Department of Computer Science, University College London

# COMP0249 Lab 03: EKF-SLAM

31th January, 2025 ver. 20250131

## 1 Scope and Purpose

The goal is to

In this lab, you will complete the develop of an EKF-SLAM system for TriangleBot, which was described in Lecture 04. TriangleBot is a nonlinear system: the platform itself is oriented, and it measures range and bearing to the target.

Since in the last lab you worked on augmenting and updating landmarks and platform estimates, this lab focuses more on handling the nonlinear models. When we come to factor graphs, the complexities of the models are such that you will only focus on the models and graph architecture.

## 2 Scenario

The robot's state is now its position and its orientation,

$$\mathbf{x}_k = \begin{pmatrix} x_k \\ y_k \\ \theta_k \end{pmatrix}. \tag{1}$$

The state of the $i$th landmark is

$$\mathbf{m}^i = \begin{pmatrix} x^i \\ y^i \end{pmatrix}. \tag{2}$$

A full set of equations are described in Lecture 04, Slides 30–46.

## 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_03` folder, there were some changes to the `ebe`libraries which need to be incorporated as well.

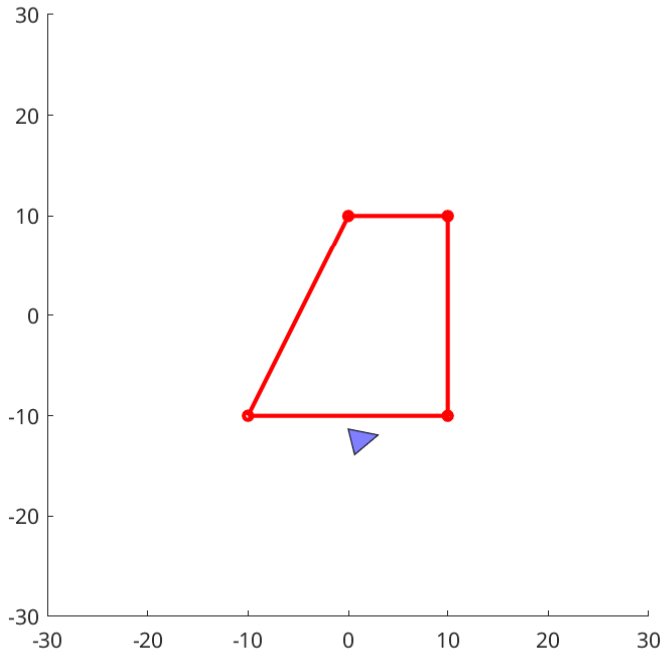


**Figure 1: Simulator output from activity 0. TriangleBot moves in an empty space. The filled red circles are waypoints the robot drives through. The solid lines shows how they connect. Note that TriangleBot uses a simple PID-based steering controller which means it will not, in general, follow the red line precisely.**

**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:**

```
>> l3.activity0
```

**A window should open and you should see something like the image in Figure 1. The setup consists of TriangleBot operating in an environment entirely devoid of any landmarks or other sensor observations.**

## Activity 1: Run the Open Loop Predictor

**In this activity, you are simply running the open loop predictor.**

```
>> l3.activity1
```

**You might need to change the size of the plotting axes (in `l3.activity1`) to see the behaviour more fully.**

**What happens to the vehicle covariance prediction and covariance in this case? Why do you think the covariance has the shape it has? How do you think orientation errors might be impacting the results?**

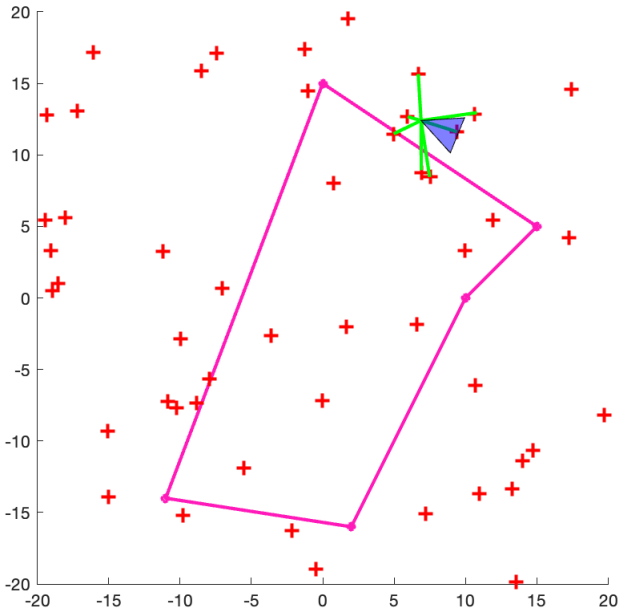


**Figure 2: Simulator output from activity 2.**

**Hint: Think what is the effect of an orientation error as the distance from the origin grows larger and larger.**

## Activity 2: Implement the Observation Model

**We are now going to start implementing the sensing models for the simulator and the EKF-SLAM system. To do this, you will need to work with the `l3.trianglebot.SystemModel` class. This class stores all the equations, which are used by both the simulator and the SLAM system. However, they are configured slightly differently for each: in the simulator, random noise is added in most cases.**

**Modify the method `predictSLAMObservation` to compute just the value of `z` (you will be asked later to finish the code to compute `gradHx`, `gradHm` and `gradHw`). Running:**

```
>> l3.activity2
```

**you should see output similar to that from Figure 2.**

## Activity 3: Implement Inverse Observation Model

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

```
>> l3.activity3
```

**You will need to finish implementing the method `predictLandmarkFromSLAMObservation`. You need to implement the code to compute the values of `mXY`, `gradGx` and `gradGw`. You should seen an output similar to what's in Figure 3.**

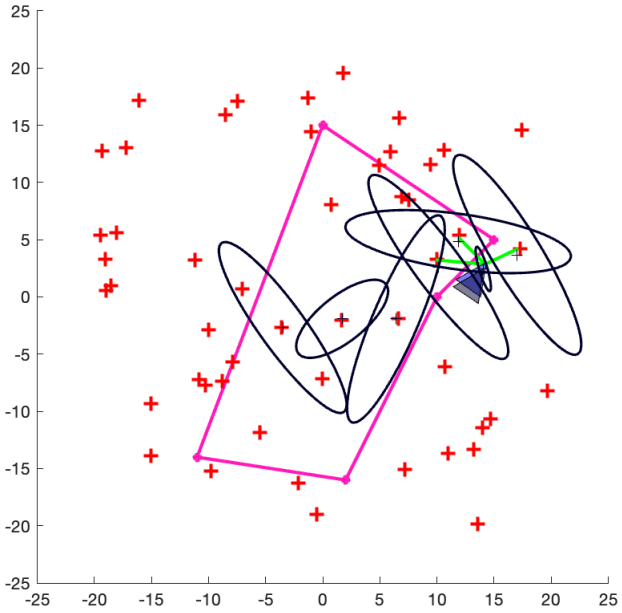


**Figure 3: Simulator output from activity 3. The landmarks are initialized, but they are not updated. As a result, the landmark estimates should not change.**

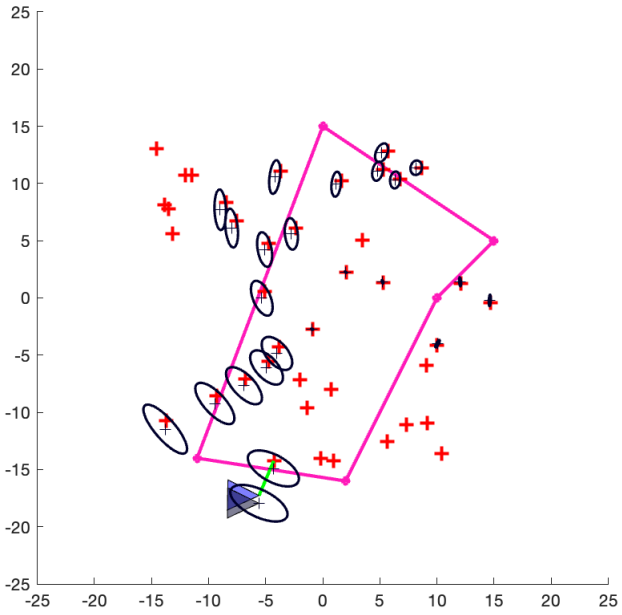


**Figure 4: The SLAM system output from activity 5.**

## Activity 4: Use of GPS and Compass with Initial Landmark Correlations

**The last activity got you initializing the landmarks in the correct place. In this activity we still don't do a full SLAM system. Rather, the idea is to give you a sense of what information is contained within the cross correlations which are computed.**

**For this activity, the platform has access to a GPS and a compass. These sensors only update once every `5 s`. code for this is already provided and you do not have to code anything. Run:**

```
>> l3.activity4
```

**What do you see happening in this case? What do you think the cross correlations might be doing?**

## Activity 5: Complete Implementation of the SLAM System

**In this activity, we'll go back to finishing up the full SLAM system without aid from GPS or compass. To do this, you need to complete the implementation of `gradHx`, `gradHm` and `gradHw` in `predictSLAMObservation`. See the lectures for the structure of the equations. Run:**

```
>> l3.activity5
```

**You should see something like the output in Figure 4. We have adjusted the sensor noises here to make EKF-SLAM behave better. However, recall that EKF-SLAM has issues with drift and errors, and so errors can arise.**

# 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

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \end{bmatrix}. \tag{3}$$

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

$$\mathbf{x}_{k+1} = \mathbf{F}_{k+1}\mathbf{x}_k + \mathbf{B}_{k+1}\mathbf{u}_{k+1} \tag{4}$$

where

$$\mathbf{F}_{k+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \ \mathbf{u}_{k+1} = \begin{pmatrix} \dot{x}_{k+1} \\ \dot{y}_{k+1} \end{pmatrix}, \ \mathbf{B}_{k+1} = \begin{bmatrix} \Delta T \\ \Delta T \end{bmatrix} \tag{5}$$

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

$$\mathbf{z}_k^i = \begin{pmatrix} z_k^{x,i} \\ z_k^{y,i} \end{pmatrix} = \mathbf{m}^i - \mathbf{x}_k + \mathbf{w}_k^i = \begin{pmatrix} x^i - x_k \\ y^i - y_k \end{pmatrix} + \mathbf{w}_k^i, \tag{6}$$

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,

$$\mathbf{z}_k^i = \mathbf{H}_k^i \mathbf{x}_k + \mathbf{w}_k^i, \tag{7}$$

where

$$\mathbf{H}_k^i = \begin{bmatrix} -1 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}. \tag{8}$$

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

By rearranging (6), the landmark can be computed from the observation and platform estimate from,

$$\mathbf{m}^i = \mathbf{z}_k^i - \mathbf{x}_k + \mathbf{w}_k^i = \begin{pmatrix} z_k^{x,i} + x_k \\ z_k^{y,i} + y_k \end{pmatrix} + \mathbf{w}_k^i \tag{9}$$

## 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.