

Localization and Kalman filtering

For this assignment, you will simulate the motion of a mobile robot. In Part A, you will keep track of the uncertainty of the robot's position which will increase with time due to the accumulation of dead reckoning errors. In Part B, you will implement a simulated laser range sensor which will measure the distance (with some error) from the robot to an obstacle. You will use the Extended Kalman filter to combine the position estimates from dead reckoning and from the laser range sensor.

For both parts of the assignment, you will read in a file which contains the instructions your robot is to execute (i.e. what motion to make and when to use the sensor). You will need to display graphical output to show the estimated position of the robot and its uncertainty. You will also need to use CGAL (or equivalent code) in order to simulate the sensor readings.

The written questions for Parts A and B are due on February 22. Your program is due on March 1. I will specify a few tests for you to run with your program (which may involve a few more written answers for March 1).

A. Dead reckoning

For this part of the assignment, you are to write a program that reads in a problem description file which will specify certain parameters for the robot and a list of motion instructions. Your program should initialize the robot simulator accordingly, and then execute each motion. After each motion, your program should graphically display the true position, the estimated position, a 95% elliptical uncertainty limit for the robot's estimated position, and a 95% uncertainty "cone" for the robot's estimated heading.

A.1 A robot simulator

I will provide C++ code that will serve as the basis for your robot simulator. This code should be all you need for this part; you will need to extend it for the second part of this assignment. The files `robot.cc` and `robot.h` will be available off the course web page. These depend on the files `rand.cc` and `rand.h` which contains a function to generate a pseudorandom number from a Gaussian distribution. Many aspects of this simulator are described in the comments (or are apparent from the code).

The basic simulator is very simple. It maintains the real position of the robot and adjusts it after every motion command. There is a method to access the real position of the robot, but you should use this for display purposes only.

Note that although you may command the robot to move a certain distance, the robot will not move exactly this distance. Some gaussian noise is added to the requested distance in order to determine the actual distance moved. Additional noise is added to the actual distance in order to determine the odometry measurement.

A.2 Problem specification file

Here is a sample problem specification file:

```

# Problem 1
part= A

# seed for the random number generator
seed= 3389943

# The world file contains the boundary and obstacles
# (You don't need the world for Part A, but you could display it if
# you want.)
world= world1.txt

# The robot's start position (x y theta) with theta in degrees
start= 1.4 2.8 34.2

# standard deviations for the gaussian noise generated by the simulator
dSigmaFrac= 0.05
aSigmaFrac= 0.05
dExecSigma= 0.03
aExecSigma= 0.02

# parameters for the simulated laser range sensor
lSigmaFrac= 0.01

# Instructions of the form "distance angle" follow the "intructions" keyword.
# The angle is the amount to turn specified in degrees.
#
# The keyword "sense <heading>" may appear after the angle. In Part B, this
# is when you will simulate a sensor reading. The heading is given in
# degrees; 0 degrees is straight ahead.
instructions=
3.2 -45.0
1.1 87.5 sense 35.2
2.8 53.0

```

A.3 Combining odometry measurements

Represent the configuration of the robot at step k as $x(k) = [x \ y \ \theta]^T$ and the input as $u(k) = [d \ a]^T$ where it is assumed that the robot first moves a distance d along the current heading and then turns an angle a . We can write the state update equation as:

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

You will need to linearize this system which will put it in the following form:

$$\hat{x}(k+1) = \Phi\hat{x}(k) + \Gamma u(k) \quad (2)$$

where Φ and Γ are Jacobians defined by:

$$\Phi_{i,j} = \frac{\delta\Phi_i[\hat{x}(k), u(k)]}{\delta x_j} \quad (3)$$

$$\Gamma_{i,j} = \frac{\delta\Phi_i[\hat{x}(k), u(k)]}{\delta u_j} \quad (4)$$

Note that Φ is used to represent both the nonlinear function and a Jacobian as in our text.

Let the state covariance matrix at step k be $P_x(k)$ and the covariance of the input u be $P_u(k)$. The state covariance matrix is updated according to:

$$P_x(k+1) = \Phi P_x(k) \Phi^T + \Gamma P_u(k) \Gamma^T \quad (5)$$

A.4 Drawing elliptical uncertainty limits

To indicate the uncertainty of the robot's position, you will draw a 95% elliptical uncertainty limit. Since the underlying probability density functions are Gaussian, this means you can simply draw an ellipse whose major and minor "radii" are 2σ for the respective axes. The covariance matrix defines the equation of an ellipse, for example:

$$x^T P_x x = 1$$

However, this does not exactly help you draw the ellipse. In two dimensions, we can write the major and minor axes as the vectors $m_1 = [\cos \theta \quad -\sin \theta]$ and $m_2 = [\sin \theta \quad \cos \theta]$. These vectors will form a basis that will diagonalize the covariance matrix, i.e.

$$P_x = M^T \begin{bmatrix} \sigma_{m_1}^2 & 0 \\ 0 & \sigma_{m_2}^2 \end{bmatrix} M$$

If P_x is of the form:

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

then a bit of matrix multiplication, algebra, and trigonometry yields:

$$\tan 2\theta = \frac{2b}{a-c} \quad (6)$$

$$\sigma_{m_1}^2 = a + b \tan \theta \quad (7)$$

$$\sigma_{m_2}^2 = c - b \tan \theta \quad (8)$$

A.5 Written questions

1. Write the actual equations to calculate the state estimate update and the state covariance matrix update. (I.e. write down the nonlinear state update function $\Phi[\hat{x}(k), u(k)]$, find its Jacobians, etc.)
2. One might expect the state covariance matrix to always be of the form:

$$P_x = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & 0 \\ \sigma_{xy} & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_y^2 \end{bmatrix}$$

Why isn't it? What does this mean about the 3-dimensional ellipsoidal uncertainty limit?

B. Sensing and the Extended Kalman Filter

For this part, you will extend your program to simulate a sensor reading and then use the extended Kalman filter to combine this information with the robot's position estimate.

If there is a "sense" command on a line in the instruction section of the problem file, then you should move the robot, simulate the sensor reading, combine this reading and the robot state estimate, and then draw the uncertainty limits for position and heading.

If you did not draw the world on the screen for Part A, you should do so for this part.

B.1 Enhancing the robot simulator

In order to simulate the laser range finder, you will need to intersect a ray starting at the robot's real position in the direction of the specified heading. We will assume that there is no error in the heading, but there is Gaussian noise in the distance reading. After you calculate the actual distance, you should add to it a pseudorandom number from the appropriate Gaussian distribution.

If you are using CGAL on a Windows platform (where we have had trouble getting the intersection routines to work), then I suggest the following strategy: start with a short line segment and see if it intersects any obstacle (or the boundary). Increase the length of the line segment (doubling it each time or using a fibonacci sequence) until it intersects one obstacle and then use binary search to find the approximate intersection point.

B.2 The extended Kalman filter

After you have computed an updated state estimate and covariance based on the dead reckoning information (which we now refer to as $\hat{x}(k+1|k)$ and $P_x(k+1|k)$), you will combine this with the simulated sensor reading.

The sensor reading $z(k) = h[x(k), \xi]$ will be a nonlinear function of the robot state $x(k)$ and the environment ξ . You will need to linearize this function so it is of the form:

$$z(k) = \Lambda x(k) \quad (9)$$

where Λ is the Jacobian. You can then compute the Kalman gain:

$$K(k+1) = P_x(k+1|k)\Lambda^T[\Lambda P_x(k+1|k)\Lambda^T + P_z(k)]^{-1} \quad (10)$$

where $P_z(k)$ is the covariance matrix for the measurement $z(k)$ (denoted $C_w(k+1)$ in our text).

We next compute the difference between the measurement and the expected measurement:

$$r(k+1) = z(k+1) - h[\hat{x}(k+1|k), \xi] \quad (11)$$

The combined state estimate is now given by:

$$\hat{x}(k+1) = \hat{x}(k+1|k) + K(k+1)r(k+1) \quad (12)$$

The new state covariance is:

$$P_x(k+1) = [I - K(k+1)\Lambda]P_x(k+1|k) \quad (13)$$

B.3 Written questions

1. Write the actual equations and/or algorithm that you will use to implement the extended Kalman filter.