CALIFORNIA INSTITUTE OF TECHNOLOGY

ME/CS 132a, Winter 2011

Lab #2 Due: Mar 10th, 2011

Part I
Data Association for SLAM

1 Introduction

For this part, you will experiment with a simulation of an EKF SLAM system and investigate approaches to robust data association. The scenario is a mobile robot with wheel odometry and a laser range-finder sensor which is driven around a square corridor. There are point features around the corridor whose positions relative to the robot can be observed by the sensor. The features are mapped and their locations estimated along with that of the robot within an Extended Kalman Filter (EKF).

2 Starting the Simulation

The main program of the SLAM simulation is the text file `slam.m`. This is a program in the Matlab M-file format. Other files in the `tools/` directory contain functions called by the main program.

In the main Matlab window, type `slam` to start the simulation. This starts the main function defined in `slam.m`. A window marked 'Figure 1' will pop up with a view of the simulation scenario: a set of red points represent the ground-truth locations of landmark features which the robot is able to observe as it moves around a square, corridor-like trajectory. The simulation will be paused and pressing `Enter` will then take it through the Action Sequence below, pausing again after each action. At any time when the program is paused you can hit `Control-C` to come out (then type `slam` to start again).

3 Displays & Output

Alongside the main Matlab window, various displays will pop up which have the following roles:

Figure 1	Ground Truth	Shows the ground-truth layout of features
Figure 2	SLAM Map	Shows the latest estimated robot and feature positions with uncertainty ellipses (in blue) on top of the ground truth positions (in red)
Figure 3	Observations	Shows the latest sensor measurements and uncertainties in these (in green) in the robots coordinate frame
Figure 6	Compatibility	Shows how feature measurements are checked for matching using different methods
Figure 7	Matching	Shows the assignments by different methods of which observations match which features

The simulation also outputs data association information at each timestep in text form. To help explain the output, an example is given below. Seven observations are made and the data association algorithm attempts to determine if these observations correspond to landmarks in the map or are of a previously unobserved landmark. In this example, the algorithm correctly matched the observations of the landmarks with IDs 14, 6, 8, and 18. It also correctly recognized that the fifth observation was of a new landmark (indicated by 0 and then given the ID 2 when it is added to the map). The observation of landmark 15 was incorrectly associated with landmark 16 (a false positive). The observation of landmark 17 was incorrectly thought to be a new landmark (a false negative).

```
------------------------------------------------------------
Step: 9
GROUND TRUTH:  14  15  6  17  0  8  18
MY HYPOTHESIS: 14  16  6   0  0  8  18
Correct (1/0)?  1   0  1   0  1  1   1

NEW:  2
Hypothesis not in agreement with ground truth
True positives:  4
True negatives:  1
False positives: 1
False negatives: 1
------------------------------------------------------------
```

After the simulation is complete, the statistics for the data association performance over the entire run is displayed as well as a graph of the error and uncertainty of the vehicle state estimate.

4 Action Sequence

Each time you hit Enter, the simulation will carry out the next action of the following sequence (note that at the very start of the simulation, there are extra Get observations and Update steps before the robot moves for the first time to stock the map with some initial features):

1	Robot moves	The robots motion is estimated with odometry and displayed in Figure 2.
2	Get observations	The sensor gets a set of measurements and the results are displayed in Figure 3.
3, 4, 5	Compatibility	The robot checks which measurements match up to mapped features (Figure 6). Trying different methods is the goal of this practical.
6	Matching	Using one of these methods (as selected in the program) a match set is decided on and these are displayed in Figure 7.
7	Update	The robot and feature positions are updated, and any new features are added to the map (Figure 2).

After all of these actions, one full step is complete and the program cycles back to action 1.

5 Simulation Parameters

The file `slam.m` contains various parameter settings which can be adjusted to produce different behavior.

- `configuration.step_by_step`: when set to 1 the program will pause after every movement step allowing you to analyze each data association solution in detail; set it to zero and the program will run continuously right to the end. You will see the robot come right round the square corridor and close the loop at the end. Observe how the uncertainty in the positions of new features (and in the position of the robot) increases as the robot moves around the loop, but then shrinks back down once the robot manages to re-observe early features and close the loop. This is classic SLAM behavior.
- `configuration.people`: set to 1 and rogue measurements will be reported by the sensor, corresponding to the effect of people walking in front of the robot.
- `configuration.odometry`: set to 0 and odometry will not be available, so the data association algorithm will have to work with large motion error and uncertainty.

5.1 Measurement Uncertainty

It is possible to change the parameters of the sensor to simulate a range-finder which is more or less accurate, or has a different range. You can try increasing the measurement standard deviations and see how the feature ellipses change.

- `sensor.srho`: the standard deviation of range measurements from the sensor.
- `sensor.stita`: the standard deviation of angular measurements.
- `sensor.range`: the depth range of the sensor.
- `sensor.minangle`, sensor.maxangle: the angular range of the sensor.

6 Assignment

Initially, the simulator has a perfect data association algorithm provided by the Ground Truth. Set `configuration.step_by_step` to 0 and let the simulation continue to the end where the loop is closed. You can see that the quality of the estimate is good since the estimated position of the landmarks and vehicle is close to the ground truth. Also, the uncertainty in the estimate is good since error typically falls within the 3 sigma uncertainty bounds. In the real world, ground truth is not available so a data association algorithm must be used to determine which landmarks an observation corresponds to. You will use this simulation to investigate three different data association algorithms.

1. [5 points] Try the Nearest Neighbor (NN) data association algorithm by changing the appropriate line in `data_association.m`. Comment on the quality of the estimate produced. Set `configuration.step_by_step` to 1 (true) to look at the data association process for a step in detail. Why does Nearest Neighbor sometimes fail?

2. [10 points] You will now implement a data association algorithm called `SINGLES` which avoids using ambiguous observations. The file `SINGLES.m` has been provided to get you started. Be sure change `data_association.m` to use `SINGLES` rather than `NN`. Your algorithm should associate an observation with a landmark if:

   - no other observation is individually compatible with that landmark
   - and no other landmark is individually compatible with that observation.

   `SINGLES` should be able to cope better by rejecting the ambiguous observations resulting from the pairs of nearby landmarks in this environment.

3. [5 points] Now include moving people in the simulation by setting `configuration.people` to 1 and try the `NN` and `SINGLES` algorithms. Why do they fail?

4. [5 points] Try using the Joint Compatibility Branch and Bound Algorithm (JCBB) implemented for you in `JCBB.m`. Why can this algorithm cope with people better?

5. [10 points] Try each algorithm without odometry by setting `configuration.odometry` to 0. This results in a larger motion uncertainty. How do the different algorithms perform?

6. [10 points] Moving people are initially added to the map as a new landmark. In later timesteps, observations of these spurious landmarks are rejected by JCBB. In `slam.m`, modify the commented-out code starting from Line 107 to include a map maintenance procedure. The procedure should eliminate all features seen only once and more than two steps ago. This will remove most of those spurious landmarks from the map and so reduce the update time and the chance of future data association to these spurious landmarks.

Part II
EKF Localization

1 Introduction

In this lab you will produce a navigation system for a mobile robot. This system will build on the work done in the previous lab. You will use the system you made then to make observations of visual landmarks which will be used to determine the position of the robot as it moves through a known map. The localization algorithm you should use is the Extended Kalman Filter which has been discussed in class.

For this lab you will be provided with a dataset which includes:

- Sensor measurements from a robot as it moves through the environment:
  - Timestamps
  - Motor control inputs
  - Images from the stereo pair (right camera only)
  - The depth map from the stereo pair
- Calibration information:
  - Camera-frame to robot-frame transformation

- A map of known landmarks:
  - A reference image for each landmark
  - The 2D location of each landmark
- The approximate pose for the robot at the starting position

You will need to write code for several different components of the localization system. Each of these is described below. You may use Matlab or another programming language if you prefer.

2 EKF Localizer

A localization algorithm should produce an estimate of the pose of the robot given the map and the sensor measurements. In this assignment, we assume that the robot's motion is restricted within a plane, and therefore reduce the localization problem to a 2D one. The pose of the robot can be represented using its 2d coordinates (x, y) and its orientation θ. The estimate can be maintained in an Extended Kalman filter with state

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{\theta} \end{bmatrix} \tag{1}$$

and 3×3 covariance matrix

$$\mathtt{P} = \begin{bmatrix} \mathtt{P}_{xx} & \mathtt{P}_{xy} & \mathtt{P}_{x\theta} \\ \mathtt{P}_{yx} & \mathtt{P}_{yy} & \mathtt{P}_{y\theta} \\ \mathtt{P}_{\theta x} & \mathtt{P}_{\theta y} & \mathtt{P}_{\theta\theta} \end{bmatrix} \tag{2}$$

Your system should perform the EKF prediction step each time it gets the control inputs and an EKF update step each time it gets a new visual observation. In the dataset we supply, we have synchronized the sensors to have the same timestep. In this case, apply the motion prediction first and then the update step.

3 Motion Model

This module is needed for the EKF prediction. This module updates the estimate of the robot pose according to the latest motor control inputs, $\mathbf{u} = [u_1, u_2]^\top$. The uncertainty inherent in the motion also leads to an increase in the uncertainty in the robot pose estimate.

$$\hat{\mathbf{x}}(k|k-1) = f(\hat{\mathbf{x}}(k-1|k-1), \mathbf{u}) \tag{3}$$

$$\mathtt{P}(k|k-1) = \mathtt{F}_x \mathtt{P}(k-1|k-1)\mathtt{F}_x^\top + \mathtt{F}_u \mathtt{Q} \mathtt{F}_u^\top \tag{4}$$

In the absence of any visual observations this module should allow you to estimate the trajectory of the robot using dead reckoning.

The control inputs come in the form of a forward and an angular velocity w.r.t. the body frame which can then be converted into velocities in the map frame using the current robot orientation. This gives the continuous-time motion model:

$$\dot{x} = u_1 \cos\theta \tag{5}$$

$$\dot{y} = u_1 \sin\theta \tag{6}$$

Figure 1: Observations of known landmarks are used to localize the robot.

$$\dot{\theta} = u_2 \tag{7}$$

In the dataset, we will provide recorded u_1 (forward body velocity) and u_2 (angular body velocity) in m/s and rad/s, respectively. You can assume the covariance matrix `Q` to be diagonal:

$$\mathtt{Q} = \begin{bmatrix} \sigma_{u1}^2 & \\ & \sigma_{u2}^2 \end{bmatrix} \tag{8}$$

with $\sigma_{u1} = 1.0$ cm/s and $\sigma_{u2} = 4$ deg/s.

4 Visual Observations

The robot is equipped with a stereo camera to make observations of visual landmarks in the environment. These landmarks are the images that you detected on the wall in the previous lab. The first stage is to use your code from the previous lab to determine the position of these landmarks relative to the robot for each image in the sequence. These are your visual observations.

Use the SIFT detector to determine the pixel location of the landmark and your homography code to find the pixel coordinates for the four corners of the landmark. Then use the stereo depth map in the dataset to determine the coordinates of the corner points in the camera frame. The (x, z) coordinates of the corners of each landmark in the camera frame are the observations for your EKF system. Although the dataset gives the 3D coordinates of the corners, we have ignored the y-component because the y-axis in the camera frame corresponds the z-axis in the map frame, and therefore provides no useful information. Refer to the end of this section for the convention used in converting a point in the robot's body frame to that in the camera frame.

You will then have to produce an Observation Model. This model can predict what the observation would be given a pose for the robot and the known position of the landmark.

$$\underbrace{\mathbf{z}(k|k-1)}_{\text{Prediction}} = \underbrace{h(\hat{\mathbf{x}}(k|k-1))}_{\text{Observation Model}} \tag{9}$$

It is the difference between the prediction and the true observation which the EKF uses to improve the state estimate during the EKF update.

$$\underbrace{\hat{\mathbf{x}}(k|k)}_{\text{new state estimate}} = \underbrace{\hat{\mathbf{x}}(k|k-1) + \mathtt{W}\boldsymbol{\nu}(k)}_{\text{prediction and correction}} \tag{10}$$

$$\underbrace{\mathtt{P}(k|k)}_{\text{new covariance estimate}} = \underbrace{\mathtt{P}(k|k-1) - \mathtt{W}\mathtt{S}\mathtt{W}^\top}_{\text{update decreases uncertainty}} \tag{11}$$

where

$$\overbrace{\boldsymbol{\nu}(k)}^{\text{innovation}} = \overbrace{\mathbf{z}(k)}^{\text{measurement}} - \overbrace{\mathbf{z}(k|k-1)}^{\text{predicted measurement}} \tag{12}$$

$$\mathtt{W} = \underbrace{\mathtt{P}(k|k-1)\mathtt{H}_x^\top \mathtt{S}^{-1}}_{\text{Kalman Gain}} \tag{13}$$

$$\mathtt{S} = \underbrace{\mathtt{H}_x \mathtt{P}(k|k-1)\mathtt{H}_x^\top + \mathtt{R}}_{\text{Innovation Covariance}} \tag{14}$$

Note that the number of observations in each timestep may vary. This is no problem for the EKF algorithm. Simply concatenate your observations and predictions into one long vector. Do the same for the observation Jacobian matrix `H` ensuring that the row ordering corresponds to the observation order you chose for the prediction and observation vectors. `R` is created as a block diagonal matrix with each block corresponding to each observation. In this assignment, use $\sigma_x = 2$ cm and $\sigma_z = 5$ cm for the x and z component (observed in the camera frame), respectively.

You will also be given the rigid body transformation $(R_{cb}, T_{cb}) \in \mathrm{SE}(2)$ required to transform a point from the camera reference frame to the robot body frame (the subscript cb means "from body b to camera c"). A 2D point $p^b = \left[p_x^b, p_y^b\right]^\top$ in the body frame is transformed to a 2D point $p^c = [p_x^c, p_z^c]^\top$ in the camera frame according to: $p^c = R_{cb}p^b + T_{cb}$. A similar transformation can be used to transfer between the robot body frame and the map frame if needed.

4.1 Configuration data

The configuration data is a Matlab file that contains the robot's initial pose and the location of the landmarks in the world frame. See the accompanying `README.txt` file for more details. Inside the Matlab file, there are:

- `camera_to_body`: the rigid-body transformation from the camera frame to the body frame. `R` and `T` represent rotation and translation, respectively.
- `landmark`: the (x, y) coordinates of the corners of each image in the world frame (in cm)
  - `xpos`: the x coordinates of the four corners (starting from the upperleft one and going clockwise); the i-th row corresponds to the i-th landmark, whose reference image is stored under the `landmark` folder;
  - `ypos`: the y coordinates of the corners
- `pose_init`: the initial pose and standard deviation for the pose, `x` and `y` are in centimeters, and `theta` is in degrees.
  - `x0, y0, theta0`: the initial pose
  - `sigma_x, sigma_y, sigma_theta`: the initial standard deviation for the pose (not the variance!)

4.2 Sensor data

The sensor data is provided in two formats: Matlab files and Python.

The Matlab data file contains a "`log`" variable, which is a structure array. `log(i)` contains the data at the i-th instant.

- The `timestamp` field contains the timestamp, in seconds, from the beginning of the log.
- The `commands` field contains the two commands corresponding to linear and angular velocity.
- The `rgb` field contains an RGB image.
- The `xyz` contains the stereo coordinates for each pixel in the image. **Note:** they are expressed in millimeters.[1]

The Python format contains the equivalent information in the form of a Numpy array.

5 Assignment

Create an EKF based localization system as described above. In your report:

1. [10 points] Show the equations you use for your (discrete-time) motion model and its Jacobians w.r.t the robot pose and the control input, as they would appear in Equations (3) and (4).
2. [10 points] Show the equations you use for your observation model and its Jacobian w.r.t the robot pose.
3. [25 points] Make a 2D plot showing the map, the dead reckoning trajectory, and the estimated trajectory using your localization system. Include 3σ covariance ellipses for the uncertainty in (x, y) position of the robot at several locations along this trajectory.

[1] They are represented in millimeters because then the field can be represented with integers rather than floats. This saves considerable space in the log file.