SLAM Summer School 2006

Practical 3: Introduction to the Information Filter

M. Walter and J. Leonard, MIT; R. Eustice, University of Michigan
mwalter@mit.edu, jleonard@mit.edu, eustice@umich.edu

1 Introduction

The information (canonical) form of the Gaussian has recently received a fair bit of attention as a possible alternative for the scalability problems of the EKF. This practical explores the use of the information filter (IF) for feature-based SLAM. The session will hopefully provide a basic understanding of localization and mapping with the information filter.

Today's lab considers two simulations of a robot operating in one-dimensional and two-dimensional worlds. In both cases, the vehicle moves according to a linear, constant-velocity model and makes relative observations of point features in the environment, both subject to Gaussian noise.

In addition to this explanation of the lab, we have included a brief introduction to the information form and its application to SLAM. The supplement addresses the basic steps of the IF including time projection, measurement updates, and the addition of new features to the map, i.e. the components that are fundamental to any SLAM algorithm. It might be useful to skim this over before going ahead with the lab since we will refer to it several times later on.

2 Setup

Obtain the files from the SSS06 CDROM or from the website at
`http://www.robots.ox.ac.uk/~SSS06/` and run matlab in the `code/` directory.

There are two main files in the directory: `slam_if_1d.m` performs the one-dimensional simulation while `slam_if_2d.m` implements the two-dimensional SLAM information filter.

3 Fundamentals of the Information Filter

As with the standard Kalman Filter, the marginalization and conditioning processes form the basis behind the time projection and measurement update steps. The one-dimensional simulation is meant to explore the basics of the SLAM information filter in terms of these two processes.

3.1 Starting the Simulation

Type `slam_if_1d` at the Matlab prompt to start the simulation. The program will create a window titled 'Figure 1' that contains two axes, the top a rendering of the environment and, below that, a schematic of the information matrix. Figure 1 below shows an example. Within the top plot, the red triangle denotes the estimated position of the robot, the black dots correspond to the ground-truth feature locations, and each red "x" indicates their estimated position. The rows and columns of the

information matrix are labeled with the corresponding state elements, $x_v(t)$ for the vehicle pose and id numbers for the features.

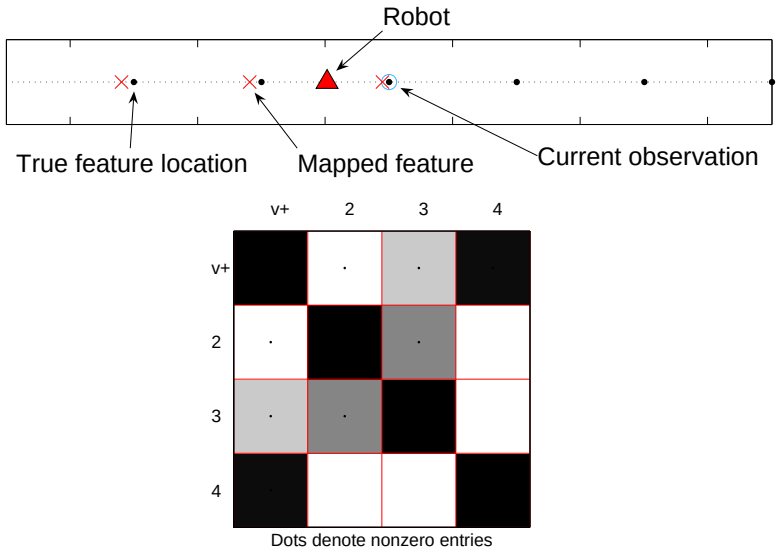

Figure 1: The main figure window for `slam_if_1d.m`.

The function pauses as it steps through the various key points in the IF algorithm. Hitting `Enter` causes the simulation to continue. It can be aborted at any time by hitting `Control-C`.

Note that the `slam_if_1d.m` file contains various parameters that control the simulation, most of which are fields of the global `Params` structure. Meanwhile, the majority of the filter variables are stored within `TheJournal`[1]. Feel free to change these settings to see how they affect the filter and have a look at the contents of `TheJournal`.

3.2 Displays

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

Figure 1 (top)	Simulation Plot	Shows the ground-truth feature locations (black dots) together with the current SLAM estimates for the vehicle (red triangle) and feature (red "x") positions.
Figure 1 (bottom)	Information Matrix	Depicts the structure of the information matrix. Shade intensity corresponds to the magnitude of the normalized matrix elements, white being zero and black being one.
Figure 2	Information Matrix	Schematic of the information matrix immediately prior to the current iteration.
Figure 3	Shared Information History	Plot of the magnitude of the shared information between each feature and the robot pose as a function of time.

[1] I think that I adopted this notation from Paul Newman.

3.3 Action Sequence

In the simulation, the robot starts at the origin and moves along the x-axis, observing features along the way. The program pauses each time the robot moves as the filter executes the time projection step as well as for each of the measurement updates. Each time you hit `Enter`, the simulation will carry out the next action, following the sequence:

1	Robot moves (a)	The new position of the vehicle, $x_v(t+1)$, is estimated from odometry, and added to the state vector. The state, $\mathbf{x} = [x_v(t+1)\ x_v(t)\ \mathbf{M}]$, includes the current and previous robot poses as well as the map.
2	Robot moves (b)	The time projection step concludes as the old robot pose, $x_v(t)$, is removed from the state vector.
3	Observations	The robot observes neighboring features as indicated by empty circles plotted around the true feature locations (black dots).
4	New features added	Any new features that are observed are added to the map.
5	Update	The filter performs an update step based upon measurements of the relative location of known landmarks.

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

Note that you can set the code to run through the simulation without these pauses by setting `Params.verbose` = 0 within `slam_if_1d.m`.

3.4 Things of Note

Let us take a closer look at the different action sequences of the simulation.

3.4.1 Time Projection Step

In order to get a better understanding of the time projection step as performed in the information form, we break it up into two processes. In the first, the new estimate for the robot pose, $x_v(t+1)$, is added to the state, together with the previous pose and map, based upon odometry information. Notice in 'Figure 1 (bottom)' that a row and column corresponding to the new pose have been added to the information matrix and that it shares information only with the old pose. Otherwise, the matrix remains unchanged as the sub-block associated with $x_v(t)$ and $\mathbf{M}$ is the same as prior to augmenting the state.

The information matrix and vector now describe the posterior distribution over the two poses and the map. The final component of the time projection step is to marginalize this distribution over the old pose, $x_v(t)$. Unlike state augmentation, this marginalization significantly alters the structure of the information matrix. To see this, compare the the new matrix to the form prior to marginalization (shown in 'Figure 2'). All shared information between the map and old pose has "moved" over to the new vehicle state in the form of off-diagonal elements in the first row and column. Also, this set of features is now fully-connected while many of them may not have been linked immediately before. At the same time though, a close inspection reveals that old constraints between the robot and map, which have been transferred to the new pose, are now a little weaker. What we see is consistent with Figure 2 of the supplementary notes in which the marginalization component of the time projection step populates the information matrix while also weakening many of its off-diagonal elements.

While links weaken as a result of the marginalization, they never decay to zero. Together with the constraints that the process creates among some of the features, the time projection step has the important

consequence of populating the information matrix. As the simulation progresses, the matrix appears to be sparse but, as we indicate with the dots, the it is actually filled-in.

3.4.2 Adding New Features and Updating the Filter

The robot measures the relative position to the point features on the x-axis. We build the map from new observations and update the filter with those of known landmarks[2]. New features are added according to the state augmentation process described in (2) of the notes, just like we do with new poses in the time projection step. Notice in 'Figure 1' that the new features share information only with the vehicle pose and not any other map elements.

In the case where the robot has observed a feature that is already in the map, the IF updates the information matrix and vector per (6) in the notes. The matrix is modified by adding information only to the elements associated with these features and the robot pose. This "new information" has the effect of strengthening the constraints between the robot and the map that have decayed as a consequence of the time projection step.

3.5 Things to Try

Several parameters that control the simulation are specified at the beginning of `slam_if_1d.m` as fields of the `Params` structure. These include those listed below. Try changing these to modify the IF behavior.

`Params.Q`	The variance of the Gaussian odometry noise.
`Params.R`	The variance of the noise that corrupts feature observations.
`Params.density`	Density of features in the environment.
`Params.nObsPerIteration`	Maximum number of features that the robot can observe at any one time.

3.6 Take-Home Message

One obvious benefit of the information filter is that, when the measurement model is linear, the update step is simple. The same update for the Kalman Filter is quadratic in the size of the state. On the other hand, the time projection step is computationally expensive for the IF but efficient for the KF. Intuitively, this is due to the fact that, in marginalizing over the old robot state, we have to transfer information to the new pose as well as create links among all landmarks that were linked to the old robot state. When there are a bounded number of constraints between the robot and map, this is *easy* but, as the map grows and the matrix becomes dense, this process is no longer tractable [2, 3].

4 Two-dimensional SLAM Simulation

The two-dimensional simulation is very much like the 1D simulation, though there are a few differences. For one, the default settings of the code create an environment with many more features and allow the

[2]Of course, we are assuming that we've solved the data association problem. While this is challenging for any SLAM algorithm, additional complications arise with the information form [1].

robot to observe multiple landmarks at once. Consequently, the information matrix will be larger and have a more complex structure. Secondly, the robot will follow a counter-clockwise path that results in a loop closure as the robot comes back to the (approximate) starting location. This point in the simulation is particularly interesting since you can see how the information matrix changes as the vehicle re-observes features that it hadn't seen since the beginning of the run.

4.1 Starting the Simulation

In order to start the simulation, type `slam_if_2d` at the Matlab prompt. A window, 'Figure 1', is created with a pair of axes side-by-side. The one on the left is a plot of the simulated environment while the schematic on the right depicts the information matrix. The legend is the same as in the 1D simulation with the addition of red three-sigma uncertainty ellipses around the position estimates. With regards to the plot of the information matrix, we visualize the determinant of each 2×2 sub-block (x and y) of the matrix.

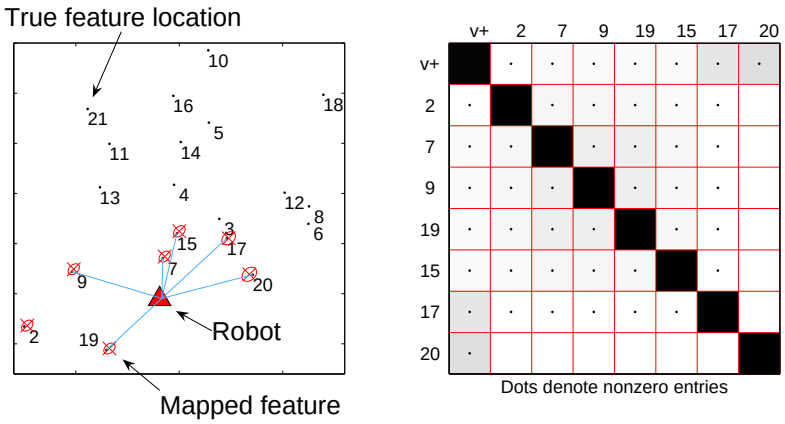

Figure 2: The main figure window for `slam_if_2d.m`.

4.2 Displays

Alongside the main Matlab window, additional displays will pop up, with the following roles:

4.3 Action Sequence

Prior to looping through the simulation, the main Matlab window asks whether or not you want verbose output. In verbose mode, the function will pause for the time prediction and measurement update steps as well as when new features are added. By turning this option off, the program will loop through the simulation without stopping.

The simulation iterates through the action sequences as described in the table below. The plots of the simulation and current information matrix in 'Figure 1' are updated and, in verbose mode, the previous information matrix is shown in 'Figure 2' for comparison. Hit `Enter` to continue or `Control-C` to abort the program.

Once the simulation has finished, 'Figure 3' shows the history of the shared information between the robot and the first few features that were added to the map. Unlike the 1D simulation where the vehicle

Figure 1 (left)	Simulation Plot	Display of the ground-truth feature locations (black dots) together with the current SLAM estimates for the vehicle (red triangle) and feature (red "x") positions. The red ellipses signify the 3σ uncertainty bounds.
Figure 1 (right)	Information Matrix	Depicts the structure of the information matrix. Shade intensity corresponds to the magnitude of the normalized matrix elements, white being zero and black being one.
Figure 2	Information Matrix	The window shows the form of the information matrix immediately prior to the current step in the filter.
Figure 3	Shared Information History	Plot of the determinant of the off-diagonal submatrix between the first few features and the robot pose as a function of time.

1	Robot moves	The new position of the vehicle, $x_v(t+1)$, is estimated from odometry. The filter performs time prediction as one single step.
3	Observations	The robot measures the relative (x, y) position of neighboring features as indicated by lines drawn between the vehicle and map.
4	New features added	Any new features that are observed are added to the map.
5	Update	The filter performs an update step based upon measurements of known landmarks.

didn't really close any loops, you may notice that there are points where the links strengthen over time. This is a result of loop closure.

4.4 Things to Try

Once the simulation has finished, take a closer look at the resulting information matrix for the map. If you type "`global TheJournal`" at the Matlab prompt, you will find that the information matrix is stored as `TheJournal.Lambda` while `TheJournal.eta` is the information vector. Note that you will probably have to type "`global TheJournal`" every time you run the simulation if you want to access global variables.

4.5 Matrix Density

The time history plots of the shared information for the 1D and 2D simulations show, to some extent, how the magnitude of the off-diagonal terms decays with time. In order to get a better idea of the distribution of values within the matrix, plot a histogram over the magnitudes. Matlab provides the `hist` function to compute histograms over a (optional) desired number of bins (type "`help hist`"). Rather than the original information matrix, though, look at the normalized matrix returned by a call to `rhomatrix`.

You should see a large disparity in the distribution with a significant majority of the terms nearly zero. What fraction of the total number of elements within the matrix falls within the left-most bins (see "`help numel`" and "`help nnz`")? If these terms were actually zero and did not need to be explicitly stored, how much memory would we save? Assume double precision (8 bytes) and that you

can ignore the overhead necessary to store sparse arrays. In our simulation, this improvement is minor since we are tracking only a few states. On the other hand, calculate how much memory would we save if we had 10,000 states[1] and an equivalent percentage of zeros in the information matrix?

4.6 Effects of Noise Strength

The extent of the noise that corrupts the motion and observation data is one factor that affects the strength and rate of decay of the shared information between the robot and map. How you think that less observation noise would affect the magnitude of the off-diagonal terms in the information matrix? How about a decrease in the motion model uncertainty? Edit `slam_if_2d.m` and modify the measurement and motion noise covariances, `Params.R` and `Params.Q`, and check to see if your intuition is correct. To speed up the simulation, you can turn off the plotting by setting `Params.PlotSwitch` = 0 and disabling verbose mode.

4.7 Computation Time

One advantage of the canonical form is that the number of computations in the measurement update step is proportional to the number of observations and not the size of the map. Contrast this with the KF, which is quadratic in the number of features. For the 2D code, we have tried to be efficient in the way that we store the information matrix as well as in our implementation of the measurement update step. Over the course of the simulation, the function has kept a history of the elapsed time for each time projection and measurement update. These are stored as `TheJournal.ProjectionTime` and `TheJournal.UpdateTime`, respectively, with one column for each record. The columns are of the form $[k\ n\ \Delta_t]^\top$ where k is the simulation time stamp, n the current size of the state vector, and Δ_t is the computational time required for the projection/update. Plot the computation times (third row) as a function of the number of states (second row).

4.8 Additional Simulation Parameters

There are a few other parameters defined as fields of the `Params` structure within `slam_if_2d.m`. These include `Params.density` and `Params.nObsPerIteration` that we defined earlier, as well as `Params.MaxObsRange`. If you have time, you can try changing these to see how they influence the information filter.

References

[1] R. Eustice, H. Singh, J. Leonard, M. Walter, and R. Ballard. Visually navigating the RMS Titanic with SLAM information filters. In *Proceedings of Robotics: Science and Systems (RSS)*, Cambridge, MA, USA, June 2005.

[2] S. Thrun, Y. Liu, D. Koller, A.Y. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *International Journal of Robotics Research*, 23(7-8):693–716, July-August 2004.

[3] M. Walter, R. Eustice, and J. Leonard. A provably consistent method for imposing exact sparsity in feature-based SLAM information filters. In *Proceedings of the 12th International Symposium of Robotics Research (ISRR)*, San Francisco, CA, October 2005.