Department of Computer Science, University College London

COMP0222 / COMP0249 Lab 03: EKF-SLAM

21th January, 2026 ver. 20260121

1 Scope and Purpose

In this lab, you will complete the develop of an EKF-SLAM system for TriangleBot, which was described in the lecture on EKF-SLAM. TriangleBot is a nonlinear system: the platform itself is oriented, and it measures range and bearing to the landmarks. The model reasonably closely resembles, for example, SLAM systems which are fitted on forklift trucks and cars.

Since in the last lab you worked on augmenting and updating landmarks and platform estimates, this lab focuses more on handling the nonlinear nature of the 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 ith landmark is

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

For this lab we do

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.

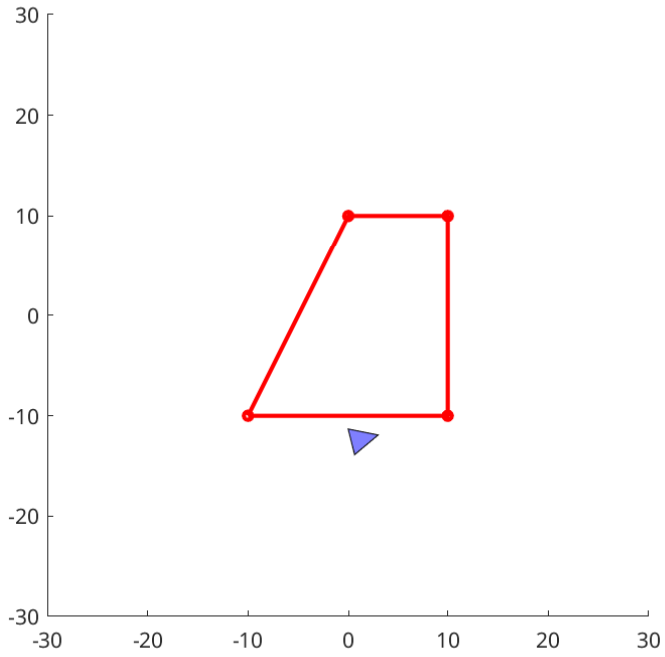

Figure 1: Simulator output from activity 0. TriangleBot moves in an empty space. The filled red circles are waypoints the robot drives through. The solid lines show how the waypoints connect with one another. Note that TriangleBot uses a simple PID-based steering controller which means it will not, in general, follow the red line precisely. If you have a go at authoring your own maps, it's possible to get the robot stuck circling around and around a waypoint.

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.

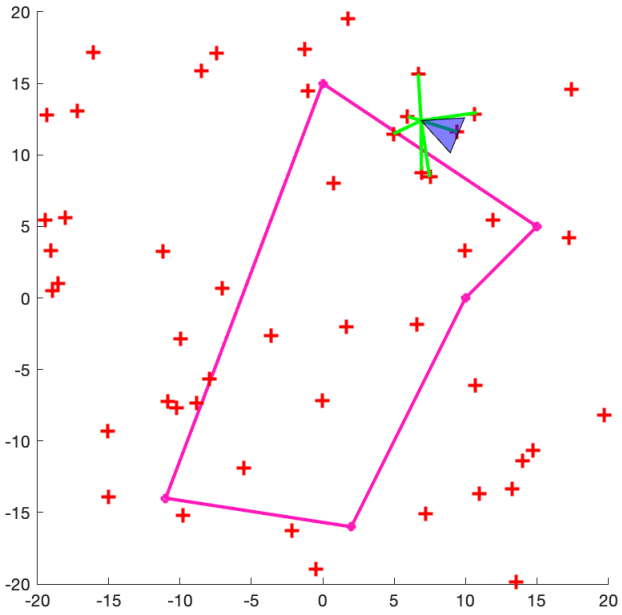

Figure 2: Simulator output from activity 2.

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?

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
```

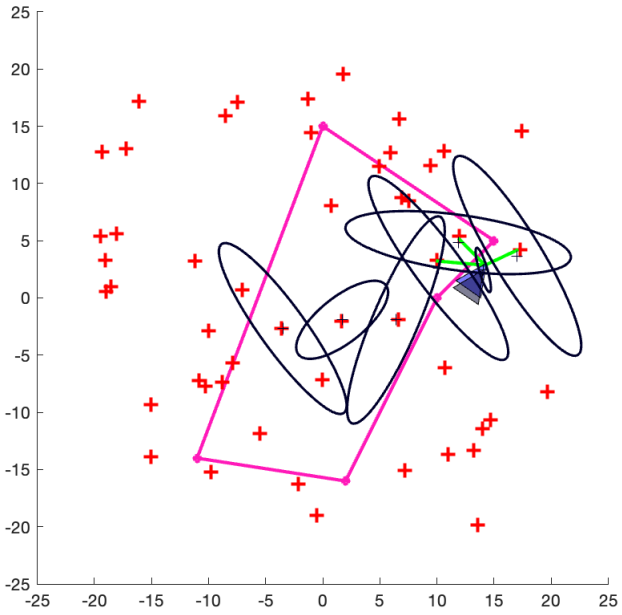

Figure 3: Simulator output from activity 3. The landmarks are initialized, but they are not updated. As a result, the landmark estimates should not change.

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.

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
```

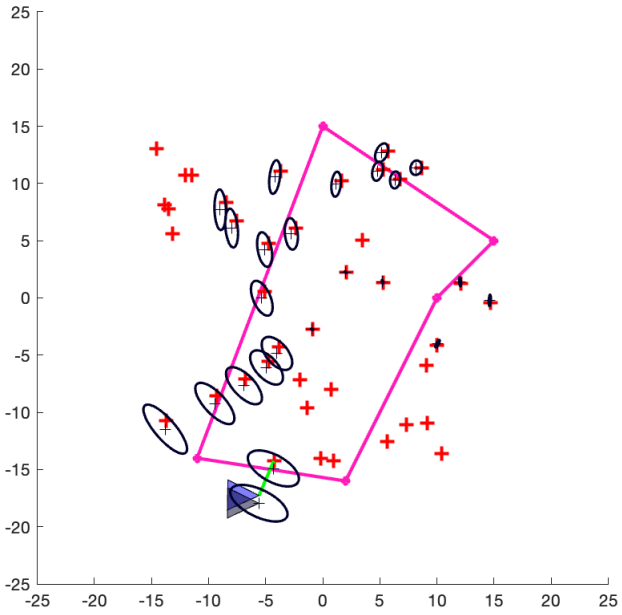

Figure 4: The SLAM system output from activity 5.

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 TriangleBot. The full equations are in the lectures slides; here we just provide the system equations and do not include the Jacobians.

A.1 Process Model

The platform state space is

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

The platform process model is the nonlinear equation

$$\mathbf{x}_{k+1} = \mathbf{f}\left[\mathbf{x}_k, \mathbf{u}_{k+1}, \mathbf{v}_{k+1}\right].$$

This has the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta T \mathbf{M}\left[\theta_k\right]\left(\mathbf{u}_{k+1} + \mathbf{v}_{k+1}\right).$$

The control input $\mathbf{u}_{k+1}$ is specified in the body-frame and consists of the linear speed in the x-direction and the angular velocity of the platform,

$$\mathbf{u}_{k+1} = \begin{bmatrix} s_{k+1} \\ 0 \\ \omega_{k+1} \end{bmatrix}$$

The process noise $\mathbf{v}_{k+1}$ is specified in the body-frame and consists of the velocity noise in the x, y directions as well as the angular velocity,[1]

$$\mathbf{v}_k = \begin{bmatrix} v^l_{k+1} \\ v^t_{k+1} \\ v^\omega_{k+1}. \end{bmatrix}$$

Finally, the matrix $\mathbf{M}[\theta_k]$ carries out the operation to project from platform coordinates to world coordinates,

$$\mathbf{M}[\theta_k] = \begin{bmatrix} \cos\theta_k & -\sin\theta_k & 0 \\ \sin\theta_k & \cos\theta_k & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

transform

TriangleBot uses a PID controller to generate the sequence for $\mathbf{u}_{k+1}$ automatically using an ordered list of waypoints. The process noise $\mathbf{v}_{k+1}$ is sampled randomly at the start of each prediction step and held constant.

A.2 SLAM Sensor Observation Model

The SLAM sensor measures the range and bearing from the centre of the platform to the landmark.

The equations are

$$r^j_k = \sqrt{\left(\Delta x^j_k\right)^2 + \left(\Delta y^j_k\right)^2} + \mathbf{w}^{r^j}_{k+1}$$
$$\beta^j_k = \arctan\left(\frac{\Delta y^j_k}{\Delta x^j_k}\right) - \theta_k + \mathbf{w}^{\beta^j}_{k+1}$$

where

$$\Delta x^j_k = x^{i_j} - x_k$$
$$\Delta y^j_k = y^{i_j} - y_k$$

A.3 GPS Sensors

The GPS sensor is a simple linear model which directly measures the position of the vehicle. Because it isn't a relative sensor, it's not strictly a "SLAM sensor". However, any real system for, say, mobile mapping will use GPS to supplement a SLAM system.

The observation equation is

$$\mathbf{z}^G_k = \begin{pmatrix} z^{G,x}_k \\ z^{G,y}_k \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} + \mathbf{w}^G_k, \tag{4}$$

[1] The notation is inconsistent with the rest of the process model. The superscripts l and t refer to longitudinal and lateral vectors — terminology from vehicle dynamics — which are parallel with the x and y vectors in the body-fixed frame.

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

However, note we do a bit of a fudge here. Angles are actually nonlinear manifolds — if you turn through 2π you end up back where you started from. The normal fix for this is to do "angle wrapping" which you can find details about on the web. An implementation is proviedd in `g2o.stuff.normalize_theta`, which is part of the github repo for this module.

A.4 Compass

The compass is very similar to the GPS: it is a direct measurement (this time of platform orientation) and is not a true "SLAM sensor". However, in practice compasses (or magnetometers) are less useful. The reason is that they can easily be disturbed by soft and hard iron in the environment and can easily produce errors in the range $5^\circ - 90^\circ$.

The observation equation is

$$\mathbf{z}_k^C = (\theta_k) + \mathbf{w}_k^C, \tag{5}$$

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

The same angle wrapping trick from the bearing measurement is also used.