CSCI–4190 Introduction to Robotic Algorithms, Spring 2006
Assignment 2: out February 13, due February 23 and March 2

Localization and the extended Kalman filter

In this assignment, you will write a program that simulates the motion of a mobile robot and keeps track of the uncertainty in its configuration. Actually, I've already written most of it for you. You will just need to add the stuff that properly estimates the robot's pose based on its odometry and sensing.

In the first part of the assignment, the robot will execute only movement commands; the uncertainty in the robot configuration will increase due to accumulated errors from executing each movement. In the second part of the assignment, the robot will have a simulated laser rangefinder to measure the distance to an obstacle in a given direction. We assume the robot has a map, so it can compare the sensor reading with the expected value (based upon the current estimated robot configuration). You will implement an extended Kalman Filter (EKF) to merge this uncertain measurement with the current estimate of the robot configuration.

There will be one or two written questions for this assignment due on February 23. No late papers will be accepted for these questions because I will hand out solutions in class for these problems. Your program (and some additional written questions) will be due on March 2.

Support code libraries

The support code will make use of the *dolt-lite* library, as before. For this assignment, there is one more library: *SVL* (the Simple Vector Library). This library provides classes to represent vectors and matrices as well as mathematical operations, in particular the matrix inverse. See the assignment information page (off the course home page) for information on the SVL library.

1 Program operation and interaction

The support code for this assignment as working program that reads the problem specification files and already implements all the stuff that you need to display on the screen, albeit with "dummy" values. The things that are drawn on the screen (in addition to the world boundary and obstacles) are:

- the robot in its current estimated position and orientation (also include a point at the reference point of the robot)
- a 95.4% confidence limit on the robot's position (using an ellipse) and on the robot's orientation (using a cone)
- a path from the start, through all previous estimated final positions, to the current estimated position
- a point at the robot's true position and a line segment showing its true orientation

After this point, the user should be able to press '`q`' to quit or '`n`' to execute the next command. There are two types of commands: a "move" command and a "sense" command.

- If the command is a "move" command, then your program should:
  - Give the commanded motion to the robot simulator
  - Compute an updated estimate of the robot configuration based on the commanded motion
  - Update the display for the new configuration estimate and uncertainty

- If the command is a "sense" command, then your program should:
  - Display the beam from the robot (i.e. a line of the appropriate length in the proper direction from its current estimated configuration).
    Since the robot is not where it thinks it is, the end of this line will either not reach an obstacle or will intersect an obstacle.
  - Compute a revised estimate of the robot configuration and its uncertainty.
  - Wait for the user to press 'n' again, and then update the display.
    The beam should be removed from the display, and the last point of the robot path should be changed to the current estimated position.

Your program should also print a narrative of its calculations to `cout`. For example:

```
==>The robot moved: distance=1.75, angle=90
Current state estimate is (x, y, th) = (1.10114, -1.25, 86.0965)
New state estimate is (x, y, theta) = (1.22095, 0.505793, 181.969)
New state covariance matrix is:
[[0.0300499 -0.000998142 -0.00991148]
[-0.000998142 0.00375455 0.000676311]
[-0.00991148 0.000676311 0.0362557]]

True state is (x, y, th) = (1.15, 0.5, 180.000)
Uncertainty ellipse is:
  minor radius is 0.131259
  major radius is 0.35163
  orientation is 169.91084 degrees
  2*sigma_theta is 12.44284  degrees

==>Commanding the robot to sense at an angle of 90
The range is 2.00572

The predicted range was 2.00698
The derivatives are:
  dh/dx = 0, dh/dy = 1.00059, dh/dth = 0.0689879
Revised state estimate: (x, y, theta) = (1.22157, 0.504722, 181.936)
Revised covariance matrix is:
[[0.0203742 0.0134885 -0.00853237]
[0.0134885 0.0301436 -0.0188613]
[-0.00853237 -0.0188613 0.0181603]]
Uncertainty ellipse is:
  minor radius is 0.0481291
  major radius is 0.343865
  orientation is 75.51344 degrees
  2*sigma_theta is 10.01445 degrees
```

Don't use the above numbers to test your program — I just made up at least some of them!

2 Simulated robot operation

The support code will provide a simple robot simulator. The motion commands to the robot will be a forward straight motion followed by a turn in place (about the reference point). The forward distance and turning angle are the parameters for a command.

The robot, however, does not execute the commanded motion perfectly. It adds some Gaussian noise to the commanded motion and updates the actual configuration of the robot accordingly. For this assignment, the parameters of the commanded motion are not relevant. We are trying to recover the true robot pose, i.e., how the robot actually moved, using only the odometry and the sensing.

The odometry gives a measurement of the actual motion of the robot but with some Gaussian noise added. The width of these Gaussian distributions depends on the magnitude of the motion, i.e., smaller motions have less error. We will assume that the standard deviation (sigma) of the Gaussian is proportional to to magnitude of the motion command. The constants of proportionality are given in the problem specification file:

```
executionDstSigmaRatio
executionAngSigmaRatio
odometryDstSigmaRatio
odometryAngSigmaRatio
```

For example, if `odometryDstSigmaRatio` is 0.02, then for an actual motion of $d = 10$ m, the Gaussian random noise of the odometry measurement would have a standard deviation of $\sigma_d = 0.2$ m. You should assume that the error in distance for the forward motion and the error in angle for the turn are independent.

This robot will be equipped with a simulated a laser rangefinder. There is one parameter to a sensing command: the angle (relative to the forwards direction of the robot) in which to point the laser rangefinder. We will assume that the sensor is pointed exactly at the specified angle. The simulated robot will calculate the sensor reading using the true robot pose but will add some noise to this reading. This is done by the `getRange` method. The amount of this noise is controlled (in a similar way to the odometry noise) by the parameter `rangeSigmaRatio`.

In order to apply the EKF, you will need to predict what the sensor reading should be based upon your current estimated pose of the robot. This is done by the `predictRange` method. There is no noise added to this predicted value.

3 The extended Kalman filter

The state of the robot will be the vector $x = [\ x \quad y \quad \theta\ ]^T$ and the motion command will be the input vector $u = [\ d \quad a\ ]^T$, where d is the distance and a the angle as measured by the odometry. We will not use the commanded distance or angle at all. Assume that the robot first translates a distance d (at its current heading) and then rotates an angle a.

We have assumed that there is noise associated with the input u, so we must represent this using the covariance matrix:

$$P_u = \begin{bmatrix} \sigma_d^2 & \sigma_{da} \\ \sigma_{ad} & \sigma_a^2 \end{bmatrix} \tag{1}$$

At each step, we will maintain an estimate of the state, notated $\hat{x}(k)$ for step k. To characterize its uncertainty, we maintain an associated state covariance matrix (here, also for step k):

$$P_x(k) = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{y\theta} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_\theta^2 \end{bmatrix} \tag{2}$$

At the beginning, when we know the robot's configuration exactly, the elements of the state covariance matrix are all zero.

3.1 Compounding

We use Φ to represent the function that calculates the next state as a function of the current state estimate and the input. Our model of the system is:

$$x(k+1) = \Phi[x(k), u(k)] \tag{3}$$

Note that this differs slightly from the usual formulation of the extended Kalman filter in that there is no additional noise term $w(k)$. This is because we are assuming that there is noise associated with the elements of $u(\cdot)$ instead.

At step k, we have an estimate of the state $\hat{x}(k)$ and its covariance $P_x(k)$. To update the estimate, we compute:

$$\hat{x}(k+1|k) = \Phi[\hat{x}(k), u(k)] \tag{4}$$

Recall that $\hat{x}(k+1|k)$ is a preliminary estimate of the new state after the motion command at step k.

In order to compute the covariance of this preliminary estimate, we must linearize the system. The first order approximation to Φ about the current state estimate is:

$$\Phi[x, u] \approx \Phi[\hat{x}(k), u(k)] + J_x(x - \hat{x}(k)) + J_u(u - u(k)) \tag{5}$$

where J_x and J_u are the Jacobians of Φ:

$$J_x = \begin{bmatrix} \frac{\partial \Phi_x}{\partial x} & \frac{\partial \Phi_x}{\partial y} & \frac{\partial \Phi_x}{\partial \theta} \\ \frac{\partial \Phi_y}{\partial x} & \frac{\partial \Phi_y}{\partial y} & \frac{\partial \Phi_y}{\partial \theta} \\ \frac{\partial \Phi_\theta}{\partial x} & \frac{\partial \Phi_\theta}{\partial y} & \frac{\partial \Phi_\theta}{\partial \theta} \end{bmatrix} \qquad J_u = \begin{bmatrix} \frac{\partial \Phi_x}{\partial d} & \frac{\partial \Phi_x}{\partial a} \\ \frac{\partial \Phi_y}{\partial d} & \frac{\partial \Phi_y}{\partial a} \\ \frac{\partial \Phi_\theta}{\partial d} & \frac{\partial \Phi_\theta}{\partial a} \end{bmatrix} \tag{6}$$

Note that these Jacobians should be evaluated at $\hat{x}(k)$ and $u(k)$.

The preliminary state covariance can then be computed as:

$$P_x(k+1|k) = J_x P_x(k) J_x^T + J_u P_u(k) J_u^T \tag{7}$$

If there is no measurement after the motion, then this preliminary estimate becomes the final state estimate at step $k+1$, i.e. $\hat{x}(k+1) = \hat{x}(k+1|k)$ and $P_x(k+1) = P_x(k+1|k)$.

3.2 Merging

When we do have a measurement at step k, this information must be incorporated into the state estimate and its covariance. We assume that this measurement process can be modeled by the equation:

$$z = H[x] + v(k) \tag{8}$$

The H function computes the sensor value from the state, and v is a random variable representing the noise in the measurement. In our problem, the measurement z will be a scalar, the measured range from the sensor.

We compute the error between the actual measurement $z(k)$ and the predicted measurement (without noise) based on our best estimate of the state:

$$r(k+1) = z(k+1) - H[\hat{x}(k+1|k)] \tag{9}$$

Now we can compute the final state estimate at step k

$$\hat{x}(k+1) = \hat{x}(k+1|k) + K(k+1)r(k+1) \tag{10}$$

where $K(k+1)$ is the Kalman gain:

$$K(k+1) = P_x(k+1|k)J_H^T(J_H P_x(k+1|k)J_H^T + C_v)^{-1} \tag{11}$$

Here, J_H is the Jacobian of the H function and C_v is the covariance matrix for the measurement noise. The new state covariance matrix is:

$$P_x(k+1) = (I - K(k+1)J_H)P_x(k+1|k) \tag{12}$$

Note that the `predictRange` method will return the values of the derivatives that you need for the Jacobian J_H.

4 Miscellaneous notes

- For orientation uncertainty, assume that this is a 1-dimensional Gaussian with variance σ_θ^2; the 95.4% confidence limit is the interval $\pm 2\sigma_\theta$ about the mean.
- For position uncertainty, take the upper left 2×2 submatrix of P_x. See the notes, Section 5.1.10, for information on how to calculate ellipse parameters from this covariance matrix.
- I suggest you adopt the following convention: all internal representation of angles should be in radians, all external representations (e.g. when printed to the screen or read from a file) should be in degrees.