Department of Computer Science, University College London

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

24th January, 2025 ver. 20250127

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. As seen in the lectures, the SLAM system estimates the platform position and gradually estimates the map over time. At a time step k suppose N_k landmarks have been inserted into the map. The state space vector has the form

$$\hat{\mathbf{s}}_{k|k} = \begin{pmatrix} \hat{\mathbf{x}}_{k|k} \\ \hat{\mathbf{m}}^1_{k|k} \\ \vdots \\ \hat{\mathbf{m}}^{N_k}_{k|k} \end{pmatrix} \tag{1}$$

dotbot moves on a pre-programmed trajectory. The control inputs to dotbot are expressed as a control input on the velocity (see A.1). Therefore, the platform state space only consists of the position of the platform,

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

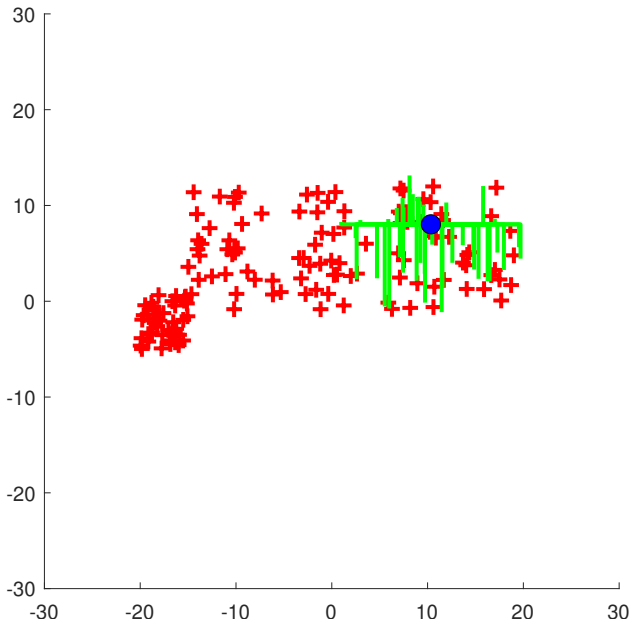

Figure 1: Simulator output from activity 0. Since the noise is randomly generated, you will probably see a slightly different view than this one.

The state of the ith landmark is its position,

$$\mathbf{m}^i = \begin{bmatrix} x^i \\ y^i \end{bmatrix}. \tag{3}$$

Further details are provided in Appendix A.

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.

To test if everything is installed, type:

```
>> l2v2.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:

```
>> l2v2.activity1
```

This scenario shows the case where the robot moves in a straight line without any observations. 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:

```
>> l2v2.activity2
```

The augmentation step was described in Slides 48–55 and 57–58 of the lectures for BeadSLAM. Appendix A.2 outlines the observation model for pointbot. The main difference over BeadSLAM is that the landmark is a 2D one instead of a 1D one.

To augment the map, you will need to modify the class `dotbot.SLAMSystem` **and, in particular the method** `handleSLAMObservationEvent`**. See lines 250–271. This part of** `handleSLAMObservationEvent` **loops over the observations of landmarks which are not in the map, and adds them one at a time.**

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 incrementally correct the platform and map position. The script is:

```
>> l2v2.activity3
```

The update was described in slides 77–78 of the lectures. The update is carried out using the standard Kalman filter equations.

You will need to modify the method `handleSLAMObservationEvent` **lines 235–242. This part of** `handleSLAMObservationEvent` **loops over the observations of landmarks which are present in the map, and updates them one at a time.**

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

```
>> l2v2.activity4
```

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

Activity 5: Optional Additional Investigations

We created a couple of slightly more substantial maps just to illustrate some behaviours. Run:

```
>> l2v2.activity5
```

You will be prompted to select `a`, `b`, `c` or `d`.
In brief:

- **`a`: Consists of landmarks in two isolated islands**
- **`b`: Conists of a set of landmarks forming a loop.**
- **`c`: This uses the same setup as `b`, but the GPS is enabled.**
- **`d`: This uses the same setup as `b`, but both the GPS and the bearing sensor are enabled.**

What behaviours do you notice in both cases? What do you think constitutes a loop closure? What impact do the non-SLAM sensors have on the behaviour of the system?

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

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \end{bmatrix}. \tag{4}$$

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

$$\mathbf{x}_{k+1} = \mathbf{F}_{k+1}\mathbf{x}_k + \mathbf{B}_{k+1}\mathbf{u}_{k+1} \tag{5}$$

where

$$\mathbf{F}_{k+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \ \mathbf{u}_{k+1} = \begin{pmatrix} \dot{x}_{k+1} \\ \dot{y}_{k+1} \end{pmatrix}, \ \mathbf{B}_{k+1} = \begin{bmatrix} \Delta T \\ \Delta T \end{bmatrix} \tag{6}$$

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.

Remember that the process model used with the full SLAM state must be augmented to include the landmark states. See slide 106 of the lectures for details.

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 observation model has the form

$$\mathbf{z}_k^i = \begin{pmatrix} z_k^{x,i} \\ z_k^{y,i} \end{pmatrix} = \mathbf{m}^i - \mathbf{x}_k + \mathbf{w}_k^i = \begin{pmatrix} x^i - x_k \\ y^i - y_k \end{pmatrix} + \mathbf{w}_k^i, \tag{7}$$

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

This can be written as the linear equation,

$$\mathbf{z}_k^i = \mathbf{H}_k^i \mathbf{x}_k + \mathbf{w}_k^i, \tag{8}$$

where

$$\mathbf{H}_k^i = \begin{bmatrix} -1 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}. \tag{9}$$

Note the identity block (values $10, 01$) for a beacon i lie in columns $2i + 1$ and $2i + 2$.

By rearranging (7), the landmark can be computed from the observation and platform estimate from,

$$\mathbf{m}^i = \mathbf{z}_k^i - \mathbf{x}_k + \mathbf{w}_k^i = \begin{pmatrix} z_k^{x,i} + x_k \\ z_k^{y,i} + y_k \end{pmatrix} + \mathbf{w}_k^i \tag{10}$$

A.3 GPS and Bearing Sensors

These sensors from Lab 01 have been updated to work with the SLAM system, and are used in Activities 5c and 5d. See Lab 01 for details on how they are implemented.