CSE 483: Mobile Robotics - Assignment 03

Due: **23rd September 2016**, 2300 hrs

September 19, 2016

General Instructions

1. **Form teams of 2 people for this assignment.** This assignment is fairly long, and you would immensely benefit from collaborations. Working alone is *discouraged.*
2. The assignments are to be done in Matlab. The code provided along with the assignments may run in Octave as well, but we provide no guarantees on that front. The evaluation will be done in MATLAB R2015a.
3. Plagiarism is strictly prohibited. The popular *MOSS* tool will be employed for code plagiarism detection.
4. Ensure that the plots included in the report are reproducible from the code that you submit, failing which measures such as penalization of marks and a deeper investigation for plagiarism ensue.
5. **In all your scripts and functions, add the following random seed '201602', for repeatability**. If you do not know what a random seed is, look it up. In Matlab, just adding the line `rng(201602)` at the top of the main script would suffice.

1 EKF Localization (20 points)

In this assignment, we will implement the Extended Kalman Filter (EKF) in a localization setting. A basic framework for the code is provided along with the assignment. You are REQUIRED to stick to the framework. Do NOT modify already written code. Only add code in functions that need to be filled-in.

Disclaimer: At a lot of places, I haven't explicitly mentioned if I'm referring to μ or $\hat{\mu}$, or if I'm referring to z or $\hat{z}$. This lack of clarity is purely intentional, and this is done because, that part is often the trickiest, and results in good brainstorming sessions.

1.1 Milestones

The following functions in the provided code need to be filled-in for successful implementation of the EKF. Some of them are from the previous assignment.

1. `runMotionModel.m` - updates the robot's state, using the control command, and the motion model
2. `computeJacobianState.m` - computes the jacobian of the motion model with respect to the state
3. `computeJacobianControl.m` - computes the jacobian of the motion model with respect to the control
4. `getMeasurements.m` - fires the sensor and receives measurements
5. `computeJacobianObs.m` - computes the jacobian of the observation with respect to the state
6. `runIncrementalEKF.m` - the main file which implements the EKF localization in the incremental update mode
7. `runBatchEKF.m` - the main file which implements the EKF localization in the batch update mode
8. `perfAnalysis.m` - computes the localization error, to analyze the performance of the EKF
9. `sane_mvnrnd.m` - sample sanely from mvnrnd (refer to the previous assignment for details)
10. `get_error_ellipse.m` - plots the uncertainty ellipse for a 2D gaussian

The following functions/scripts are already provided (to facilitate simulation), and need not be edited.

1. `updateSimulation.m` - updates the GUI
2. `startup.m` - script to be executed at startup; sets the bare-bones structs required for the code
3. `globals.m` - declares global variables
4. `initSimulationParams.m` - initializes simulation parameters

1.2 Basic workflow

This section provides a broad overview of what you need to do for the assignment. For a more detailed statement, go through the documentation present in the code.

The current environment provides you with two structs (MATLAB structures), viz. `params` and `robot`. The `params` struct contains several simulation parameters, including the environment (map of landmarks), control actions the robot would take, etc. You have two write two main scripts `runIncrementalEKF.m` and `runBatchEKF.m`. However, for these files to run as expected, you need to fill out other functions, most of which are already provided.

The directories `maps` and `controls` contain different maps the robots can be tested on, and different sets of controls the robots can run (straight line trajectories, circular trajectories, etc.). For now, I've only released *toy* samples that you can test on, but towards the last week, I'll post large maps as well.

The sample script `testSimulation.m` contains a snippet that loads a map and controls. Adapt the same when you write `runIncrementalEKF.m` and `runBatchEKF.m`.

1.3 The EKF Cycle

An EKF can be viewed as a two-step process - a state prediction step, and a state update step. Here, we assume that the robot's state is characterized by a multivariate gaussian distribution. Hence, a state is represented by a mean vector and a covariance matrix.

1.3.1 Prediction Step

Robot's do not execute their controls perfectly. We model this by incorporating noise in the robots motion. In our code, we have the robot's state stored in the structure `robot.pose`. This struct has three elements `x`, `y`, and `theta`, which store the robot's (x, y) location and the heading θ respectively.

To simulate a discrete time step, we run a loop from 1 to T, where T is the time limit upto which EKF is to be run. In each time step, the robot does the following:

- Run the motion model to predict the next state
- Update the state covariance using the jacobians of the motion model
- Run the prediction step (more on this in a subsequent section)

To run the motion model, we first take the next control, and corrupt it by noise (that noise must be *sanely* sampled from the control noise distribution). However, the robot is unaware of what exact noise occured in the control. According to the robot, it has executed the ideal control. Use the motion model and the ideal control to update the robot's state. Intuitively speaking, this state is *where the robot thinks it is.* For the purposes of measuring localization error and thus analyzing performance, we would also want to store *where the robot actually* *is.* This can be again obtained by keeping track of all true states in another variable.

Once the robot has predicted its new state, we linearize the motion model about the mean of the estimate. This is done since the kalman filter's update step works only for linear models. We linearize the motion model by Taylor series expansion, as discussed in class. To this end, implement the functions that compute the jacobian of the motion model with respect to the state and the control.

Using the computed jacobians, update the covariance of the robot's state. This wraps up the prediction step of the EKF.

1.3.2 Update Step

To update the robot's estimate of state, we make use of measurements (a.k.a. observations or sensor readings). We assume that the robot is equipped with a range-bearing sensor that can measure all landmarks within sensing range. Its uncertainty is characterized by a gaussian with zero-mean and some covariance matrix R.

We fire the sensor and get the true sensor measurements, i.e., we get the range and bearing measurements to all landmarks within a stipulated sensor radius. We then corrupt the observations by the observation covariance. These measurements can intuitively be thought of as *what the robot actually sees.*

In the above procedure, note that the measurements are obtained from the *true* pose of the robot, not from *where the robot thinks it is.*

We now compute *expected measurements*, i.e., *what the robot expects to see* from *where the robot thinks it is.* This serves as a second estimate of state, which can be used to refine the estimate obtained from the motion model.

Run the EKF update equations, as discussed in class. You may also refer to the book *Probabilistic Robotics* for the same. At a coarse level, the update involves, computing the Kalman gain, using the jacobian of the observation model, and then computing the updated mean and covariance (using the Kalman gain).

Note that there are two primary ways in which updates can be done, viz. incremental and batch modes. In the incremental mode, you perform the update on a per-landmark basis, i.e., you run a loop for each observation (inside the loop that runs for every time step) and carry out Kalman gain computation, etc. inside the loop. In the batch update, you compute a joint update for all landmarks visible in a particular time step. A more extensive treatment of the incremental and batch modes is available in the *Probabilistic Robotics* book.

After this, run the `updataSimulation.m` script to update the GUI.

Compute the error in the estimated pose and the true pose of the robot at each iteration, and plot the error. Include the plots in the report. Needless to say, the plots must be labeled and aptly titled.

2 Bells and Whistles (10 points)

1. Instead of the motion model discussed in class, use the (α, T, β) motion model which you completed in assignment 02. Make appropriate ammendments to all other parts in the prediction and update steps. Plot the localization error.

2. Instead of the motion model discussed in class, use the (δ_x, δ_y) motion model. Under this motion model, you assume that the robot is holonomic (in a simplistic sense, the robot does not have to rotate to move between any two points in the plane). Assume that the robot's orientation always remains zero, and the control input that you give the robot is (δ_x, δ_y), where δ_x is the amount of translation in the X-direction and δ_y is the amount of translation in the Y-direction. Derive the motion model for the system (include it in your report). Implement the EKF using this model and plot the localization error.

3 Addendum

Added on 19th September 2016.

3.1 Some Clarifications

3.1.1 Observation Normalization

You need to normalize ALL observation angles. Concretely speaking, if your measurement $z = [r, \psi]$ is not normalized (i.e., if psi does not lie in the range $[0, pi]$), it may lead to some of the following catastrophic consequences.

1. Catastrophy 1: Your robot would expect an observation with bearing α, but it gets one with bearing $\pi - \alpha$, which results in a non-zero innovation vector (the innovation vector is $z - \hat{z}$).

2. Catastrophy 2: Your robot gets an observation with bearing α, but it gets one with bearing $\pi - \alpha$, which results, again, in a non-zero innovation vector.

In both the above cases, your uncertainties would blow up, and, given that you would not normalize in the future as well, the EKF estimates will potentially NEVER converge.

3.1.2 Numerically (Un)stable Covariance Matrices

Many may have noticed that their covariance matrices are not exactly symmetric (they differ at the fifth or sixth decimal place). To overcome this, you have two options.

1. Initialize covariances by squaring them, as opposed to initializing diagonal values directly. This tricks helps only to a certain extent.
2. Symmetrize the covariance matrix. A (square) matrix $\mathbf{C}$ is symmetrized by replacing C with $\frac{1}{2}\left(\mathbf{C}+\mathbf{C}^T\right)$.

A more formal way to approach it would be to realise that $(\mathbf{I}-\mathbf{KH})\hat{\mathbf{\Sigma}}$ was an algebraic simplification made in the derivation of the EKF (for the multi-variate case, which wasn't done in class). That update holds only when K is in-fact the optimal Kalman gain. In case of EKF, it definitely isn't. In such a situation, one must use the *Joseph form* of the Extended Kalman Filter.

- **Short version** Use the update rule:

$$\mathbf{\Sigma} = (\mathbf{I}-\mathbf{KH})\hat{\mathbf{\Sigma}}(\mathbf{I}-\mathbf{KH})^T + \mathbf{KQK^T} \tag{1}$$

- **Long(er) version** Look at the following online resources:
  http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated3/kleeman_kalman_basics.pdf

  https://en.wikipedia.org/wiki/Kalman_filter
- **Still not clear?** Use the TA Office Hours!

3.1.3 Bugs in the codebase

Despite the quality measures adopted, there are a few bugs in the current code. Specifically, there are (minor) bugs (that lead to major consequences) in drawing the robot (in updateSimulation.m), and in plotting the ellipse (in get_error_ellipse.m). They've been rectified now, although none of the students (surprisingly) reported any of them.

3.2 Analysis Required

In addition to the above tasks, the following analysis is required (for the first task only, i.e., you need not do the following for the *Bells and Whistles* task, but you NEED to do it for the *EKF Localization* task).

1. For the `random_small.map` map and controls (provided with this version of the document), plot the error in the state estimate, i.e., abs$(\hat{x} - x_{\text{true}})$. Have an independent plot for each state variable x, y, θ.

2. In each plot for x, y, θ, have one error curve for the incremental mode and one for the batch mode. Note the fact that *state error* needs to be computed only once per state in batch mode as well as incremental mode (irrespective of the number of landmarks are visible from that state).

3. Make another set of plots for errors in x, y, θ for *Joseph form* of the covariance update, vs the *optimal Kalman gain form* (the optimal Kalman gain form is the one we did in class).

4 Deliverables

1. A zipped (`.zip`) folder (try not to use any other format such as `.bz2`, `.tar.gz`, etc.) whose name is composed of the ID numbers of the two team members, separated by an underscore, eg. `201507666_201507555.zip`. The folder should have a `report.pdf` file which contains the other deliverables for the assignment.

2. Additionally the folder `03_EKF`, provided along with the assignment, must be present. The `.m` files inside this folder must contain the modified code.

3. Please keep the input/output signatures for each function the same. Also do not change the directory structure, and do not add additional files. The code will be evaluated on a set of test cases, and common input/output signatures are required for automated testing.