Department of Computer Science, University College London

COMP0130 Coursework 02: Graph-based Optimisation and SLAM

February 21, 2020

1. Overview

- Assignment Release Date: Thursday 27th February, 2020
- Assignment Submission Date: 12:00 Monday 23rd March, 2020
- Weighting: 33% of module total
- Final Submission Format: a PDF file containing a report, and a zip file containing code.

2. Coursework Description

The goal of this coursework is to develop a graphical-model based SLAM system and assess its properties. The system will be implemented using the MATLAB g^{2}o port.
The scenario is shown in Figure 2.1. A wheeled robot drives through a two-dimensional environment which is populated by a set of landmarks. The vehicle is equipped with three types of sensors:

1. A vehicle odometry sensor, which records speed and heading.
2. A "GPS" sensor which measures vehicle position.
3. A 2D range bearing sensor which measures the range and bearing to the landmarks in 2D.

The mathematical models for these sensors are described in Appendix A.
We assume that data association has already been addressed. Therefore, each landmark measurement you receive will have a unique (and correct) landmark ID.
You will implement the system using the g^{2}o MATLAB implementation from Workshop 04, within an event-driven framework called MiniSLAM. The code for the event-driven framework is described in Appendix .
The coursework consists of four tasks:

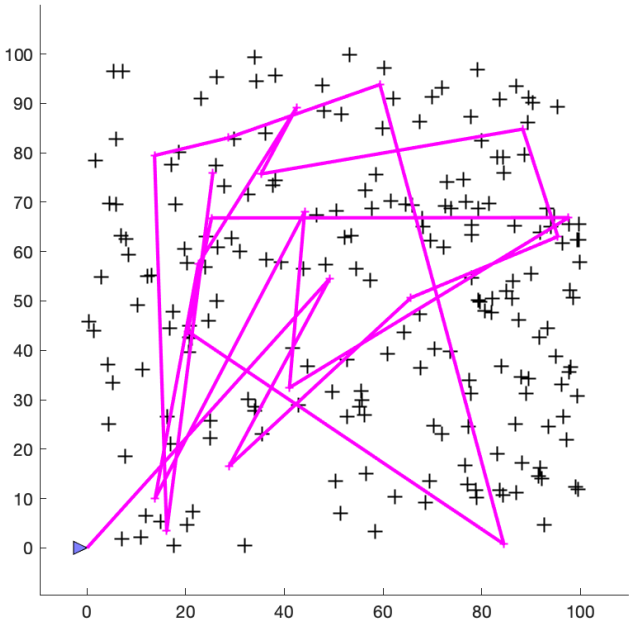

Figure 2.1: The scenario. The vehicle (small blue triangle) drives a path (purple line) through an environment populated by landmarks (blue crosses).

Task 1: Implement vehicle odometry in both a Kalman filter and the g^{2}o within MiniSLAM. Compare the performance of both. (10%)

Task 2: Extend your implementation to include the GPS for both the Kalman and g^{2}o systems. (15%)

Task 3: Extend your implementation with g^{2}o to implement SLAM and evaluate its properties. (40%)

Task 4: Explore a number of ways to improve the performance of the SLAM system (35%).

3. Task 1: Implement and Analyse Platform Prediction (10%)

You should use the script `task1TestPrediction` to launch the system.
For both the Kalman and g^{2}o SLAM systems, implement the code to predict the future state of the platform by implementing the methods `handleNoPrediction` and `handlePredictionToTime` in `+minislam.slam.g2o.G2OSLAMSystem` and the methods in `minislam.slam.g2o.VehicleKinematicsEdge`.
Describe your implementation and, when describing your g^{2}o implementation, provide a diagram to show the structure of the graph you create.
Compare the performance of the Kalman and g^{2}o systems. For your comparison, you should examin both the computed trajectories and the covariances. Explain the results you obtain.

4. Task 2: Implement and Analyse GPS Measurements (15%)

You should use the script `task2TestGPS` to launch the system.
Extend your implementations of the Kalman and the g^{2}o SLAM systems to use the GPS observations by implementing `handleGPSObservationEvent` and developing any edges that you need to support the new observation type. Describe your implementation and, when describing your g^{2}o implementation, provide a diagram to show the structure of the graph you create. Your diagram should show the case where the GPS observation events arrive less frequently than the odometry events.
Compare the performance of the Kalman and g^{2}o systems for GPS measurement periods (time between GPS measurements) of 1 s, 2 s and 5 s (you can set this by changing the value of `gpsMeasurementPeriod` in the simulation parameter object created in line 4 of `task2TestGPS`). For your comparison, you should compare both the computed trajectories and the covariances. Explain the results you obtain.

5. Task 3: Implement SLAM (40%)

Extend your implementation of the g^{2}o SLAM system to implement a full SLAM system by providing an implementation of the `handleLandmarkObservationEvent` method. You will need to create your own vertex type to represent the landmarks and any edges to support observations. Describe your implementation and, as before, show the structure of the graph you create. You should also provide information about the size of the graph. This includes the number of landmarks initialized, the number of vehicle poses stored, and the average number of observations made by a robot at each timestep, and the average number of observations received by a landmark.
In developing your system, two scripts are provided. The script `task3DevelopSLAM` runs the SLAM system on a small scenario of only a few landmarks. The script `task3TestSLAM` runs a more extensive example with 200 landmarks and 5000 timesteps.

6. Task 4: Improving Performance (35%)

This contains a number of sub-tasks to explore the performance further. The script `task4ReviseSLAM` as based on `task3TestSLAM`, but provides one or two extra parameters you might find useful.

6.1. Task 4.1: Adding GPS (5%)

One way to improve the performance of a SLAM system is to use additional measurements such as GPS. Experiment with different values of GPS measurement periods and see how the performance of the SLAM system changes. (You should find no changes should be required to your code.) You can start with the values in Task 2, but you are not restricted to those choices. Assuming you want to minimize the number of GPS measurements taken, what do you think would be optimal rate will be? Provide evidence of your choice.

6.2. Task 4.2: Removing Odometry (10%)

One way to improve the performance of a SLAM system is to use a keyframe based approach in which the vehicle prediction edges are not kept. Briefly describe how a keyframe-based SLAM

system differs from a regular SLAM system. Modify the g^{2}o SLAM system to delete the vehicle prediction edges before optimization and explain your implementation.
Compare two cases: when you delete all vehicle prediction edges, and all but the first vehicle prediction edges. By considering the structure of the graph, explain why you think the two behave differently.
*H*int: The algorithm will fail in one of these two cases.

6.3. Task 4.3: Graph Reduction (20%)

Two approaches for simplifying graphs have been proposed: sparse keyframe based SLAM, and graph pruning. Explain what these are.
Chose one strategy — either sparse key frames or graph pruning. Explain which one you chose and why. Modify the g^{2}o SLAM system to support your strategy and present the results. Your results should compare the size of the graph, its runtime, and performance as compared with a graph which contains no pruning.

A. System Models

A.1. State Description

The vehicle state is described by its position and orientation in 2D:

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \psi_k \end{bmatrix}. \tag{A.1}$$

A standard right handed coordinate system is used: x points to the right, y points up and a positive rotation is in an anticlockwise direction. Angles are in radians.
Landmarks are in 2D. The state of the ith landmark is given by

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

A.2. Process Model

The vehicle process model is the similar to the one you saw in Workshop 04. The model has the form,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta T_k \mathbf{M}_k \left(\mathbf{u}_k + \mathbf{v}_k\right). \tag{A.3}$$

The matrix

$$\mathbf{M}_k = \begin{bmatrix} \cos\psi_k & -\sin\psi_k & 0 \\ \sin\psi_k & \cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

is the rotation from the vehicle-fixed frame to the world-fixed frame and ΔT_k is the length of the prediction interval. Note that the prediction interval is governed by when various sensors are available, and can vary through the simulation.
The control input consists of the speed of the vehicle (which is oriented along a body-fixed x axis) together with an angular velocity term,

$$\mathbf{u}_k = \begin{bmatrix} s_k \\ 0 \\ \dot{\psi}_k \end{bmatrix}.$$

The process noise is zero mean, Gaussian and additive on all three dimensions,

$$\mathbf{v}_k = \begin{bmatrix} v_k \\ v_y \\ v_{\dot{\psi}} \end{bmatrix}.$$

The noise in the body-fixed y direction allows for slip and the fact, as discussed in the lectures, that the velocity is related to the front wheel and not the body orientation. The process noise covariance $\mathbf{Q}_k$ is diagonal.

A.3. Observation Models

The GPS sensor is a highly idealized one which provides direct measurements of the position of the robot. The observation model is

$$\mathbf{z}_k^{\mathrm{G}} = \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \mathbf{w}_k^{\mathrm{G}}$$

The covariance of $\mathbf{w}_k^{\mathrm{G}}$, $\mathbf{R}_k^{\mathrm{G}}$ is diagonal and constant.
The landmark observation model measures the range, azimuth and elevation of a landmark relative to the platform frame,

$$\mathbf{z}_k^{\mathrm{L}} = \begin{bmatrix} r_k^i \\ \beta_k^i \end{bmatrix} + \mathbf{w}_k^{\mathrm{L}},$$

where

$$r_k^i = \sqrt{(x^i - x_k)^2 + (y^i - y_k)^2}$$
$$\beta_k^i = \tan^{-1}\left(\frac{y^i - y_k}{x^i - x_k}\right) - \phi_k$$

The covariance of $\mathbf{w}_k^{\mathrm{L}}$, $\mathbf{R}_k^{\mathrm{L}}$ is diagonal and is assumed to be time invariant and the same for all landmarks.

B. MiniSLAM Framework

Modern robotic systems fuse data from a variety of sensing systems with different properties and update rates. You have already seen how these can be implemented in ROS. For this coursework, we introduce a small (self-contained) framework for event-driven estimation called MiniSLAM.
The code for MiniSLAM is provided in the `Coursework_02/+minislam` directory.
A set of scripts are provided for the different tasks which launch the MiniSLAM scenario.
MiniSLAM consists of the following:

1. A light weight event driven system to support multiple event types.
2. A scenario generator, which simulates or creates an environment.
3. A set of one or more SLAM systems which process this data to create results.

B.1. Event Driven System

Communication to the SLAM systems is provided by a set of events. Events are the analogy of ROS messages. The event types can be found in the `Coursework_02/+minislam/+event_types` subdirectory. The event types inherit from the `Event` class. Further documentation on the different events can be found in this class.

B.2. SLAM Systems

The main class which handles the processing the events is the `VehicleSLAMSystem` class. This provides logic to handle the different incoming event types. These are eventually passed onto a series of event handlers which carry out operations dependent upon the SLAM systems. The main ones you need to define are:

```
1         % Handle a GPS measurement
2         handleInitialConditionEvent(this, event);
3
4         % Handle not predicting
5         handleNoPrediction(this);
6
7         % Handle everything needed to predict to the current
             state.
8         handlePredictToTime(this, time);
9
10        % Handle a GPS measurement
11        handleGPSObservationEvent(this, event);
12
13        % Handle a set of laser measurements
14        handleLandmarkObservationEvent(this, event);
```