# ENGNX627 Lab 3: Visual Odometry

## 1 Overview

This lab has two parts:

- Gather a dataset.
- Implement a visual odometry algorithm.

The first part should be completed quickly, since the second part is difficult, and will require that you work on it consistently over time. For this lab, you need to submit your report on wattle.

## 2 Instructions

### 2.1 Provided code

The code for the pibot simulator is hosted on the CECS's teaching gitlab here. Update your copy of the simulator by downloading it over the previous version, or by using `git pull`. If you are having trouble using GIT, have a look at the end of the lab 1 instructions, or watch some tutorial videos online.

For this lab, some skeleton code has been provided to help you create your solutions. You can download this in a folder called `lab3visualodometry.zip` from the course wattle page.

#### 2.1.1 Recommended Directory Layout

You should have a directory `ENGNX627_code` from the previous lab. The provided code is designed to work when put in the following directory structure.

```
ENGNX627_code
├── simulator
│   ├── piBotSim.m
│   └── ...
├── lab1kinematics
│   └── ...
├── lab2robotcontrol
│   └── ...
└── lab3visualodometry
    └── visual_odometry.m
```

## 2.2 Data Collection

When implementing complex algorithms such as visual odometry, we would like to be able to use quick and repeatable tests to evaluate any changes in the code. This is where datasets come in handy. In this context, a dataset is a collection of input data for an algorithm with timestamps. In the particular case of our visual odometry system, this means we need to record velocity inputs and landmark measurements with timestamps for both.

**Task** Write a script that records data from your robot for a set period of time. Note that, for the script to make sense, you will need to send velocity commands via the script as well. Ultimately, this data recording structure should sit on top of your line following algorithm from the previous lab. This will help you make the implementation and testing of visual odometry easier.

Here are some hints to get you on your way. The `tic` and `toc` commands can be used as follows.

```
start_time = tic;
elapsed_time = toc(start_time);
```

This is an easy way to measure the time from the start of your dataset and provides timestamps that are easy to understand. MATLAB has nice structures other than the standard arrays, such as the `cell` array. MATLAB also lets you define data structures on the fly. Other useful tools are the `save` and `load` commands Have a look here for more information:
cell arrays
structs
saving and loading

## 2.3 Visual Odometry

The second (and biggest) task of this lab is to implement and run the non-linear odometry observer discussed in the lectures. You need to implement the algorithm and then test it. To test the algorithm, you will have your robot follow some of the line paths from lab 2, and estimate its position using the odometry observer. Then you can compare this to the 'blind' odometry estimate (uncorrected kinematics integration) from lab 1.

You have been provided a script called `visual_odometry.m` to get you started. You must ensure your solution runs from this script. This script should both follow a line on the ground as in lab 2, as well as estimate the robot pose and landmark positions using the odomety observer discussed in the lectures.

**Task** Develop the script `visual_odometry.m` so that, when run, the robot follows a line on the ground and performs visual odometry to estimate its pose and the positions of landmarks over time.

**Task** Test the visual odometry observer by driving on some of the lines. You will need to tune the gain values in the algorithm to get good results.

Here are some hints.

- The first thing you should do is record some datasets of the robot driving the course and measuring landmarks. You could also consider recording some simple motion datasets (drive in a straight line / turn in place). If you write another script to use the datasets with your observer then debugging will be a lot easier. This will also help you tune your observer's gains.
- Try to create visualisations of your algorithm as soon as possible, since these will help in understanding how changes to your algorithm affect things. They will also be useful for your report!
- Before you start writing code, make a short plan of how you will implement the algorithm. You don't have to stick to this plan once you start coding, but it can help you foresee some challenges coming up.

## 3 What to hand in

You only need to submit a lab report addressing the **questions** below. The report must titled `u1234567_engnX627_Lab2report.pdf`, where `u1234567` is replaced with your university id, and must be in the `pdf` format. The answers to these questions in the report **must not exceed 5 pages in length.** Your lab report should include figures, tables, diagrams, and equations where they are helpful in answering questions. The lab report must **not** contain large blocks of code that you have copied from MATLAB. The page limit doesn't include the Extension work and Appendix, so you can add more figures and code snippets into your appendix.

**Question 1** Consider the provided line `floor_course.jpg`. Place the landmarks around the line so that they will all be seen by the robot as it drives along. Follow the line and collect a dataset.

**Part a.** Describe your dataset collection procedure. For example, you may describe; how you chose the timestamp for velocity commands and images? Did you decide to record any other data? etc.

**Part b.** Using the integrated robot position, and the measurements of the landmarks, generate a map of the trajectory and the landmarks. This map should be a plot featuring the robot's trajectory, and every measured landmark position. It should have a similar form to Figure 1, although the path driven by the robot is likely to be rather noisy, and the landmark positions may also be less consistent. Explain the sources of error in the map. Note that if there was no error, then all the landmark measurements should coincide exactly.

**Part c.** Finally, describe how the error changes for different landmark positions. Did some landmarks have very consistent estimates? Were there some landmarks with peculiar distributions? Why?

**Question 2** Place the landmarks in whatever positions you like, follow the line, and run the visual odometry algorithm.

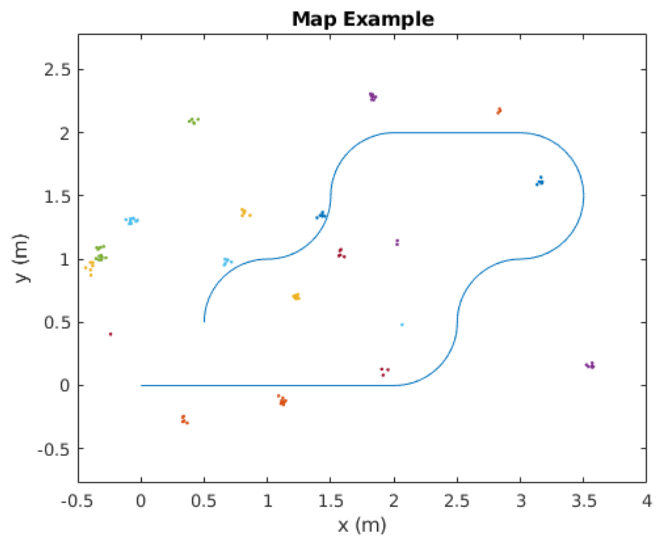

Figure 1: An example of what the map of points for question 1 might look like.

**Part a.** Briefly describe your visual odometry algorithm. What's the main steps in your algorithm? (Don't include too many equations here.)

**Part b.** Plot the computed robot trajectory and map.

**Part c.** Explain how you chose the gains of the algorithm $k_i$.

**Part d.** Explain why you placed the landmarks where you did. What placement of landmarks yields the best results in your implementation?

**Question 3** Repeat the procedure for quesiton 2, and record the landmark estimates over time.

**Part a.** Compare the estimated positions of the landmarks to the ground truth landmark positions and compute the Root Mean Square Error (RMSE):

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|\hat{p}_i - p_i\|^2},$$

where $\hat{p}_i$ and $p_i$ are the estimated and true positions of the $i^{\text{th}}$ landmark, respectively.

**Part b.** Also compute and compare the aligned RMSE as discussed in the lectures.

$$\mathrm{aRMSE} = \min_{S \in \mathbf{SE}(2)} \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|R_S^\top(\hat{p}_i - x_S) - p_i\|^2},$$

Discuss the advantages and disadvantages of RMSE and aligned RMSE. Give examples of where you would use one instead of the other.

**Extension work** Place the landmarks in a grid around the full space of the floor. Drive along the course line and map the landmarks using your visual odometry system. Now, drive your robot in any pattern you like around the room, and again use your visual odometry to construct a map of the landmarks. Find a trajectoy for the robot to drive that improves your map result as measured by aligned RMSE between the true and estimated map points. Discuss how you optimised the trajectory of your robot, and show the final result of your map.