix of arbitrary size and you need that for the Kalman gain calculation.

In `smrde.Math` read about every kind of Pose, Vector, and everything with “Polar” in its name.

The class `Pose2D` in `smrde.math` is a layer over `AffineTransform` that is also very useful. It is possibly more general than a similar one you may have done ourself in an earlier assignment, and it is used throughout the simulator, so I recommend you use it instead of yours. When you give it a value, the underlying `AffineTransform` (which is a homogeneous transform) is updated automatically.

Read up on `smrde.objects.RangeImage`. and understand the methods provided for reading data out of a range image in either IMAGE, POLAR, or SENSOR coordinates.

In java’s `Math` package, look at `sqrt()`, `abs()`, `hypot()`. The most straightforward way to print information is with something like:

```
System.out.printf("%f\n",mahalanobisDistance);
```

These days you can Google “java X” and get the Html javadocs for X off the Sun Microsystems servers with little effort.

2. Basic Dead Reckoning Filter[22 Points]

Here, you process the unfiltered odometry sensor data to see your robot get progressively lost. The state vector is:

$$\underline{x} = \begin{bmatrix} x & y & \theta & V & \omega \end{bmatrix}^T \qquad (3)$$

where V is the vehicle speed (assumed to be directed along the body x axis) and ω is the angular velocity (assumed to be directed along the body z axis).

The file ApplicationHW0KalmanFilter.java has the infrastructure for an integrated heading Kalman filter all set up.

Now, set up the code like so:

```
boolean do2DDeadReckoning = false;

boolean doKalmanFilter = true;
```

i) **[10 Points] Kalman Filter Code**.

Here you will code the Kalman filter equations. In the file `KalmanFilterIH.java`, fill in the following functions.

```
        public void setInitialState();
        public void setInitialCovariance();
        public void kalmanSystemModelSlow();
        public void processMeasurement();
```

When in doubt, look for

```
//TODO: fill me in
```

Also, choose values for the constants `P0xx`,`P0yy`,`P0tt`,`P0vv`,`and` `P0bb` as well as for `Qxx`, `Qyy`, `Qtt`, `Qvv`, and `Qbb` and use them in the initialization functions. Read every single thing in the textbook and the slides before sending me an email to ask me how to do this. This is not easy to get right but you can get it almost right by playing around. It is right when the ground truth robot remains within the uncertainty ellipse drawn at the estimated pose around 99% of the time when landmarks are not being used.

Be very careful here, and when processing landmarks later, to normalize any angles (with Angle.normalize) after performing arithmetic on them. Arithmetic occurs in updating the orientation state and in forming bearing residuals.

ii) **[5 Points] Measurements Code.**

Turn on `doKalmanFilter` in `ApplicationHW0KalmanFilter.java` and make sure `doLandmarks` is false and `doSLAM` is false. In the files `KalmanSensorSpeed.java` and `KalmanSensorGyro.java`, fill in the three skeleton functions:

```
        public Matrix getInnovation();
        public Matrix geMeasurementJacobian();
        public Matrix geMeasurementCovariance();
```

in order to make the Kalman filter mimic the behavior of the odometry system.

I will test this function by calling it from my own test code so do not change any function names. Your solution must be generated using the principles of Kalman filtering. That is, don't copy the simple dead reckoning code that is deactivated by the `doKalmanFilter` switch in `ApplicationHW0KalmanFilter.java`.

iii) **[2 points]** Write external documentation not exceeding one single-column typed pages in 12 point font describing your solution. Don't parrot what is in the notes or the text. Tell me what I did not already tell you about how to do this.

iv) **[2 points]** Explain what the quantities `P0xx`, `P0yy`, `P0tt`, `P0vv`, and `P0bb` mean and describe a good strategy for choosing their values. Explain what the quantities `Qxx`, `Qyy`, `Qtt`, `Qvv`, and `Qbb` mean.

v) **[1 points]** Plot on a graph the innovation Mahalanobis distance (a scalar) for each measurement (one graph for gyro and one graph for speed) processed versus time (cycle number) while the robot travels a little more than one circuit of the path to show that the filter is working. Explain how you computed it. Use any graphing program you like.

vi) **[1 points]** Why do we look at the Mahalanobis distance and what does it mean?

vii) **[1 point]** What is likely to happen if a measurement is processed whose Mahalanobis distance is very large due to spikes or other outliers that are present in the sensor data?

3. Landmark Aiding [42 points]

So far, your Kalman filter only processes velocity and heading measurements so it is just doing dead reckoning - and getting increasingly more lost as it moves. Now you will incorporate measurements of features for which you have a map.

Landmark states must go in the state vector right after the angular velocity in the order landmark1X, landmark1Y, landmark2X etc.

At the top of `ApplicationHw0KalmanFilter.java`, you will find:

```
boolean doBearing = true;
boolean doRange   = false;
```

When only `doBearing` is set, it will be called "bearing only mode". Conversely, when only `doRange` is set, it will be called "range only mode".

i) [15 points] Landmark Measurements

Turn on `doLandmarks` in `ApplicationHW0KalmanFilter.java`.

```
boolean doLandmarks = true;
```

In the file `KalmanSensorLandmark.java` fill in the three skeleton functions:

```
public Matrix getInnovation();
public Matrix geMeasurementJacobian();
public Matrix geMeasurementCovariance();
```

This sensor has some graphics built-in to help you debug. It draws little circles on the display at the position of the detected feature and the predicted feature.

Form innovations in the sensor frame in polar coordinates. There are other possibilities but this is what I did and it makes it easier to check if we all do the same thing.

Notice the following code in the constructor...

```
if(doBearing && doRange){
        numMeasurements = 2;
        bearingIndex = 0;
        rangeIndex = 1;
}
else if(doBearing){
        numMeasurements = 1;
        bearingIndex = 0;
}
else if(doRange){
        numMeasurements = 1;
        rangeIndex = 0;
}
```

For now, leave both `doBearing` and `doRange` turned on and form a 2X1 measurement vector etc. Later, you can experiment with using just one or the other but remember to honor how the above code changes the position of the measurements if necessary.

The function `getInnovation()` calls `kalmanLadarPerceptor.findFeatures()` to find reflections in the range image. The `kalmanLadarPerceptor` class is provided in source form. Note that it computes numerous transformations of the features such as `predictedFeaturePoseInBody` etc. You will want some of these for the innovation calculation and the measurement Jacobian.

In the file `KalmanLadarFeatureFinder.java`, fill in the function

```
public PolarPose2D findReflectionFeaturesInLadar();
```

You can ignore the size of the landmarks and simply return the coordinates of the shortest laser beam that intersected something. Be sure to account for the noise in the range data when deciding on whether a beam hit something. The sensor nominally returns `maxRange` for a beam that hits nothing. There is nothing else in the world that reflects ladar besides landmarks. Return the result in the sensor frame in polar coordinates in a `PolarPose2D` object. The orientation (not the bearing!) of the feature can be set to zero. It is used for corner features which do have an orientation. We don't do corner features in this assignment.

Finally, in the file `KalmanLandmarkAssociator.java`, fill in the function:

```
public Pose2D closestCornerPose();
```

It produces the pose of the landmark (more generally, the corner on the landmark) that is closest to the sensor. You can get the rectangle of the landmark and its position in the world with code like:

```
RigidBodySprite landmark = landmarks.get(i)
```

```
Rectangle2D rect = (Rectangle2D) landmark.shapeContext().shape;
Pose2D landPose = landmark.poseContext().poseRelative;
```

I will test these functions by calling them from my own test code, so do not change their names.

ii) **[5 points]** Write external documentation not exceeding two single-column typed pages in 12 point font describing your solution. Don't parrot what is in the notes or the text. Tell me what I did not already tell you about how to do this.

iii) [**6 points]** Describe particularly how you arrived at the innovation, its Jacobian, and the numbers in the R matrix for the features.

iv) [**2 points]** Did you put anything in Q for the landmarks? Why or why not.

v) **[3 points]** In bearing only mode, plot on a graph the innovation Mahalanobis distance (a scalar) for each measurement processed versus time (cycle number) while the robot travels a little more than one circuit of the path to show that the filter is working.

vi) **[3 points]** How do the landmark innovation magnitudes relate to the time period between landmark observations? Imagine what happens if landmarks are rarely seen.

vii) **[3 points]** How does the innovation covariance $HPH^T + R$ account for this effect?

viii) **[3 points]** Suppose the perception system used the range sensor to detect corners in the landmarks. Why would it be necessary produce the closest corner to the sensor rather than the closest corner to the detected feature location (based on the present state estimate)? Does the closest corner to sensor scheme still have a plausible failure mode and if so what is it?

ix) **[2 points]** Run your code in bearing only mode and in range only mode (change `doBearing` and `doRange` in the main ApplicationHW0... file) with `doFeatureGraphics` turned on (the little circles that show up on the display) in `KalmanSensorLandmark.java` and comment on the nature (direction relative to laser beam) of the errors on screen in each case. The errors are exaggerated and the pattern may be hard to see but think about what should happen if one cycle of the filter equations acts to fix the residual by changing the pose and then another cycle views a landmark again.

4. SLAM [35 points]

So far, your Kalman filter assumes somebody has been in the area before who spent a lot of time making a perfect map. In exploration contexts (like planetary rovers), that would be impossible. We will deal with a slightly simpler case than no map - one where the landmarks are known but not accurately positioned.

In performing the following work, you may find it valuable to turn range and bearing observations on and off in various combinations during the debugging phase. A working filter will work in any of the three possible cases.

i) **[15 points] SLAM Measurements**

Make sure `doSLAM` is turned on in `ApplicationHW0KalmanFilter.java`.

```
boolean doSLAM = true;
```

When this is turned on the main program corrupts the assumed initial landmark positions by a random amount and does not tell the filter it did so. Hence the filter must find the corrupted positions of the landmarks. Notice that the constructor for `KalmanFilterIH` lengthens the state vector when `doSLAM` is on to include two states (x,y) for each landmark. Once again in the file `KalmanFilterIH.java`, update the following functions to convert your basic landmark filter into one that will also fine tune the landmark positions. Make your changes in such a way that it also still works with `doSLAM` turned off (as it presently does).

```
        public void setInitialState();
        public void setInitialCovariance();
        public void kalmanSystemModel();
```

You will not need to use the speed enhancements in the system model algorithm that were discussed in class. There are few enough landmarks that the code is fast enough without it. You should not need to update `processMeasurement()`. If you do, you can probably make it more generic without having to look at the value of the switch `doSLAM`.

In the file `KalmanSensorLandmark.java`, update the function:

```
        public Matrix getMeasurementJacobian();
```

As a hint, note that only one landmark is associated with the innovation at any point in time.

ii) **[6 points]** Write external documentation not exceeding two single-column typed pages in 12 point font describing your solution. Don't parrot what is in the notes or the text. Tell me what I did not already tell you about how to do this.

iii) [**7 points]** Describe particularly how you arrived at the extra elements of the Jacobian, and how it was computed.

iv) **[3 points]** Configure the filter to process both bearing and range measurements.

Make sure `printLandmarkErrors` is turned on in `ApplicationHW0KalmanFilter.java`. Using its output, plot on a graph the average landmark error until it stabilizes. Provide the graph. Interpret what you see.

v) **[4 points]** How would you change the code, in principle, to account for the fact that the range to landmarks, and hence the 2D position, is not known to `KalmanLandmarkAssociator` in a bearing only system. Hint: This means more than one landmark may be close to the measured bearing. Make sure your ideas are robust.

5. SLAM Bonus [21 points]

Some harder questions for bonus points. You may not be able to see these effects if your code is not exactly right or if you can't get it to run adequately with poor odometry sensors. If you get only this far and no further you have done well.

i) **[4 points]** Notice how sometimes when one landmark moves, others move which are not even in view of the sensor. What is going on here? You may see more of this happening while working on the questions below. Hint: What are the off diagonal entries in the P matrix for?

ii) **[5 points]** Why do you suppose I set two landmarks to have zero uncertainty (look for "lN2 in ApplicationHW0KalmanFilter.java) ? Hint: Turn off `doFixTwoLandmarks`. Then, make sure you are running in bearing only mode and change the call to the kalman filter to this (change it back after you are done):

```
        kalmanFilter.estimateFromRangeImage(robot.rangeSensor.rangeImage,
0.9*speed, omega, dt);
```

iii) **[4 points]** Temporarily turn on `doRange` and comment on any differences in behavior now and on the relative merits of measuring range versus angles. In what cases may range measurements be particularly useful?

iv) **[4 points]** Turn off `doFixTwoLandmarks`. Temporarily lie about the initial robot heading in KalmanFilterIH.java with this:

```
        state.setElement(TT,0,initialPose.getTh()+0.1);
        private final double P0tt = 0.0001;
```

Comment on the capacity of SLAM to determine initial state error. Change the code back immediately.

vi) **[4 points]** Turn off `doFixTwoLandmarks`. Change the path to a square with:

```
        boolean doSimplePath = true;
```

Change the covariance drawing resolution with this.

```
        final double landmarkCovMin = 0.0;
        final double landmarkCovScale = 10.0;
        final double robotCovMin = 0.0;
        final double robotCovScale = 10.0;
```

Now run the program and interpret what you see in terms of correlation of robot state estimates and landmark state estimates in SLAM.

6. Frequently Asked Questions

6.1 Landmark Uncertainty

Q: Should we assume the same uncertainty for both the x and y values of each landmark?

A: Yes, at least initially. The filter will change them as it runs.

6.2 Linear Filter

Q: Regarding the Kalman filter and processMeasurement(), are these the correct equations?

Measurement: z_k = H_k * x_k + R_k

A: No. For two reasons. First R_k is not added in a real Kalman filter. It is simply notation to denote that the measurements are corrupted by noise. Second, you need an Extended Kalman filter for which the measurement model is z = h(x).

6.3 Innovation for State Update

Q: Is this right?

Updated state estimate: x_k_updated = x_k_old + K_k * [z_k - H_k * x_k_old]

A: No. You need the extended Kalman filter innovation which is dz = z - h(x).

6.4 Use Measurement Jacobian in Innovation?

Q: Regarding the speed (and gyro) sensors, is this right?

    innovation = z - h(x) = z - measurementJacobian*state

    where measurementJacobian = [0 0 0 0 1 0 ... 0];

A: This will work but it is notationally dangerous. Only in the case of gyro and speed is h(x) = measurementJacobian*state. For landmarks, the innovation is much much more complicated to compute because that measurement is not linear.

6.4 R Matrix

Q: Why are you passing dt into this computation.

A: It is better in general to have some elements of R be proportional to dt. This tends to make your filter behave appropriately if you change the time step.

6.5 Innovation Again

Q: I'm a bit confused on how to evaluate h(x). Can you please elaborate?

A: The kalmanLadarPerceptor class is provided in source form. Note that it computes numerous transformations of the features such as predictedFeaturePoseInBody etc. You will want some of these for the innovation calculation and the measurement Jacobian.

More generally, h(x) is calculated by assuming x is right and predicting the measurement you would get for that value of x.

6.6 Error Ellipse

Q: The variance ellipsoid shrinks somewhat when the robot slows down to make the turns, is this normal? I thought that the ellipsoid should grow linearly with time.

A: The trace should grow with time. It may rotate and it may get thinner in one direction or another in seemingly random ways.

6.7 What does Q do?

Q: How quickly the robot gets off course seems to depend hugely on Q ...?

A: The Q matrix is the "modelled" rate of growth of P. This is not to be confused with the actual error in robot position. You are controlling the size of the ellipse here (with Q) and it has no connection to how lost the robot actually is - unless you make such a connection by choosing a good value for Q.

Your goal in the first part is to make the model reflect how lost the robot actually gets, not to make it "unlost". Also the match between actual error and modelled error must be interpreted in the context of a large number of experiments (i.e. random process) rather than one run.

6.8 Validation Gates

Q: Are we supposed to use validation gates for measurements? I got an impression we'd use them only for deciding which landmark we're looking at, while the filter would disregard outlying measurements automatically.

A: There is never a requirement to use a validation gate in theory. In practice, measurements (which here include all of gyro, encoder, and landmark observations) may have outliers or your models may be wrong, and its a good practice. I use em for everything just in case and since it less work than using them sometimes.

6.8 Measurement Model Again

Q: How do I code the equation z = h(x) + noise, as z = h(x) + R?

A: The equation z = h(x) + noise is not z = h(x) + R (z is a vector, R is a matrix). In any case, this equation is conceptual and not one of the equations in a Kalman filter. The equation you do code is

dz = z - h(x) . The slides clearly state “these are the equations you will use ....”.

6.9 No System Model Noise

Q: I'm a bit confused as to why the system does not progress if the Q matrix is set to 0. If the Q matrix is zero, it should only affect the projection of state covariance in the system model such that P_k+1 = phi_F*P_k*phi_F_transpose. That being said, it's almost as if having Q set to 0 is somehow zeroing out the P matrix.

A: The equation P_k+1 = phi_F*P_k*phi_F_transpose typically does not increase the magnitude of P much. If Q is zero, P may not get much bigger than P0.

6.10 Bearing Uncertainty

Q: In the textbook, it says that the R matrix for landmarks should have a sigma_aa term, I have no idea what this should be so I'm setting it to ...

A: The R relates to the measurements produced by the landmark observations. Those are ranges and bearings and both are somewhat uncertain. Although the range sensor has no angle error, the perception process which determines the bearing to a landmark can only return exactly the angle of a laser beam. Since the world is a continuum, the angle must have some error in it most of the time. You are supposed to guess some reasonable value for it based on the space between range pixels.

6.11 What is the Measurement

Q: In the landmark getInnovation(...) function, I'm assuming that using measured and predicted points in polar coordinates is the right thing to do but it seems strange that the measurements are in polar form while the state is in x,y form. I'm assuming that the jacobian (H) does the work of changing dz into the right state variables since the dimensions seem to match up.

A: This is all as it is supposed to be. The dimensions, units, and semantics of z need not be very closely related to those of x. The measurement model z = h(x) explicitly models how x can be used to produce whatever z is supposed to mean.

6.12 Noise in Range Threshold

Q: For the findReflectionFeaturesInLadar(...) function, you say to use sigmaRR in the calculation to find the landmark, but I don't understand how this is relevant. As I understand it sigmaRR is the variance in the range sensor but what we're looking for is the shortest range measurement, what does this have to do with the variance?

A: You do not need to use the range variance but ask yourself how you could implement a more robust threshold if sigmaRR happened to be a large fraction of the range measurement. As with almost all questions in all homeworks, if you do the work to try it, you will see what the issue is.

6.13 Corners?

Q: What is "a corner"? Is it any landmark or only landmarks 0, 2, 6 and 8? I assume it is the former but it is quite unclear to me.

A: The word is used in two ways. The landmarks are little squares which have corners. Also the "room" has corners. A "corner landmark" is one of four landmarks in the room. A "landmark corner" is one of the four tiny corners that you can detect if the ladar data is dense enough. The KalmanLadarFeatureFinder has an option to detect the corners of large landmarks. I decided to take this out of the assignment to make it easier but left the code for future use. The writeup discusses this.

6.14 Landmark Corner?

Q: Please elaborate on the difference between landmark model frames and corners.

A: The basic issue I am hinting at is the difference between the landmark state (which is the center of the landmark or the landmark model pose in the notes) and the thing that the ladar is actually detecting. It can't see the center of the landmark. I left this subtle issue in the homework because it does make a big difference in practice. A real system will not work at all unless this is done right.

6.15 Landmark Uncertainty

Q: My filter breaks completely when I tell it the bearing to landmarks has zero uncertainty.

A: The range and bearing measurements are first of all not measuring the center of the landmark and second of all not even measuring the corner that is closest to the range pixel you chose to represent the corner.

Imagine zooming in on a small square where the ends of the ladar pixels hit the square. The positions of the ends of the ladar pixels are arranged randomly around the perimeter and the likelihood that on actually hits the corner is very low. In reality, the bearing you compute can be off by at least half the angular width of a pixel.

Similiarly, the range measurement you get is first of all noisy and second of all not exactly at the corner.

The moral is.... R = diag(0,sigmaRR) is an underestimate of the true uncertainty in the measurement in both dimensions. In many cases it is better to overestimate uncertainty because you will get so many measurements that the right answer will be computed anyway. You can go too far but it is a good rule of thumb.

6.16 More On What is the Measurement

Q: So in the code, we are expected to use z as z_sen - the polar measurements from the sensor, and produce the expected landmark reading from the current state to take their difference. Thus the difference is returned as the two-element innovation vector. However, this is arbitrary, right? Couldn't we pretend that the sensor returns us a position 3-tuple (x, y, theta) by converting its reading into body position inside the getInnovation() function and returning the difference with the current state, as I described ..... (provided Jacobians and measurement uncertainties are fixed appropriately)?

A: Yes it is arbitrary "provided the Jacobians and measurement uncertainties are fixed ...” To produce a synthetic pose measurement from z, you need to invert h(x) to get x = hInverse(z). Sometimes this is easy but not in our case. Also, as in our case, z has 2 dof and x has three so there is not enough info to invert it. The KF does a nice job of figuring out all that for you.

For the uncertainty R. If you managed to invert h, you then need to compute something like H^TRH to get the new R matrix which still requires the nasty H matrix. Often its much easier to figure out R if z is the real measurement.

In fact, I have done exactly what you propose for this same problem in the past. If you are willing to cut a few corners it can work.

6.17 Mahalanobis Distance

Q: What should the MD graphs look like?

A: Generally MD should NOT be << 1 or >> 10 all the time in a well tuned system. If it is, something is not tuned right. It may not be obvious what the wrong thing is because its a feedback system, so its like tuning a controller. However, many imperfectly tuned systems work fine as long as the absolute error magnitudes never exceed the point where the linearization assumptions break or a misassociation occurs.

I am not looking for perfect tuning. The main reason to ask for these graphs is what you learn about tuning by making the graphs and thinking about what they mean. In many cases, people discover many measurements are being rejected when they plot these graphs and then they go and fix something.

6.18 Q Magnitudes

Q: How can I look at the real measurements to determine the right values for Q?

A: Probably the best way to tune Q is to run the vehicle in dead reckoning mode about 100 times and determine the P matrix after, say 10 m of travel. Once you have that you then tweak the values in Q until they produce a P of the right magnitude. Pxx depends on both Qxx and Qvv over more than a few cycles so its not so obvious how to do that. I wrote a paper on this called "Fast easy Systematic and Stochastic Odometry Calibration". Having hopefully convinced you there is a good way to calibrate a stochastic system, I would also recommend you do something that looks good visually instead to save a lot of effort.

The robot error ellipse is not supposed to represent the magnitude of the error at each instant of time. Its the error distribution of a zillion such robots who are all conceptually there at the same time and each had different sensor errors consistent with the values in R. The moral is the ellipse should be bigger than the robot error almost (99%) all the time.