Department of Computer Science, University College London

COMP0249 Lab 02: Implementing a Kalman Filter-Based SLAM System

24th January, 2025

1 Scope and Purpose

In this lab, you will implement a Kalman filter-based SLAM system. All the models here are linear; as we'll see in the lectures next week when systems become nonlinear problems start to arise.

As well as looking at the implementation of the SLAM system, we'll also see some additional functionality in the library. This includes being able to extract platform and landmark estimates separately, and a new class for collecting up and presenting results.

2 Scenario

The goal is to develop a SLAM system for dotbot, a point-like robot. This is different from pointbot: dotbot is controlled and is abole to move on a pre-programmed trajectory. As a result, it receives a control input which is its velocity at each timestep. The state only consists of the position,

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

More details are provided in the appendix.

3 Activities

Activity 0: Install the Software

The repository contains the current version of the software. Although most of the changes are restricted to the `Lab_02` folder, there were some changes to the `ebe`libraries which need to be incorporated as well. You can either update by executing `git pull` or downloading a zip file. Please be careful that the pull command might try to write over your local changes.

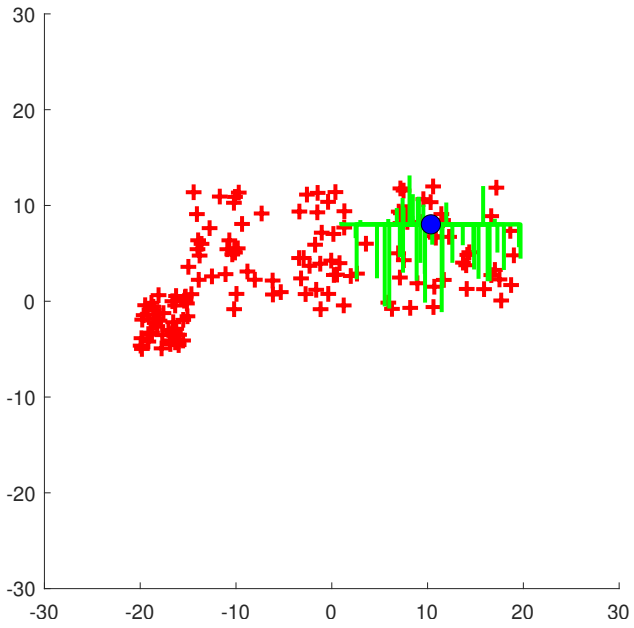

Figure 1: Simulator output from activity 0. Since the noise is randomly generated, you will probably see a slightly different view than this one.

To test if everything is installed, type:

```
>> l2.activity0
```

A window should open and you should see something like the image in Figure 1. The filled blue circle is the ground truth location of the particle position. The green lines are the measurements from the SLAM sensor, and the red crosses are the ground truth locations of the landmarks.

Activity 1: Investigating the Predictor Output

Run:

```
>> l1.activity1
```

This scenario shows the case where the robot moves in a straight line without any kind of inputs. What is the behaviour of the mean? What is the behaviour of the covariance? (Note it can take a number of seconds to become clear.)

Activity 2: Augmentation Step

In this activity, you are going to implement the augmentation step so you can start building a map.
The script to run is:
Run the script:

```
>> l2.activity2
```

You will need to modify the script `dotbot.SLAMSystem` **and, in particular the method** `handleSLAMObservationEvent`.

Hint: To aid with debugging, you might want to set the value of `perturbWithNoise` **in** `activity2-4.json` **to** `false`.

Activity 3: Update Step

In this activity, you are going to implement the update step so you can start building a map.

Run the script:

```
>> l2.activity3
```

You will need to modify the script `dotbot.SLAMSystem` **and, in particular the method** `handleSLAMObservationEvent`.

Activity 4: Mess Up the Cross Correlations

In the lectures, we mentioned that the cross correlations are very important. So, in this activity, we'll muck them up to see what happens.

Consider the following lines:

```
% ACTIVITY 4
if (obj.muckUp == true)
    for k = 1 : 2: length(obj.P)
        obj.P(k:k+1, 1:k-1) = 0;
        obj.P(k:k+1, k+2:end) = 0;
    end
end
```

What do you think this code does? Insert the implementation into `dotbot.SLAMSystem` **and run**

```
>> l2.activity4
```

What happens this time? Why do you think this happens?

A System Description

This appendix describes the process model and the observation models for dotbot. This is different from particlebot in Lab 01. Whereas particlebot's motion was purely random, dotbot now follows a path. Internally it is actually simulated using a robot which has both an orientation and a velocity. In this lab, however, we only look at the position components to make the system linear.

A.1 Process Model

The platform state space is

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \end{bmatrix}. \tag{2}$$

The dotbot controller periodically causes a change in velocity. These appear as a control input $\mathbf{u}_k$. Therefore, the dotbot state space model is

$$\mathbf{x}_{k+1} = \mathbf{F}_{k+1}\mathbf{x}_k + \mathbf{B}_{k+1}\mathbf{u}_{k+1} \tag{3}$$

where

$$\mathbf{F}_{k+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \ \mathbf{u}_{k+1} = \begin{pmatrix} \dot{x}_{k+1} \\ \dot{y}_{k+1} \end{pmatrix}, \ \mathbf{B}_{k+1} = \begin{bmatrix} \Delta T \\ \Delta T \end{bmatrix} \tag{4}$$

The SLAM system receives odometry measurements which are noise-corrupted versions of the velocity. The error in the velocity measurements are modelled as the process noise.

A.2 SLAM Sensor Observation Model

The SLAM sensor measures the Cartesian offset of the landmark from the robot. No actual sensor returns an observation like this, but it's convenient for developing the linear system here.

Suppose the system observes landmark i. The state of this landmark is $\mathbf{m}^i = (x^i, y^i)$. The observation model has the form

$$\mathbf{z}_k^i = \begin{pmatrix} x^i - x_k \\ y^i - y_k \end{pmatrix} + \mathbf{w}_k^i, \tag{5}$$

where the observation noise $\mathbf{w}_k^i$ is additive 2D Gaussian noise with covariance $\mathbf{R}_i k$.

A.3 GPS Sensors

An implementation has beenq provided, but have not been thoroughly checked for this lab. If you want to, you should be able to follow some of the procedures from Lab 01 to activate this sensor and see why, if you commercially use SLAM, people use GNSS systems wherever possible.