DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY COLLEGE LONDON

COMP0130 Lab Topic 2, Part 1: Kalman Filters and SLAM

13th February 2022

Introduction

This workshop will introduce you to the basics of developing a SLAM system using a Kalman filter for a low-dimensional linear system. Although neither Kalman filters nor linear systems are state-of-the-art, the exercises will require you to look at issues of system design, dimensionality, and representing dependencies through cross-correlations.

1 Software

For this lab, you will be using vanilla Matlab.
Downloads the starter code from moodle. This is in the file `Workshop_04.zip`. Unzip it, open matlab and change into the directory where you unzipped it. You can then run the command:

```
>> setup
```

This sets up all the include paths. No other configuration is required. The model answers will be provided at the end of the lab.

The code in this part of the module is based off of a lightweight version of an event-based SLAM system. Although this looks like it's just obfuscating the algorithms and code at this stage, SLAM systems rapidly get bogged down in managing data and resources and it's better to have a framework setup at the start to work with. The framework also means it's possible to deal with multiple different data types with different sensor update rates.

The code should (hopefully) be commented but more guidance can be provided upon request.

System Description

The goal is to develop a localization system for a point-like robot called a dotbot. The dotbot is a particle which operates on a 2D plane. Its state only consists of its position and velocity in 2D,

$$\mathbf{x}_k = \begin{bmatrix} x_k & \dot{x}_k & y_k & \dot{y}_k \end{bmatrix}^\top. \tag{1}$$

The state of the jth landmark is its coordinates in 2D:

$$\mathbf{m}_j = \begin{bmatrix} u^j & v^j \end{bmatrix}. \tag{2}$$

The process model is a damped order system in which process noises are injected into the acceleration. The continuous time process model is

$$\dot{\mathbf{x}}_k = \begin{bmatrix} \dot{x}_k \\ \ddot{x}_k \\ \dot{y}_k \\ \ddot{y}_k \end{bmatrix} = \mathbf{F}_k^c \mathbf{x}_k + \mathbf{G}_k^c \mathbf{v}_k^c, \tag{3}$$

where

$$\mathbf{F}_k^c = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\alpha & -\beta & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\alpha & -\beta \end{bmatrix}, \ \mathbf{G}_k^c = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

and $\mathbf{v}_k^c$ is zero mean, Gaussian distributed with covariance $\mathbf{Q}_k^c$. Note that the Kalman filter we are using here will be implemented in discrete time. Therefore, we use van Loan's al-

gorithm, described here (see the section on "Discretization of the Process Noise") to compute the discrete-time equations. An implementation of this approach is provided in the file `continuousToDiscrete`.

The robot is equipped with two systems: a "GPS" which measures absolute position, and landmark detection system.

The "GPS" directly observes the dotbot position. It's observation model is

$$\mathbf{z}_k^G = \mathbf{H}_k^G \mathbf{s}_\mathbf{k}^\mathbf{G} + \mathbf{w}_k^G, \tag{5}$$

where

$$\mathbf{H}_k^G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{6}$$

and $\mathbf{w}_k^G$ is a zero-mean Gaussian distributed random variable with covariance $\mathbf{R}_k^G$.

The landmark sensor measures the relative translation between the dotbot and the landmark, in a world-fixed frame. If the jth detection at the kth timestep is the landmark i_k^j, the observation is given by

$$\mathbf{z}_k^j = \begin{bmatrix} u^{i_k^j} - x_k \\ v^{i_k^j} - x_k \end{bmatrix} + \mathbf{w}_k^j, \tag{7}$$

where $\mathbf{w}_k^j$ is a zero-mean Gaussian-distributed random variable.

2 Task 0: Run the Simulator and Check it Works

Run the script `task0.m`. Assuming everything is installed correctly, you should see a figure similar to that shown in Figure 1 which should update smoothly.

Task 1: Implement Basic Kalman Filter

Modify the Matlab class `DotBotSLAMSystem` to implement a Kalman filter which estimates the position and velocity of the robot from the GPS sensor. Run your code using the script `task1.m`.

Verify your result is similar to what is shown in Figure 2.

Hint: You will need to implement the discrete time process model. You might want to check the simulator to see how it's implemented there.

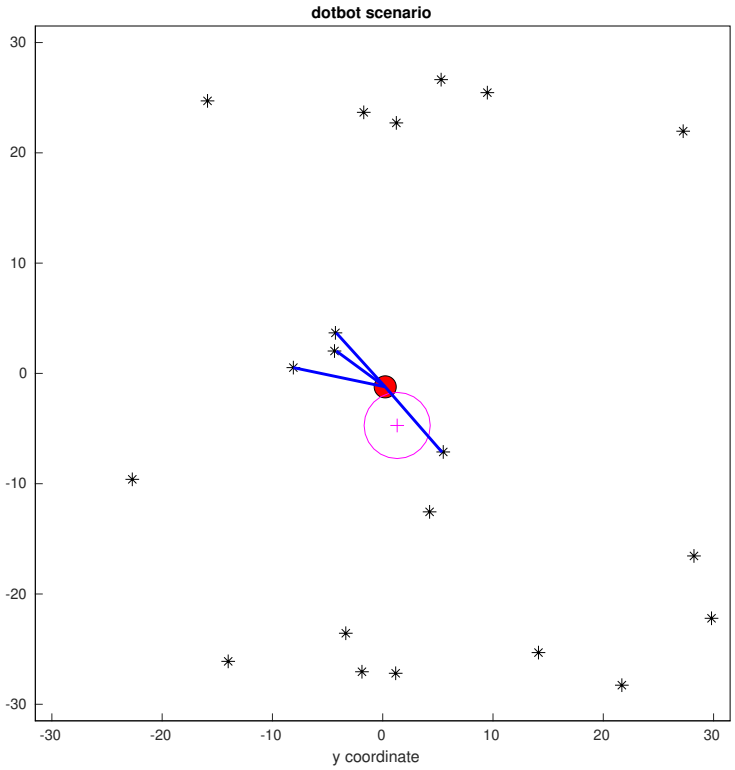

Figure 1: The dotbot simulation environment. The ground truth position of the dotbot is the red circle. Laser observations to landmarks are shown as blue lines. GPS measurements are purple crosses. The circles show the 1σ covariance ellipses of the observation noises. Here, the GPS measurement is fairly high, which is why the GPS measurement lies some distance from the dotbot platform.

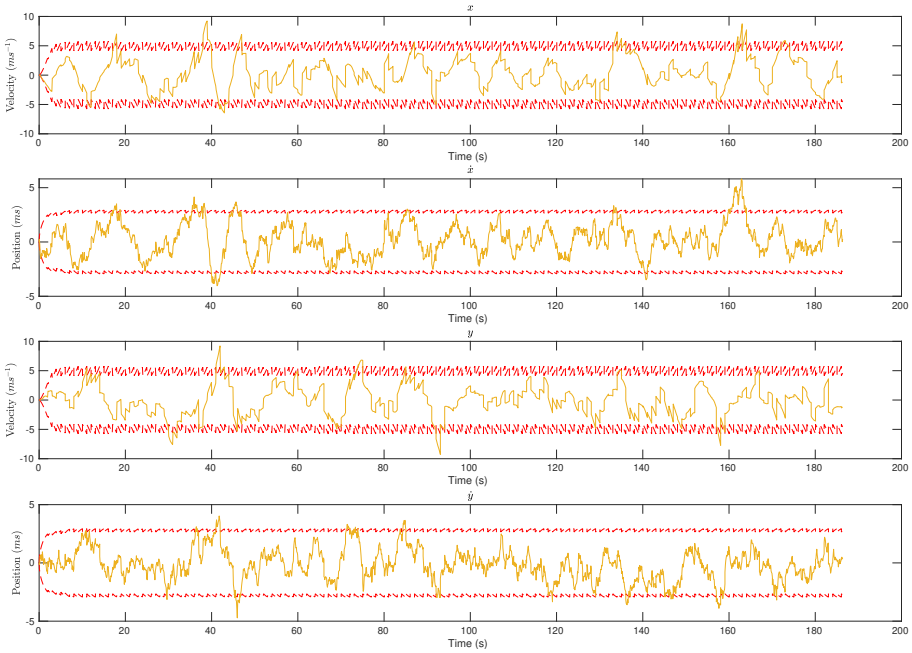

Figure 2: Sample output from task 1. This should look like a conventional Kalman filter with periodic updates from the GPS sensor.

Task 2: Implement Landmark Initialization Step

Modify the method `handleLandmarkObservationEvent` to initialize a new landmark if it has never been observed before. Use `task2.m` to test it. You should see something like Figure 3.

What do you think causes the landmark estimates to have differenc sizes?

If you want to try a more detailed analysis, graph the correlation coefficient of a landmark's x state with the dotbot x state over time and see how it fluctuates depending upon observation history.

Task 3: Update SLAM Landmark Estimates

Extend the method `handleLandmarkObservationEvent` to update the estimate of existing landmarks. Use `task3.m` to test it. You should see something like Figure 4.

If you want to try a more detailed analysis, modify `storeStepResults` to record the landmark mean and covariance histories over time.

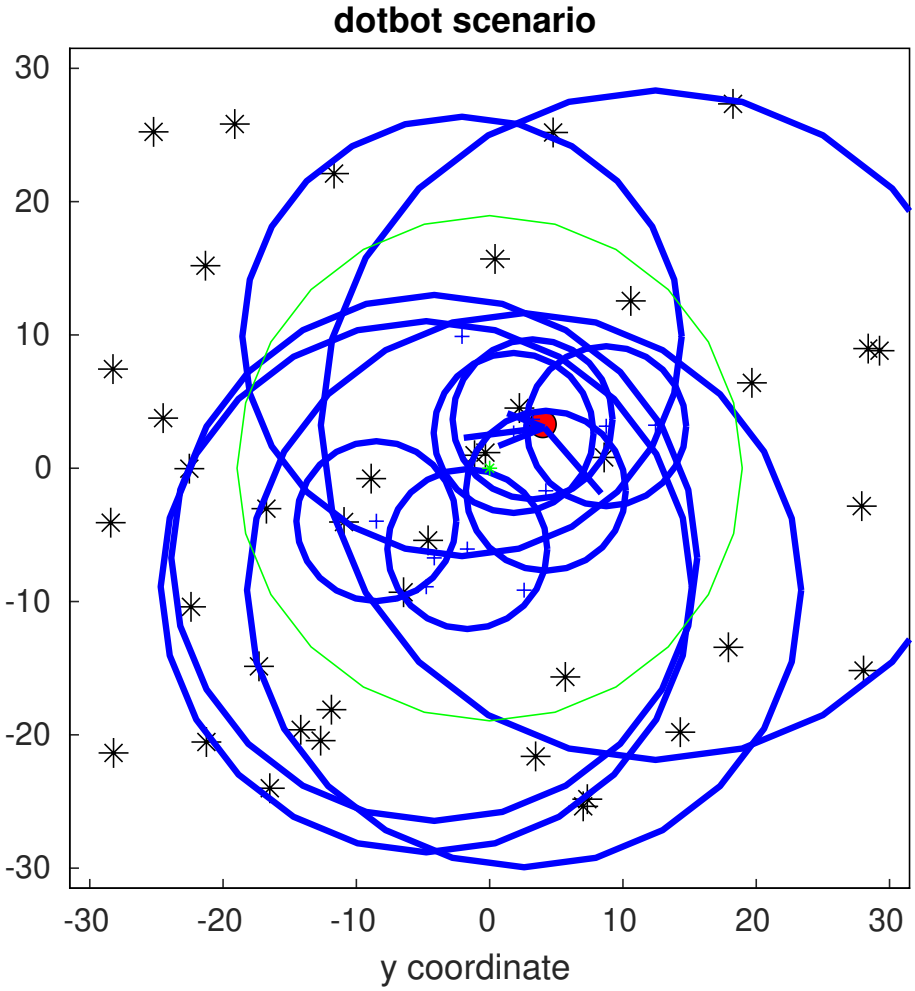

Figure 3: Sample output from task 2. The blue circles show the estimated landmarks, the green circle the dotbot estimate. You will probably see that the covariances are quite large and vary quite a lot.

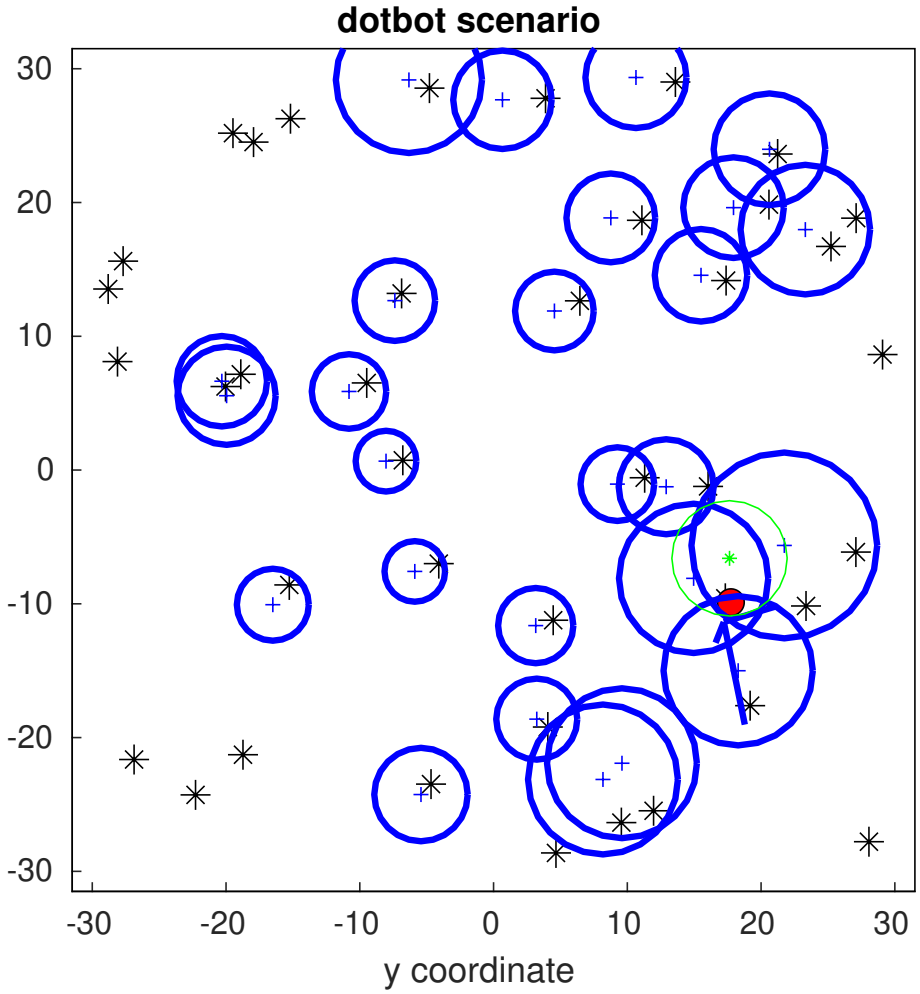

Figure 4: Sample output from task 3. The blue circles show the estimated landmarks, the green circle the dotbot estimate.

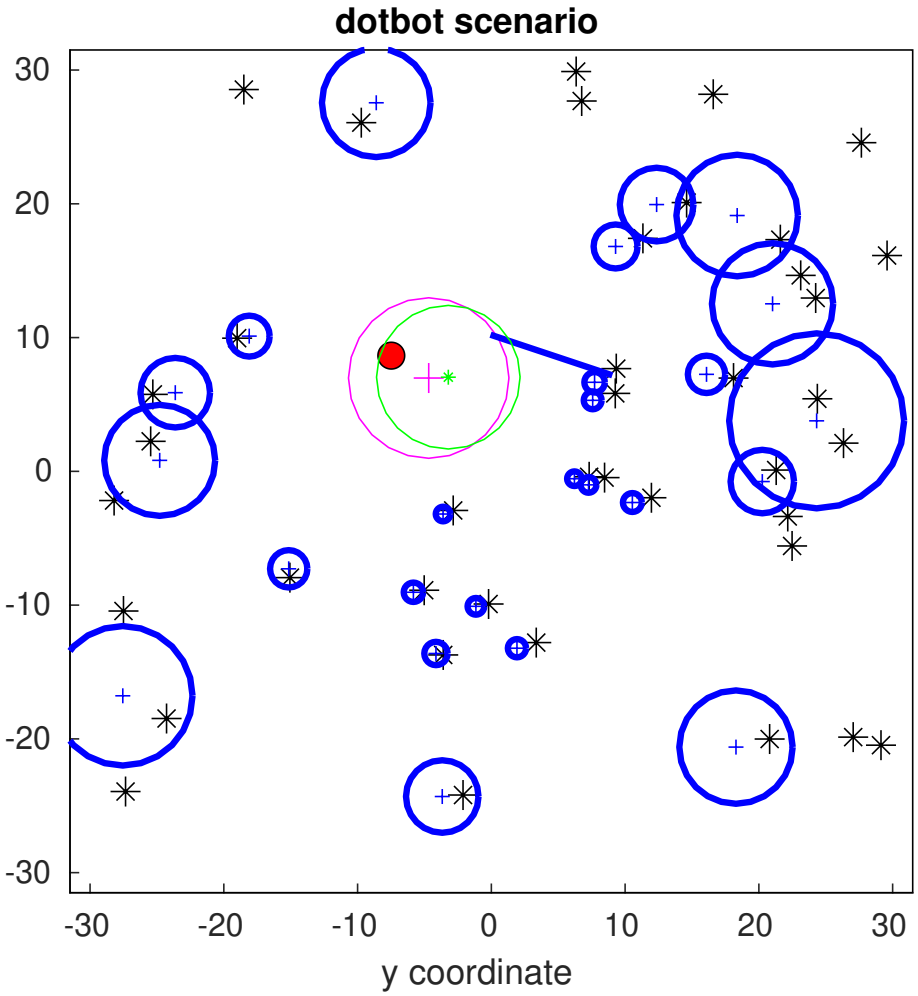

Figure 5: Sample output from task 4. The blue circles show the estimated landmarks, the green circle the dotbot estimate.

Task 4: Muck up the Cross Correlation Structure

Modify your code to corrupt the cross correlation structure and assess its impacts. There are several ways to do this. One simple idea is to zero out the vehicle-landmark cross correlations by inserting the code:

```
this.P(1:4, 5:end) = 0;
this.P(5: end, 1:4) = 0;
```

An example output is shown in Figure 5. Compare this with Figure 4 to see what the effects of incorrect correlations are.

Task 5: Play More

The behaviour of a SLAM system is the result of the interplay of several different factors. This task is open-ended: at this point, you should just play with different configurations and see what happens. Things you could test include:

1. Changing the process noise
2. Changing the observation noises.
3. Changing the GPS update frequency.
4. Changing the damping factors on the motion model.