COMP0130: Robot Vision and Navigation
Coursework 2: Graph-based Optimisation and SLAM

Hyungjoo Kim	**Ahmed Salem**	**Zhonghao Wang**
170103694	20208009	20061720
ucabhki@ucl.ac.uk	zcemasa@ucl.ac.uk	ucabzw6@ucl.ac.uk

Introduction

The goal of this paper is to present different robot navigation approaches and assess their relative performance when provided with vehicle odometry information and following a code extension with GPS measurements. The two approaches that are presented are a Kalman filter approach and a graph-based g²o MATLAB approach. Following this, a full SLAM system is presented by extending the graph-based g²o approach to use landmark observation for navigation. Finally, three methods of improving the performance of the SLAM system are discussed and the results are presented.
The scenario provided is the following, a wheeled-robot vehicle is equipped with three types of sensors: a vehicle odometry sensor, a GPS sensor, and a 2-D range bearing sensor. The vehicle is set to drive in a 2-D environment populated by landmarks along a pre-defined path. Using the sensors provided and their associated noise and errors, the different navigation approaches are assessed against the pre-set true path. The presentation of this report was set to match the coursework's question structure.

Question 1. Vehicle Odometry

The state of the vehicle was described by the position and orientation in 2D as shown by (1) below [1], where x_k is the vehicle state vector at time k, and x_k, y_k and ψ_k are the x-position, y-position and heading respectively. A standard coordinate system is used where the positive: x points right, y points up and rotation is anticlockwise.

$$\mathrm{x}_k = \begin{bmatrix} x_k \\ y_k \\ \psi_k \end{bmatrix} \tag{1}$$

The vehicle process model was described by (2) [1] where ΔT_k is the length of the prediction interval, M_k is a rotation matrix shown in (3) [1] that describes the rotation from the vehicle-fixed frame to the world-fixed frame, u_k is the control input containing the speed and the angular velocity described in (4) [1], and v_k is the process noise assumed to be Gaussian, zero-mean and additive in all directions as described in (5) [1].

$$\mathrm{x}_{k+1} = \mathrm{x}_k + \Delta T_k \mathrm{M}_k(\mathrm{u}_k + \mathrm{v}_k) = f[\mathrm{x}_k, \mathrm{u}_k, \mathrm{v}_k] \tag{2}$$

$$\mathrm{M}_\mathrm{k} = \begin{bmatrix} \cos\psi_k & -\sin\psi_k & 0 \\ \sin\psi_k & \cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$\mathrm{u}_k = \begin{bmatrix} s_k \\ 0 \\ \dot{\psi}_k \end{bmatrix} \tag{4}$$

$$\mathrm{v}_k = \begin{bmatrix} v_k \\ v_y \\ v_{\dot{\psi}} \end{bmatrix} \tag{5}$$

Question 1a) **Kalman Filter implementation**

The simultaneous localisation and mapping (SLAM) of the Kalman Filter can be derived by four kinds of each step, such as initialisation, prediction, update, and augmentation. The process model of the Kalman filer is needed to predict the next step of the vehicle position and orientation. The process model needs a Jacobian function with respect to the state of the vehicle and the process noise to compute the next step of the state and covariance by equation (1-5). Therefore, the implementation of the Kalman filter can be represented as follow:

$$\mathrm{x}_{k+1} = \mathrm{x}_k + \Delta T_k \mathrm{M}_k(\mathrm{u}_k + \mathrm{v}_k) \tag{2}$$

, where the next step of the prediction vehicle state is x_{k+1} as equation (2).

$$\mathrm{x}_{k+1} = \mathrm{x}_k + \Delta T_k \begin{bmatrix} cos\psi_k & -sin\psi_k & 0 \\ sin\psi_k & cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{u}_1 + \mathrm{v}_1 \\ \mathrm{u}_2 + \mathrm{v}_2 \\ \mathrm{u}_3 + \mathrm{v}_3 \end{bmatrix} \tag{6}$$

$$\mathrm{x}_{k+1} = \mathrm{x}_k + \Delta T_k \begin{bmatrix} cos\psi_k(\mathrm{u}_1 + \mathrm{v}_1) - sin\psi_k(\mathrm{u}_2 + \mathrm{v}_2) \\ sin\psi_k(\mathrm{u}_1 + \mathrm{v}_1) + cos\psi_k(\mathrm{u}_2 + \mathrm{v}_2) \\ (\mathrm{u}_3 + \mathrm{v}_3) \end{bmatrix} \tag{7}$$

$$\mathrm{x}_{k+1} = \begin{bmatrix} \mathrm{x}_k + \Delta T_k cos\psi_k(\mathrm{u}_1 + \mathrm{v}_1) - \Delta T_k sin\psi_k(\mathrm{u}_2 + \mathrm{v}_2) \\ y_k + \Delta T_k sin\psi_k(\mathrm{u}_1 + \mathrm{v}_1) + \Delta T_k cos\psi_k(\mathrm{u}_2 + \mathrm{v}_2) \\ \psi_k + \Delta T_k(\mathrm{u}_3 + \mathrm{v}_3) \end{bmatrix} \tag{8}$$

The Jacobian function from the process model can be defined by the derivative of the prediction of the vehicle state (x_{k+1}) with respect to each parameter of the vehicle state (x_k) to compute the covariance of the process model [2].

$$J_{fx} = \frac{\partial \mathrm{x}_{k+1}}{\partial x} = \begin{bmatrix} \frac{\partial \mathrm{x}_{k+1,1}}{\partial x_k} & \frac{\partial \mathrm{x}_{k+1,1}}{\partial y_k} & \frac{\partial \mathrm{x}_{k+1,1}}{\partial \psi_k} \\ \frac{\partial \mathrm{x}_{k+1,2}}{\partial x_k} & \frac{\partial \mathrm{x}_{k+1,2}}{\partial y_k} & \frac{\partial \mathrm{x}_{k+1,2}}{\partial \psi_k} \\ \frac{\partial \mathrm{x}_{k+1,3}}{\partial x_k} & \frac{\partial \mathrm{x}_{k+1,3}}{\partial y_k} & \frac{\partial \mathrm{x}_{k+1,3}}{\partial \psi_k} \end{bmatrix} \tag{9}$$

$$\therefore J_{fx} = \begin{bmatrix} 1 & 0 & -\Delta T_k(\mathrm{u}_1 + \mathrm{v}_1)sin\psi_k - \Delta T_k \mathrm{v}_2 cos\psi_k \\ 0 & 1 & \Delta T_k(\mathrm{u}_1 + \mathrm{v}_1)cos\psi_k - \Delta T_k \mathrm{v}_2 sin\psi_k \\ 0 & 0 & 1 \end{bmatrix} = F_s \tag{10}$$

On the other hand, the Jacobian function from the process noise can be defined by derivative of the prediction of the vehicle state (x_{k+1}) with respect to the process noise (v_k) to compute the covariance of the process noise [2].

$$J_{vf} = \frac{\partial \mathrm{x}_{k+1}}{\partial v} = \begin{bmatrix} \frac{\partial \mathrm{x}_{k+1,1}}{\partial v_1} & \frac{\partial \mathrm{x}_{k+1,1}}{\partial v_2} & \frac{\partial \mathrm{x}_{k+1,1}}{\partial v_3} \\ \frac{\partial \mathrm{x}_{k+1,2}}{\partial v_1} & \frac{\partial \mathrm{x}_{k+1,2}}{\partial v_2} & \frac{\partial \mathrm{x}_{k+1,2}}{\partial v_3} \\ \frac{\partial \mathrm{x}_{k+1,3}}{\partial v_1} & \frac{\partial \mathrm{x}_{k+1,3}}{\partial v_2} & \frac{\partial \mathrm{x}_{k+1,3}}{\partial v_3} \end{bmatrix} = \begin{bmatrix} \Delta T_k cos\psi_k & -\Delta T_k sin\psi_k & 0 \\ \Delta T_k sin\psi_k & \Delta T_k cos\psi_k & 0 \\ 0 & 0 & \Delta T_k \end{bmatrix} \tag{11}$$

$$\therefore J_{vf} = \Delta T_k \begin{bmatrix} cos\psi_k & -sin\psi_k & 0 \\ sin\psi_k & cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix} = B_s \tag{12}$$

From now, the Kalman Filter covariance of the process model can be computed using the Jacobian functions from the prediction model (F_s) and the process noise (B_s), respectively. Furthermore, the Jacobian functions should be same dimension of the covariance prediction, which is a 3 by 3 matrix. Therefore, the covariance prediction of the Kalman filter can be computed as follow [3]:

$$P_{2|1} = F_s P_{1|1} {F_s}^T + B_s Q_s {B_s}^T \tag{13}$$

This principle of the Kalman Filter is explained more detailed, which express the order for computing the prediction model by algorithm 1.

Algorithm 1 handlePredictToTime - Kalman Filter system [1] [2] [3]

Input: Initial Kalman filter mean (xEst, PEst), Initial Kalman filter covariance (xPred, PPred), ΔT_k, Current Time

1: $\mathrm{M_k} \leftarrow \begin{bmatrix} \cos\psi_k & -\sin\psi_k & 0 \\ \sin\psi_k & \cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix}$

2: $\mathrm{v}_k \leftarrow \sqrt{Odometry\ Covariance\ Matrix} * random(3,1)$: Used to generate random additive Gaussian noise (v_k).

3: $Q_s \leftarrow diag(\mathrm{v_k}^2)$: Make the same dimension as the covariance matrix

4: $xPred = xEst + \Delta T_k \mathrm{M}_k(\mathrm{u}_k + \mathrm{v}_k)$: Compute the Kalman filter prediction of the state

5: $J_{fx} \leftarrow \begin{bmatrix} 1 & 0 & -\Delta T_k(\mathrm{u}_1 + \mathrm{v}_1)sin\psi_k - \Delta T_k \mathrm{v}_2 cos\psi_k \\ 0 & 1 & \Delta T_k(\mathrm{u}_1 + \mathrm{v}_1)cos\psi_k - \Delta T_k \mathrm{v}_2 sin\psi_k \\ 0 & 0 & 1 \end{bmatrix}$: Compute the Jacobian from the prediction model

6: $J_{vf} \leftarrow \Delta T_k \begin{bmatrix} cos\psi_k & -sin\psi_k & 0 \\ sin\psi_k & cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix}$: Compute the Jacobian from the process noise

7: $F_s, B_s \leftarrow J_{fx}, J_{vf}$

8: $PPred = F_s PEst {F_s}^T + B_s Q_s {B_s}^T$: Compute the covariance prediction

9: Update the Kalman filter mean

10: $xEst \leftarrow xPred$

11: $PEst \leftarrow PPred$

Output: $xEst, PEst$

Question 1b) **g2o MATLAB implementation**

A factor graph can predict the future state of a vehicle in linear or non-linear systems by having a scalable way of storing the probabilities for the process model and the observation model. The vehicle is initialised by setting the starting position to the origin as performed in the Kalman Filter. The next step involves using the process model which uses $\mathbf{x}_{k-1}, \mathbf{u}_k$, and $\mathbf{v}_k$ to generate a conditional probability prediction $\hat{\mathbf{x}}_{k|k-1}$ (notation describes the prediction of the state vector at the current time k, given the state vector at time $k-1$) [3]. Again, the notation for the process model is described by (2). This process can be repeated for all timesteps if a control input is provided.

Factor graphs store these probabilities into vertices, which describe the state of an object of interest, and edges, which specify the conditional probabilities between objects of interest [3]. Edges may be unary or binary depending on the number of vertices that it connects to [2]. Table 1 details the objects, classes and how they were modelled in the g²o MATLAB implementation. Note that some objects which are included in this table are not used for this question but are described later in the report.

Table 1: Details of the objects and classes assigned for the scenario.

Object	Class Name	Class Type
Robot (vehicle) pose	VehicleStateVertex	Base Vertex
Vehicle process model	VehicleKinematicsEdge	Binary Edge
Initial Position	InitialPriorEdge	Unary Edge
GPS measurements	GPSMeasurementEdge	Unary Edge
Landmarks	LandmarkVertex	Base Vertex
Landmark Measurement	LandmarkMeasurementEdge	Binary Edge

Figure 1 shows the graphical representation of the Bayes filter (probabilities) for this section. It can be seen that vertices are modelled as white circles, and edges are modelled as black squares. The vertex $\mathbf{x_0}$ is

initialised using the InitialPriorEdge class and the original state is set to the origin as aforementioned. A new vertex x_k is then created with the location described by the process model. A binary edge of class, VehicleKinematicsEdge, is created and connects the two vertices together. This process repeats itself until the end of the specified odometry and is further explained by algorithm 2.

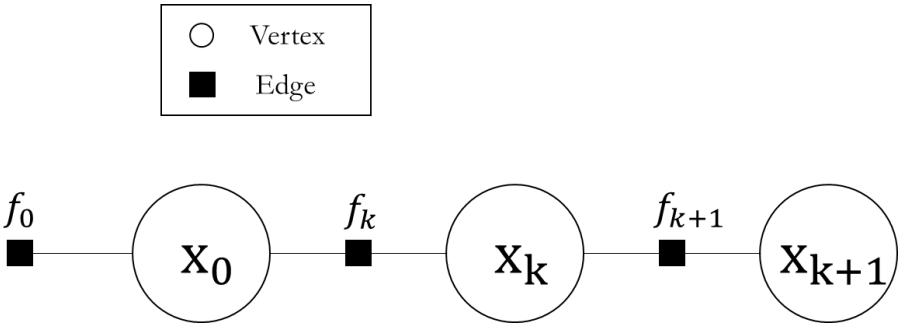


Figure 1: A visual representation of the graphical model used for this section.

To understand algorithm 2, it is essential to understand the structure of the g²o system. The simulation/scenario is produced via events of different types and the definition of the vertices, edges and SLAM systems enables the production of results. Depending on the event type, different functions are called which makes for a simpler framework to produce results as parameters such as the environment, sensor types, etc., can be changed easily. Algorithm 1 describes how the current state is predicted when a vehicle odometry event type is called.

Algorithm 2 handlePredictToTime - g²o system

Input: Current Vehicle Vertex, Optimisation Graph, ΔT_k , Current Time, Odometry, Odometry Covariance

1: newVehicleVertex = VehicleStateVertex(): Create a new vehicle vertex

2: Graph.addVertex(newVehicleVertex): Add vertex to the optimisation graph

3: $\mathrm{v}_k \leftarrow \sqrt{Odometry\ Covariance\ Matrix} * random(3,1)$: Used to generate random additive Gaussian noise (v_k).

4: OmegaQ = inverse(Odometry Covariance Matrix): Defining the Hessian matrix

5: $\mathrm{M}_\mathrm{k} \leftarrow \begin{bmatrix} \cos\psi_k & -\sin\psi_k & 0 \\ \sin\psi_k & \cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix}$

6: $\mathrm{x}_{k+1} \leftarrow \mathrm{x}_k + \Delta T_k \mathrm{M}_k(\mathrm{u}_k + \mathrm{v}_k)$: compute new vertex location using process model

7: Normalise ψ_k from x_{k+1} to $(-\pi{:}\,\pi)$

8: newEdge = VehicleKinematicsEdge(dT): create new binary edge

9: newEdge.setVertex(1, currentVehicleVertex): set the edge to connect to the current vehicle vertex

10: newEdge.setVertex(2, newVehicleVertex): set the edge to connect to the new vehicle vertex

11: newEdge.setMeasurement(odometry): Add odometry measurement to edge

12: newEdge.setInformation: setting information matrix omega (Hessian) to edge

13: Graph.addEdge(newEdge): Add the new edge to the graph

Output: None

Furthermore, the structure of the g²o system needs to compute the error, which is generated by the vehicle kinematics [2]. The error always occurs in the real world, and the engineer has the challenge of identifying the error from the output of the sensors. The system's posterior process model does not occur any noise because of the initialisation, and the posterior model can be computed as equation (2).

After defining the posterior model, the initial process noise (v_k) can be computed as follows:

$$\mathrm{M}_k{}^{-1}\,\frac{(\mathrm{x}_{k+1}-\mathrm{x}_k)}{\Delta T_k} = (\mathrm{u}_k + \mathrm{v}_k) \tag{14}$$

$$\therefore \mathrm{v}_k = \mathrm{M}_k{}^{-1}\,\frac{(\mathrm{x}_{k+1}-\mathrm{x}_k)}{\Delta T_k} \tag{15}$$

The v_k is the process noise as equation (5) and the initial of the control input (u_k) is zero because the speed and angular velocity of the vehicle is zero, so the error for the next step can be presented as the difference between the process noise and the control input [2].

$$e_k = \mathrm{M}_k{}^{-1}\,\frac{(\mathrm{x}_{k+1}-\mathrm{x}_k)}{\Delta T_k} - \mathrm{u}_k \tag{16}$$

$$e_k = \begin{bmatrix} cos\psi_k & sin\psi_k & 0 \\ -sin\psi_k & cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix} \frac{(\mathrm{x}_{k+1}-\mathrm{x}_k)}{\Delta T_k} - \begin{bmatrix} s_k \\ 0 \\ \dot{\psi}_k \end{bmatrix} \tag{17}$$

$$e_k = \begin{bmatrix} cos\psi_k \frac{(\mathrm{x}_{k+1}-\mathrm{x}_k)}{\Delta T_k} + sin\psi_k \frac{(y_{k+1}-y_k)}{\Delta T_k} - s_k \\ -sin\psi_k \frac{(\mathrm{x}_{k+1}-\mathrm{x}_k)}{\Delta T_k} + cos\psi_k \frac{(y_{k+1}-y_k)}{\Delta T_k} \\ \frac{(\psi_{k+1}-\psi_k)}{\Delta T_k} - \dot{\psi}_k \end{bmatrix} \tag{18}$$

The current and prediction of the Jacobian function can be defined by a derivative of the error (e_k) with respect to the current vehicle state (x_k) and the prediction vehicle state (x_{k+1}) and are represented as follow:

$$J_k = \frac{\partial e_k}{\partial \mathrm{x}_k} = \begin{bmatrix} \frac{\partial e_{k,1}}{\partial \mathrm{x}_k} & \frac{\partial e_{k,1}}{\partial \mathrm{y}_k} & \frac{\partial e_{k,1}}{\partial \psi_k} \\ \frac{\partial e_{k,2}}{\partial \mathrm{x}_k} & \frac{\partial e_{k,2}}{\partial \mathrm{y}_k} & \frac{\partial e_{k,2}}{\partial \psi_k} \\ \frac{\partial e_{k,3}}{\partial \mathrm{x}_k} & \frac{\partial e_{k,3}}{\partial \mathrm{y}_k} & \frac{\partial e_{k,3}}{\partial \psi_k} \end{bmatrix} = \begin{bmatrix} -\frac{cos\psi_k}{\Delta T_k} & -\frac{sin\psi_k}{\Delta T_k} & -sin\psi_k \frac{(\mathrm{x}_{k+1}-\mathrm{x}_k)}{\Delta T_k} + cos\psi_k \frac{(y_{k+1}-y_k)}{\Delta T_k} \\ \frac{sin\psi_k}{\Delta T_k} & -\frac{cos\psi_k}{\Delta T_k} & -cos\psi_k \frac{(\mathrm{x}_{k+1}-\mathrm{x}_k)}{\Delta T_k} - sin\psi_k \frac{(y_{k+1}-y_k)}{\Delta T_k} \\ 0 & 0 & -\frac{1}{\Delta T_k} \end{bmatrix} \tag{19}$$

$$J_k = \frac{1}{\Delta T_k} \begin{bmatrix} -cos\psi_k & -sin\psi_k & -sin\psi_k(\mathrm{x}_{k+1} - \mathrm{x}_k) + cos\psi_k(y_{k+1} - y_k) \\ sin\psi_k & -cos\psi_k & -cos\psi_k(\mathrm{x}_{k+1} - \mathrm{x}_k) - sin\psi_k(y_{k+1} - y_k \\ 0 & 0 & -1 \end{bmatrix} \tag{20}$$

$$J_{k+1} = \frac{\partial e_k}{\partial \mathrm{x}_{k+1}} = \begin{bmatrix} \frac{\partial e_{k,1}}{\partial \mathrm{x}_{k+1}} & \frac{\partial e_{k,1}}{\partial \mathrm{y}_{k+1}} & \frac{\partial e_{k,1}}{\partial \psi_{k+1}} \\ \frac{\partial e_{k,2}}{\partial \mathrm{x}_{k+1}} & \frac{\partial e_{k,2}}{\partial \mathrm{y}_{k+1}} & \frac{\partial e_{k,2}}{\partial \psi_{k+1}} \\ \frac{\partial e_{k,3}}{\partial \mathrm{x}_{k+1}} & \frac{\partial e_{k,3}}{\partial \mathrm{y}_{k+1}} & \frac{\partial e_{k,3}}{\partial \psi_{k+1}} \end{bmatrix} = \begin{bmatrix} \frac{cos\psi_k}{\Delta T_k} & \frac{sin\psi_k}{\Delta T_k} & 0 \\ -\frac{sin\psi_k}{\Delta T_k} & \frac{cos\psi_k}{\Delta T_k} & 0 \\ 0 & 0 & \frac{1}{\Delta T_k} \end{bmatrix} \tag{21}$$

$$\therefore J_{k+1} = \frac{1}{\Delta T_k} \begin{bmatrix} cos\psi_k & sin\psi_k & 0 \\ -sin\psi_k & cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix} = \frac{1}{\Delta T_k} \mathrm{M}_k{}^{-1} \tag{22}$$

Algorithm 3 is explained how to compute the generating error from the vehicle kinematic based on the above equations (2) and (14-22).

Algorithm 3 VehicleKinematicEdge - g²o system [1] [2]

Input: Current vehicle vertex, Rotation angle of the vehicle, and ΔT_k

1: **Initialise** $priorX \leftarrow edgeVertices(1), \psi_k \leftarrow priorX(3)$ and $M \leftarrow \begin{bmatrix} \cos\psi_k & -\sin\psi_k & 0 \\ \sin\psi_k & \cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix}$

2: $edgeVertices(2) \leftarrow edgeVertices(1) + \Delta T_k Mz$: Define the posterior assuming no noise

3: $errorZ \leftarrow inv(M)\frac{edgeVertices(2)-edgeVertices(1)}{\Delta T_k} - z$: Compute the error

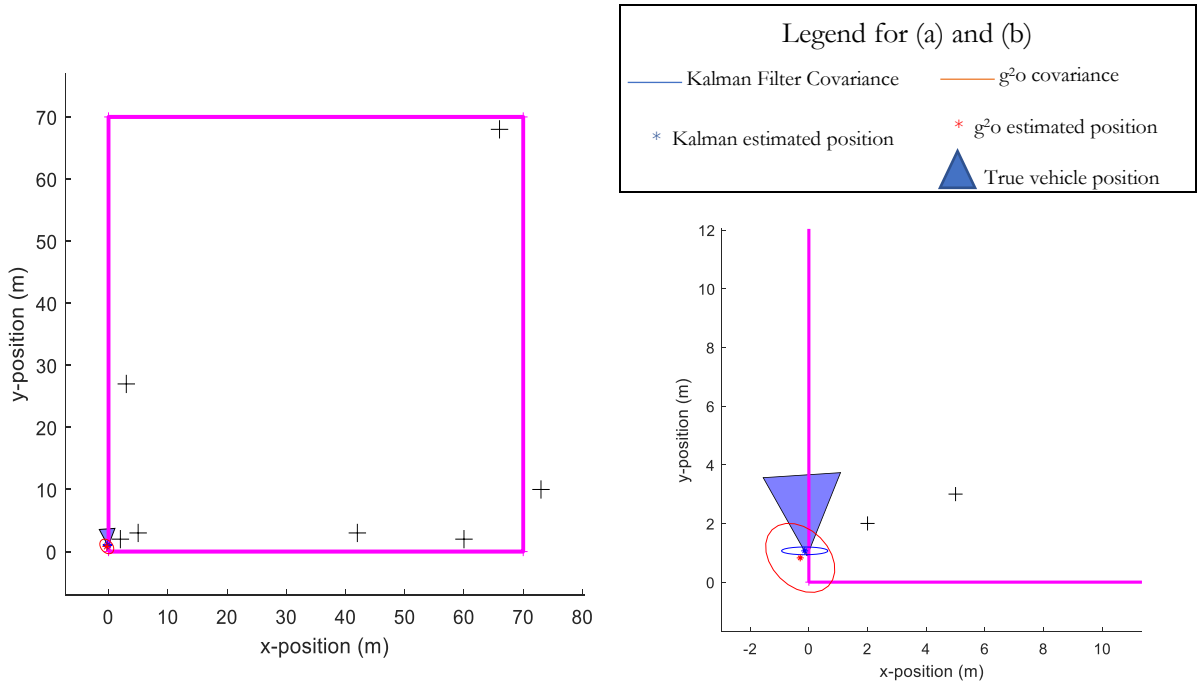
4: $J_k \leftarrow \frac{1}{\Delta T_k}\begin{bmatrix} -cos\psi_k & -sin\psi_k & -sin\psi_k(x_{k+1}-x_k)+cos\psi_k(y_{k+1}-y_k) \\ sin\psi_k & -cos\psi_k & -cos\psi_k(x_{k+1}-x_k)-sin\psi_k(y_{k+1}-y_k \\ 0 & 0 & -1 \end{bmatrix}$: The current step of the Jacobian function

5: $J_{k+1} \leftarrow \frac{1}{\Delta T_k} M_k{}^{-1}$: The next step of the Jacobian function (prediction)

Output: None

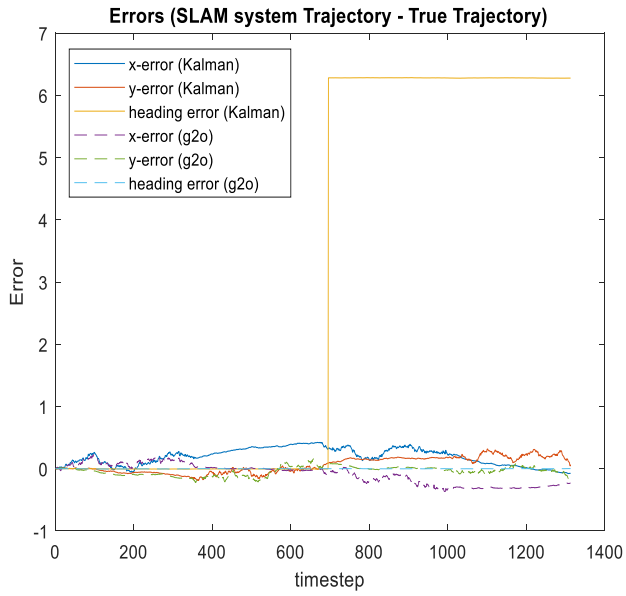
Question 1c) **Compare Performance**

As hypothesized in the coursework specifications, the trajectories between both algorithms provide very similar results. This is likely the case as the trajectory is fairly simple with the covariance associated with the odometry not being very high initially. Figure 2(f) shows the trajectory of the vehicle with both algorithms showing very similar results, further the final position is also very similar. Both algorithms utilise a similar process model for prediction of the trajectory with each odometry event, however, q2c has a slightly more complex odometry, and this is where large difference in performance between the two algorithms. As both algorithms perform some degree of linearization estimation to non-linear systems, this would have a larger effect on a more complex odometry particularly for the Kalman filter, whereas this trajectory is fairly straightforward.

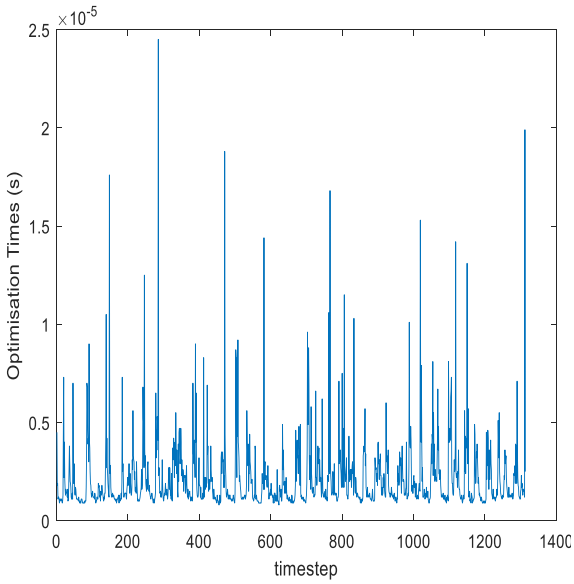


(a) Simulation Output

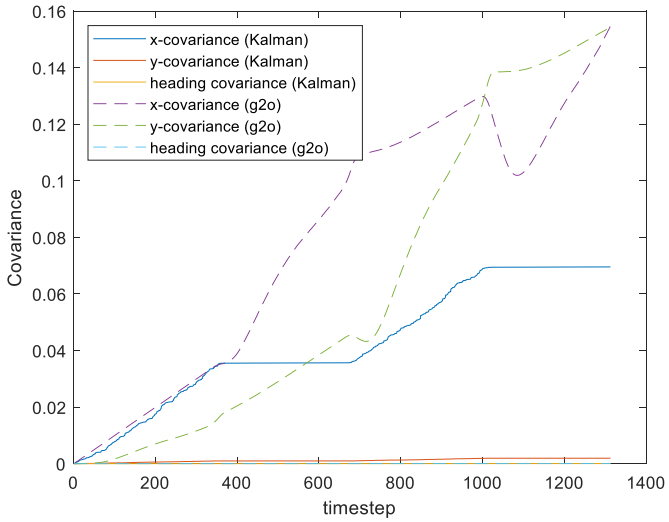
(b) Zoom in – Simulation Output



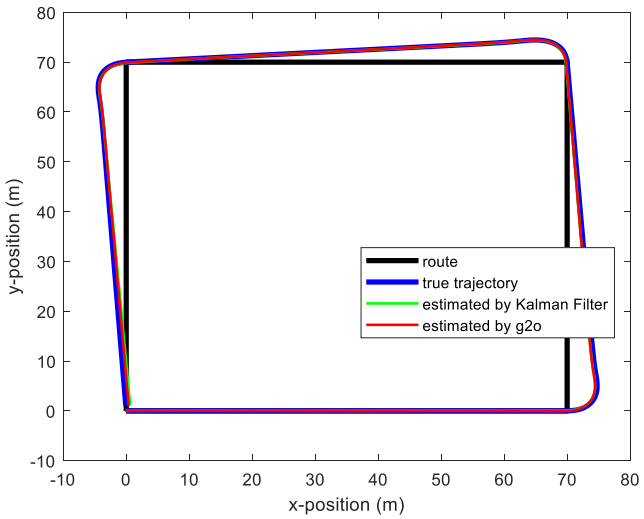
(c) Output errors



(d) Optimisation times



(e) Vehicle Covariance

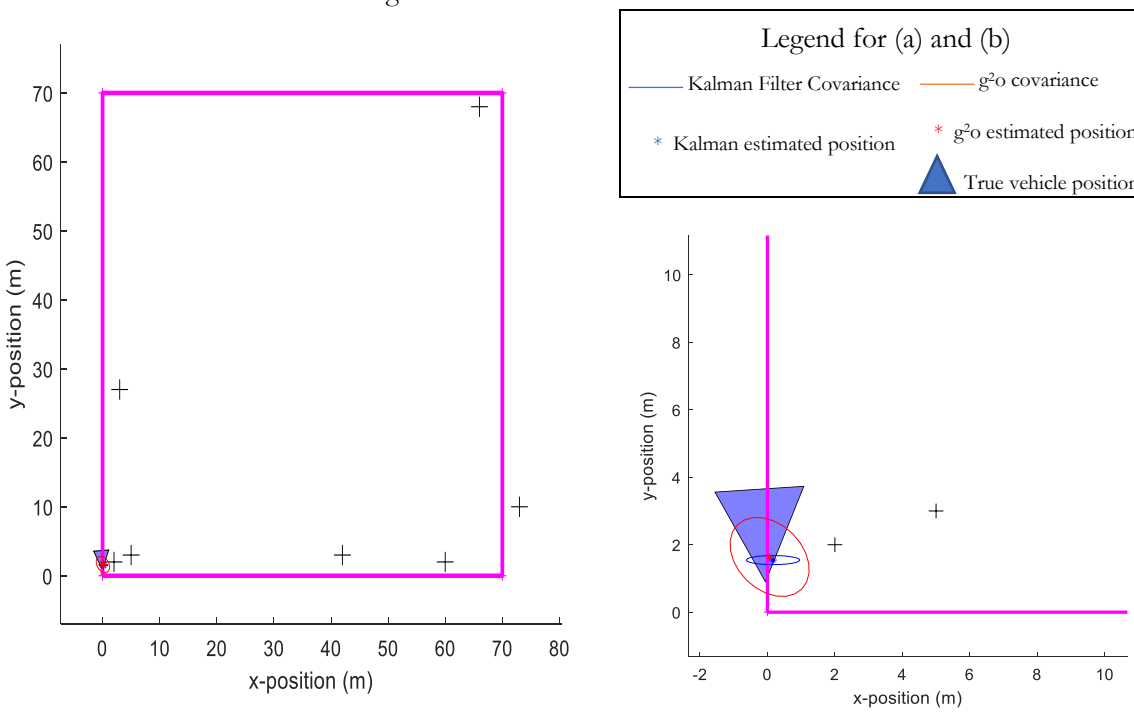


(f) Trajectory comparison

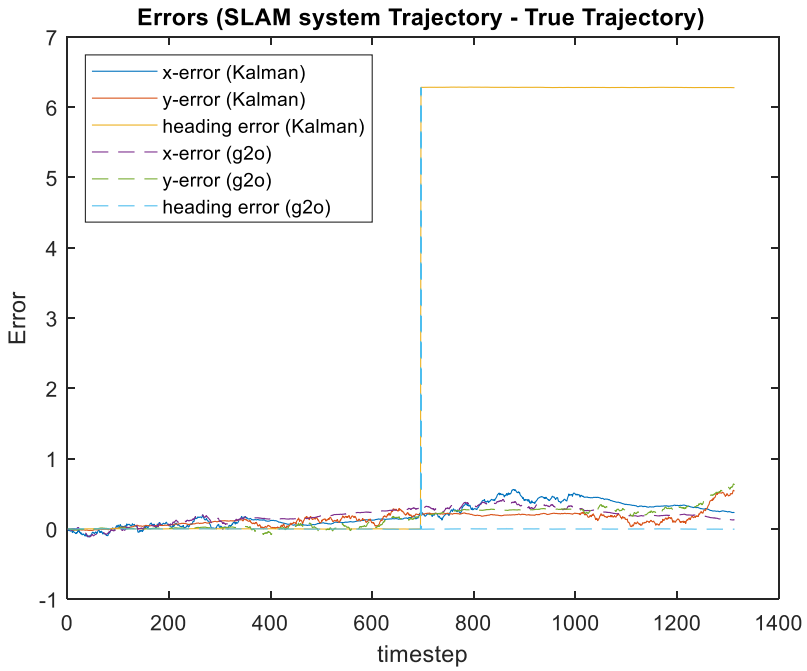
Figure 2. Comparison of the performance between the Kalman filter and graph-based model (g²o) system.

Question 1d) **Optimiser modification**

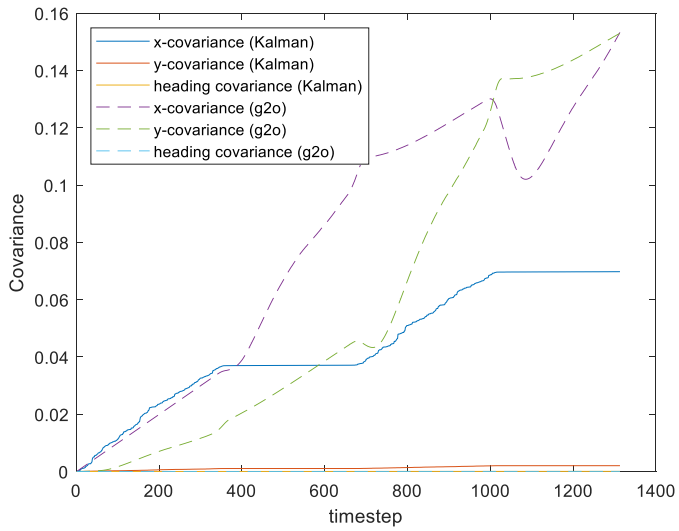
For question 1c, the results were defined using the function of optimisation as false, which means that the optimisation was applied to the loop cycle only once (at the end of the trajectory estimation) to provide fast result values. However, this might drastically affect estimates of the system, and may make it difficult to get accurate results. The optimisation is applied every 100 timesteps for each loop is shown in this question. Comparing between Figure 2(c) and Figure 3(c), the Kalman filter system's error looks similar, but the error of the graph-based model can be seen significantly reduced and close to zero. It means that despite the results being provided slower when ***recommendOptimization*** returns true, the results from Figure 3 are more accurate than the results from Figure 2.



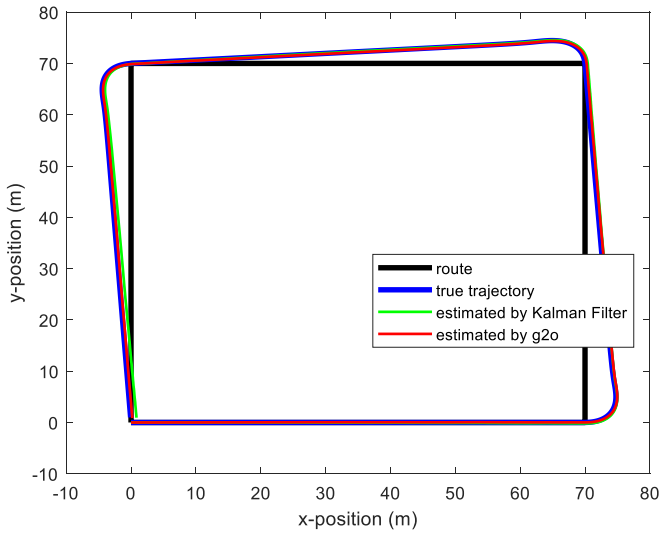
(a) Simulation Output (b) Zoom in – Simulation Output



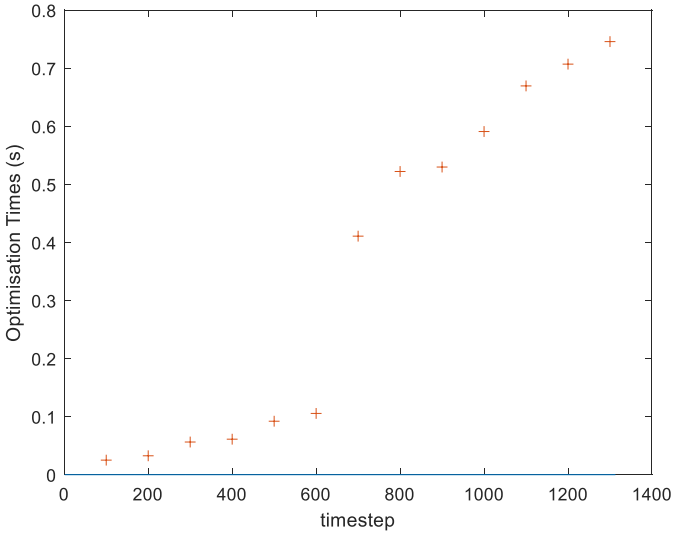
(c) Output errors



(d) Vehicle Covariance



(e) Trajectory Comparison



(f) Optimisation times

Figure 3. Comparison of the performance between the Kalman filter and graph-based model with the optimiser set to run every 100 steps.

Question 2. GPS Measurement Implementation

Question 2a) **Kalman Filter GPS extension**

The GPS observation model of the Kalman filter is directly estimates the position of the robot in real system. Before estimating the position of the robot, the observation model needs to compute the fusing platform, which represents a sensor function, and the prediction variables from question 1a. To extract the GPS measurement, the information of orientation of the vehicle is not provided so that the orientation predicted by odometry will be used in the observation model.

$$z_k = \begin{bmatrix} x_k \\ y_k \\ \psi_k \end{bmatrix} \tag{23}$$

The third step of the SLAM Kalman filter structure is to update the prediction of the vehicle state and covariance using the basic Kalman filter equations. The update state is in terms of averaging and combining the prediction model. There exist several kinds of the basic update Kalman filter equations, but the equations detailed in (24-25) were used because the equation (25) has a minus sign, which implies that the update covariance will never increase and be the same or decrease, so it generates a more stable covariance result and reduces the error. Moreover, the second term of the equation (25) must be symmetric to easily compute the updated covariance because the covariance matrix is not always symmetric [3].

$$\hat{s}_{k+1|k+1} = \hat{s}_{k+1|k} + W_{k+1} v_{k+1|k} \tag{24}$$

$$\hat{P}_{k+1|k+1} = \hat{P}_{k+1|k} - W_{k+1} S_{k+1|k} {W_{k+1}}^T \tag{25}$$

$v_{k+1|k}$ is the innovation, which is the difference between the prediction and update term, $S_{k+1|k}$ is the innovation covariance, and W_{k+1} is the Kalman filter weight. These equations [3] can be computed as follows:

$$v_{k+1|k} = z_k - H_s \hat{s}_{k+1|k} \tag{26}$$

$$C_{k+1|k} = P_{k+1|k} {H_s}^T \tag{27}$$

$$S_{k+1|k} = H_s C_{k+1|k} + R \tag{28}$$

$$W_{k+1} = C_{k+1|k} {S_{k+1|k}}^{-1} \tag{29}$$

, where $C_{k+1|k}$ is the relationship between covariance prediction ($P_{k+1|k}$) and the transpose of the observation. The near-optimal Kalman filter weight can be computed using the standard Kalman filter equations (26-29) as follows:

$$W_{k+1} = P_{k+1|k} {H_s}^T (H_s C_{k+1|k} + R)^{-1} \tag{30}$$

$$W_{k+1} = P_{k+1|k} {H_s}^T (H_s P_{k+1|k} {H_s}^T + R)^{-1} \tag{31}$$

Therefore, the update model (24-25) can be easily calculated using the already known parameters as prediction state, covariance, the observation model (z_k) and the basic Kalman filter equations (26-31).

$$\hat{s}_{k+1|k+1} = \hat{s}_{k+1|k} + W_{k+1} (z_k - H_s \hat{s}_{k+1|k}) \tag{32}$$

$$\hat{P}_{k+1|k+1} = \hat{P}_{k+1|k} - W_{k+1} (H_s P_{k+1|k} {H_s}^T + R) {W_{k+1}}^T \tag{33}$$

All-computation steps to update the position estimates of the vehicle with GPS measurements are described by algorithm 4.

Algorithm 4 handleGPSObservationEvent – Kalman Filter System [3]

Input: data (x and y position of the vehicle), xPred and PPred

1: **Initialise** $z_k \leftarrow [0;0;0], H_s \leftarrow \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$, and $R_s \leftarrow \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$

2: $z_k \leftarrow [event.data(1); event.data(2); xPred(3)]$: xPred(3) is a heading (ψ_k)

3: $R_s(1:2, 1:2) \leftarrow event.covariance$

4: $W_{k+1} \leftarrow P_{k+1|k}{H_s}^T(H_s C_{k+1|k} + R)^{-1}$

5: $xEst = xPred + W_{k+1}(z_k - H_s xPred)$

6: $PEst = PPred - W_{k+1}(H_s PPred{H_s}^T + R){W_{k+1}}^T$

Output: *xEst and PEst*

Question 2b) **g²o MATLAB GPS extension**

To extend the g²o system to handle GPS measurements, a unary edge was created as shown in figure 4, where the observation $\mathrm{z}_k^{\mathrm{G}}$, represents the measurement from the GNSS system. The GNSS system as specified earlier provides direct measurements of the position of the robot, which entails that no new vertex is needed. As shown in algorithm 5, the edge is connected to the vehicle state vertex whenever a GPS event is called.

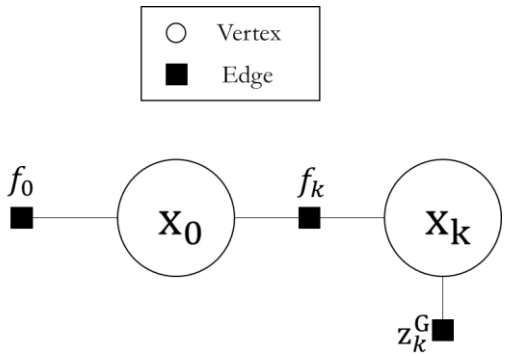


Figure 4: Visual representation of the factor graph incorporating the GPS measurement $\mathrm{z}_k^{\mathrm{G}}$ as a unary edge.

As with the other edge or vertex creation functions, g²o requires specification of the error function and the Jacobians for covariance computation during optimisation. The error was defined as the difference between the x_k and y_k positions of the robot and the GPS x_k and y_k measurements as shown in (34) where E describes the error.

$$\mathrm{E}[\mathrm{z}_k^G] = \begin{bmatrix}x_k\\y_k\end{bmatrix} - z_k^G \tag{34}$$

The Jacobian was computed from this and set as $J = \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\end{bmatrix}$.

Algorithm 5 handleGPSObservationEvent - g²o system

Input: Current Vehicle Vertex, GPS measurement, GPS Covariance

1: newGPSEdge = GPSMeasurementEdge(): create new unary GPS edge

2: newGPSEdge.setVertex(1, currentVehicleVertex): set the edge to connect to the current vehicle vertex

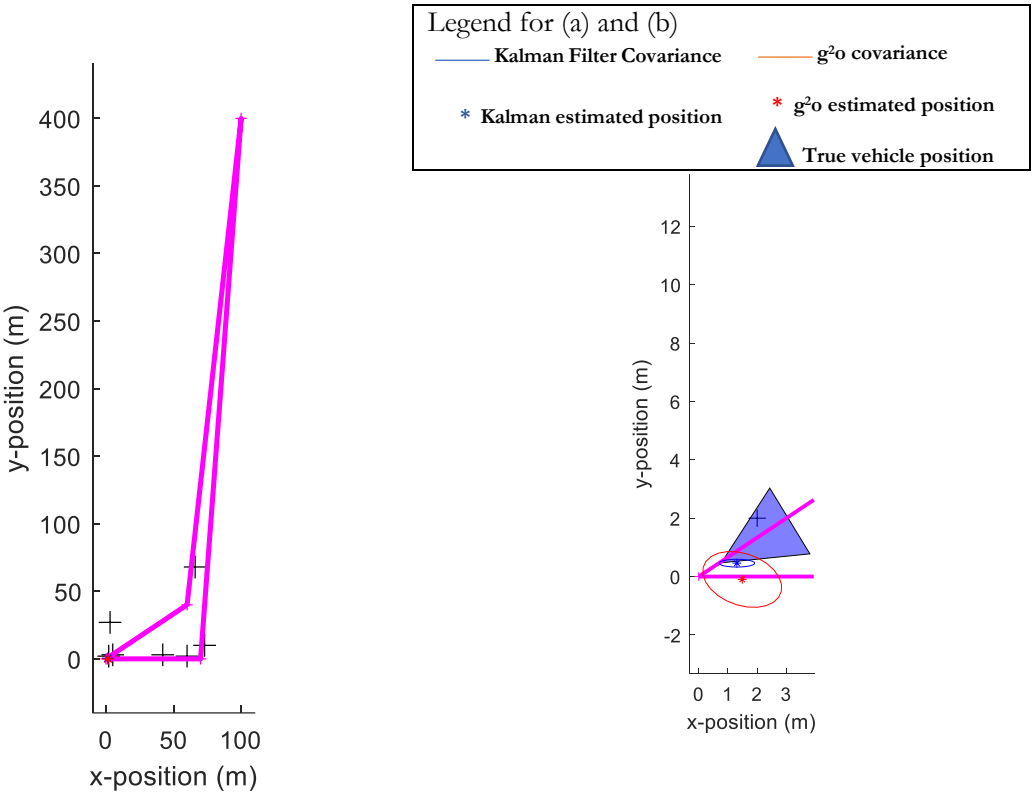
3: newGPSEdge.setMeasurement(GPS_Measurement): extracts the GPS measurement from the simulator and sets it to the GPS edge.

4: Omegaw$_k^{\mathrm{G}}$ = inverse(GPS Covariance Matrix): Defining the Hessian matrix

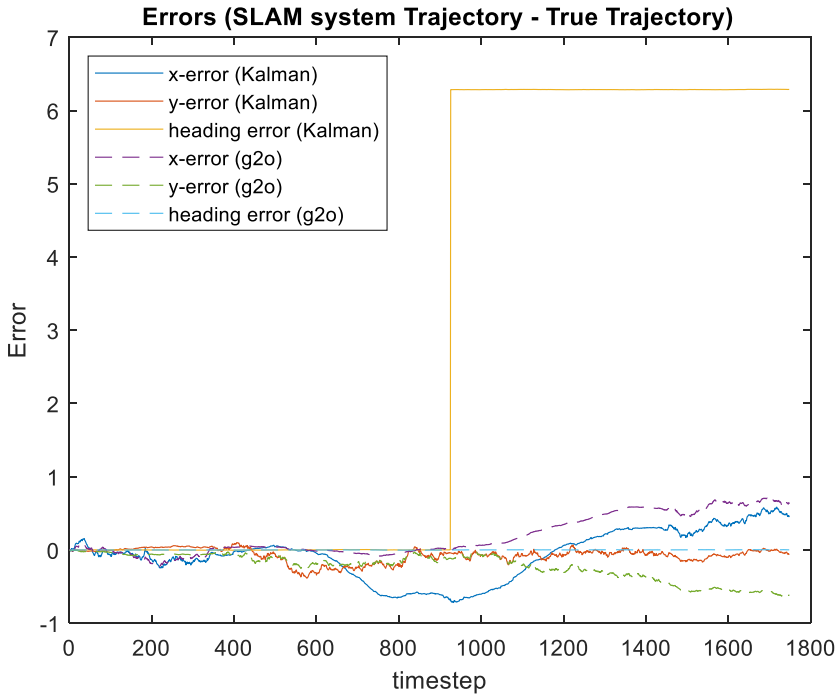
5: newGPSEdge.setInformation(Omegaw$_k^G$): setting information matrix omega (Hessian) to edge
6: Graph.addEdge(newGPSEdge): Add the new GPS edge to the graph
Output: None

Question 2c) **Compare Performance (GPS Measurement Period Variation)**

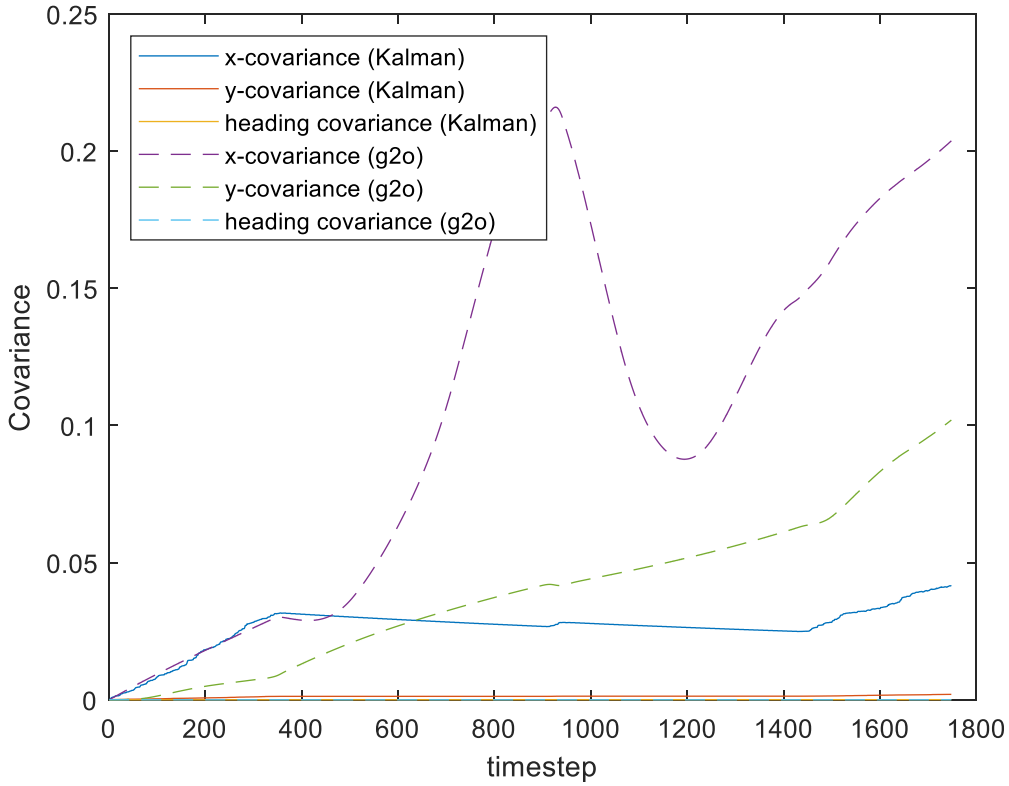
i) GPS Measurement Period = 1 second



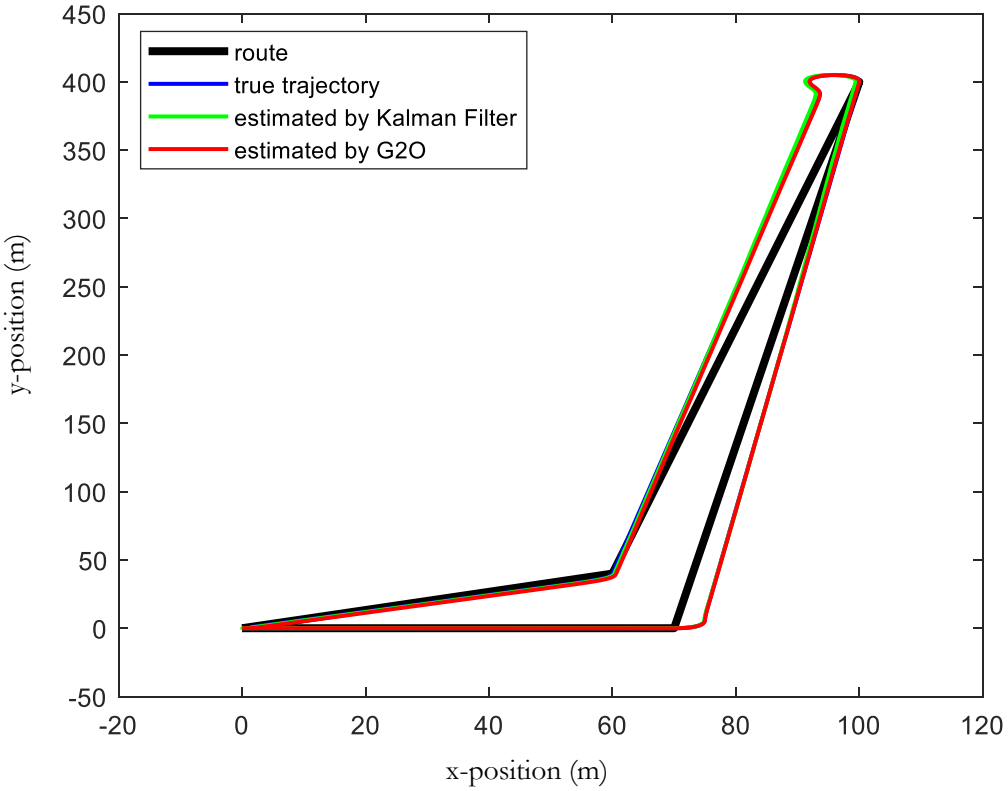
(a) Simulation Output (b) Zoom in – Simulation Output



(c) Output errors



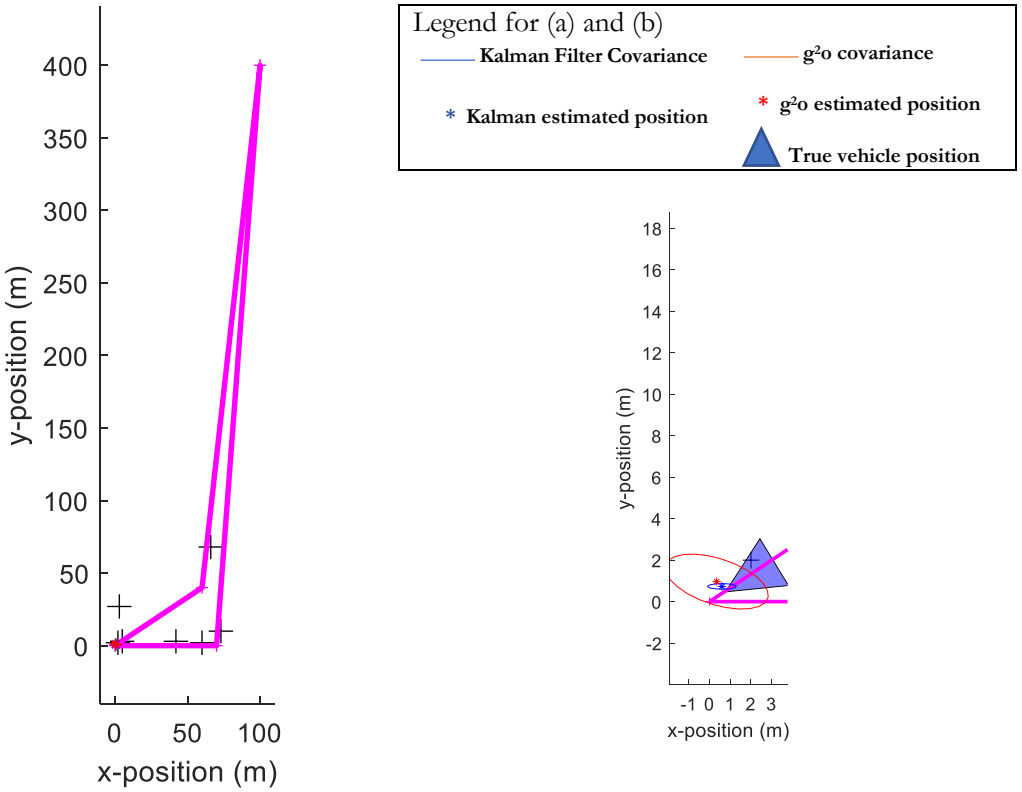
(d) Vehicle Covariances



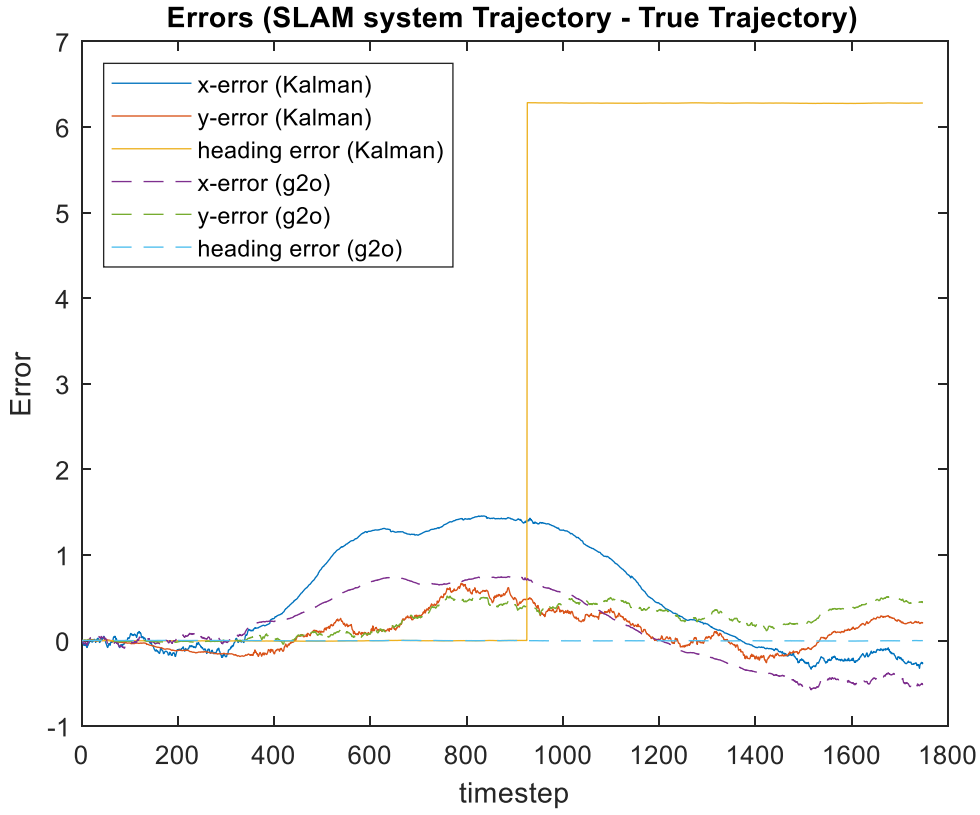
(e) Trajectory Comparisons

Figure 5: Plots of the more complicated odometry with GPS implementation and the GPS measurements set to run every 1 second.

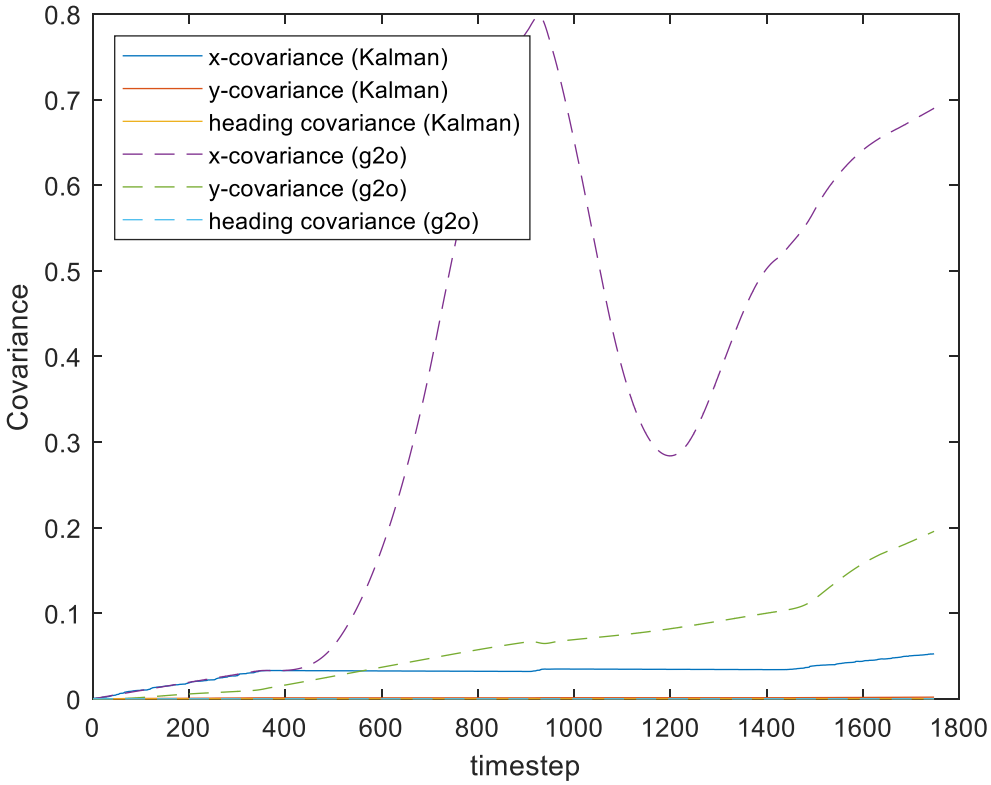
ii) GPS Measurement Period = 5 seconds



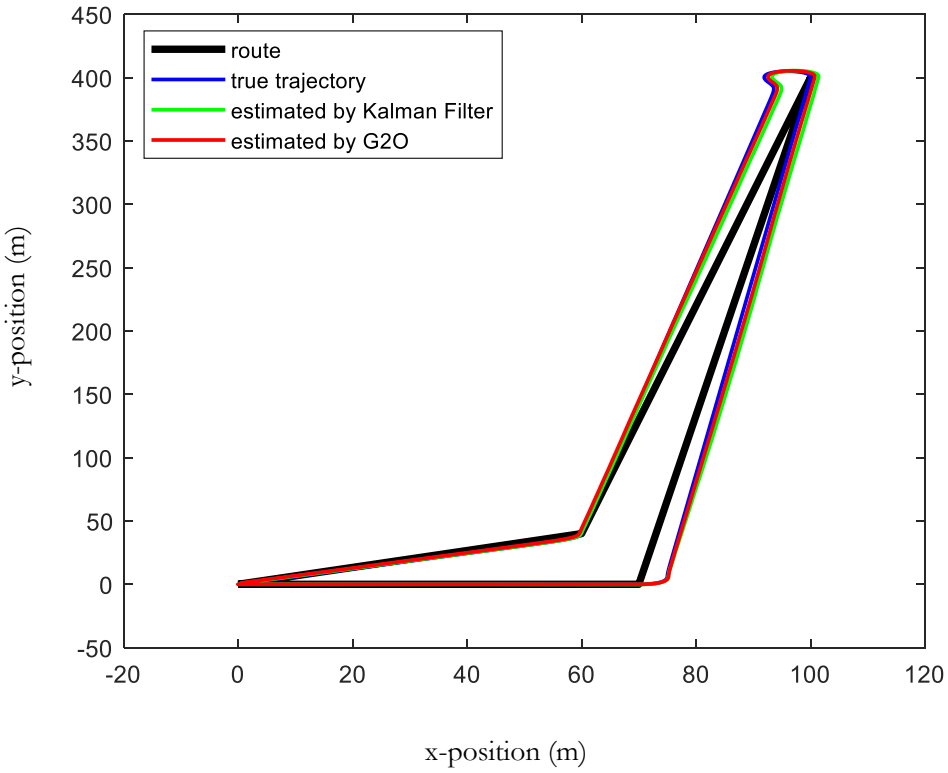
(a) Simulation Output (b) Zoom in – Simulation Output



(c) Output errors



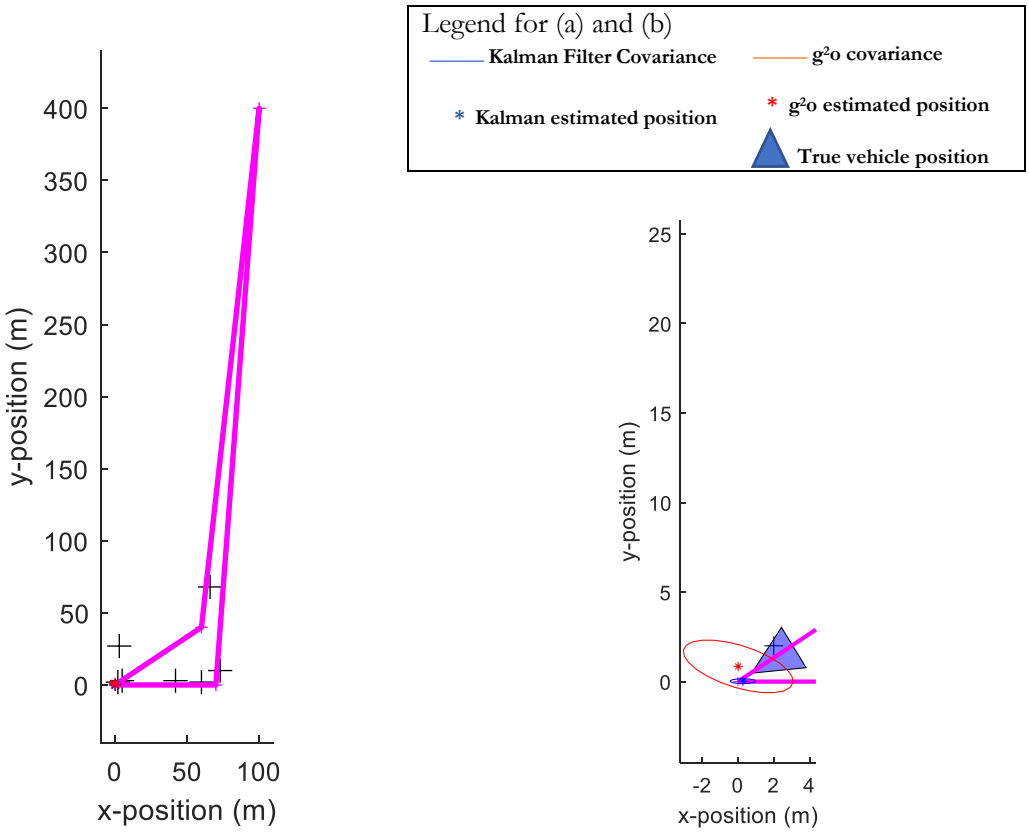
(d) Vehicle Covariances



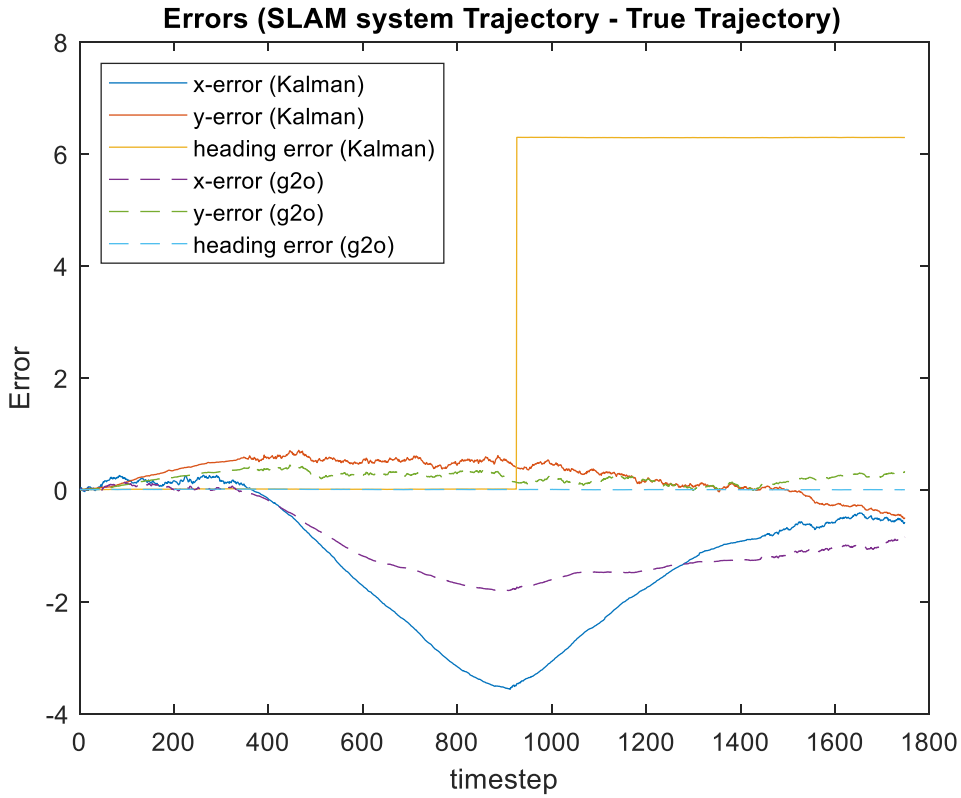
(e) Trajectory Comparisons

Figure 6: Plots of the more complicated odometry with GPS implementation and the GPS measurements set to run every 5 seconds.

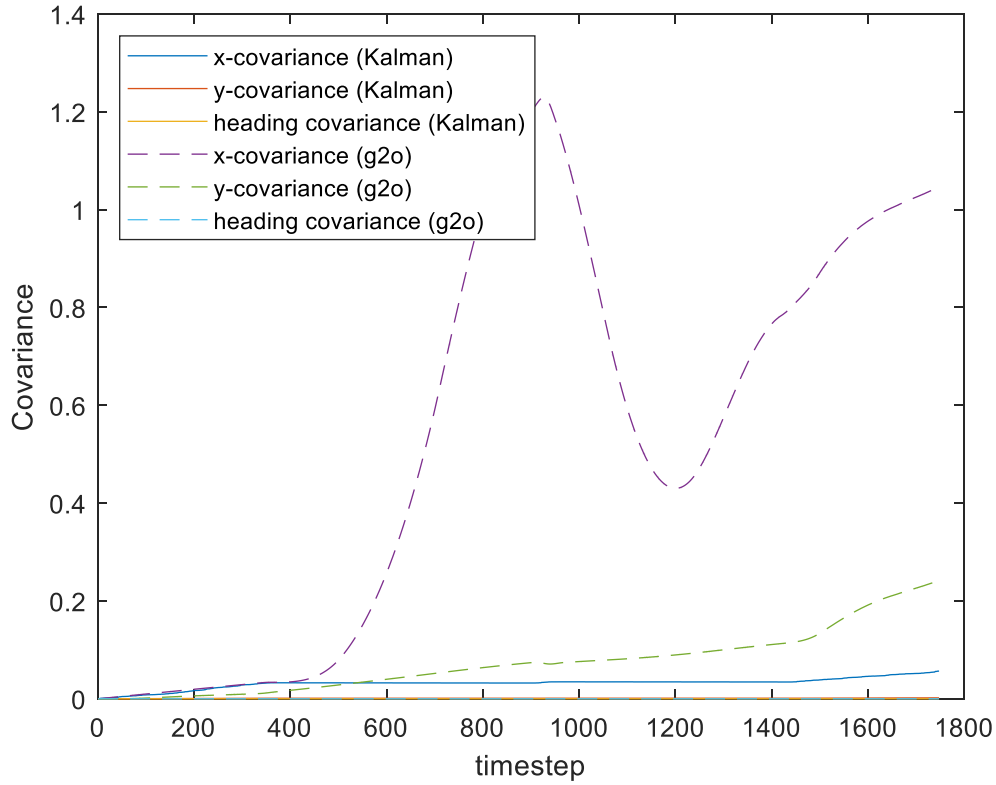
iii) GPS Measurement Period = 10 seconds



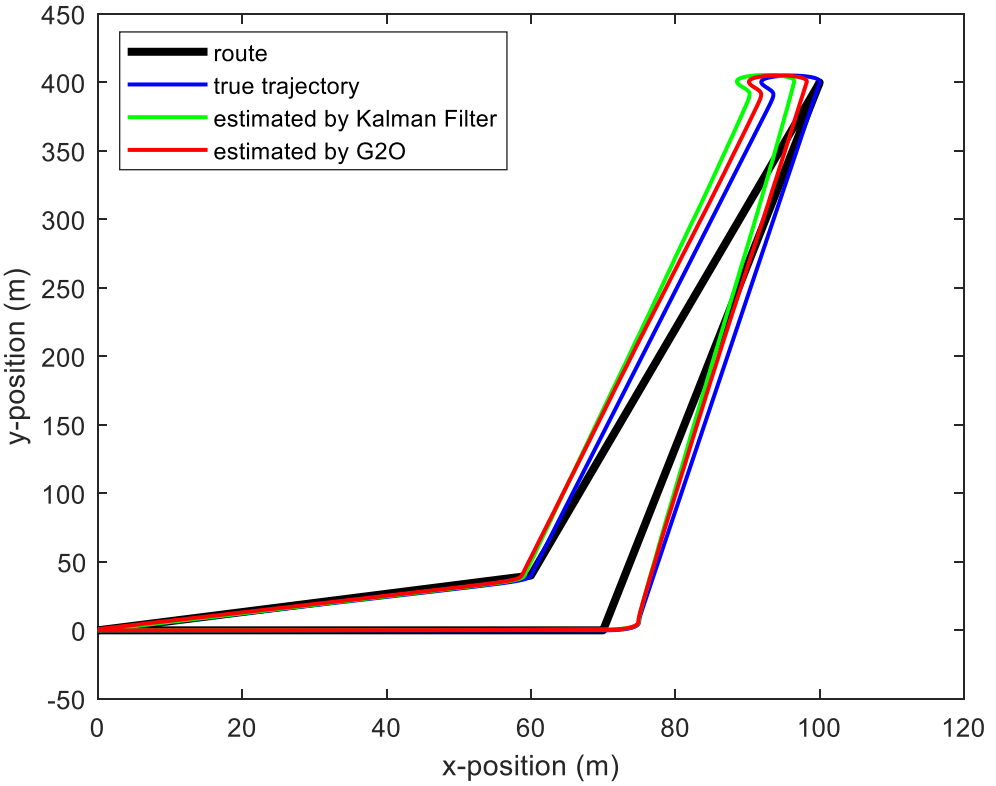
(a) Simulation Output (b) Zoom in – Simulation Output



(c) Output errors



(d) Vehicle Covariances



(e) Trajectory Comparisons

Figure 7: Plots of the more complicated odometry with GPS implementation and the GPS measurements set to run every 10 seconds

From these plots a few phenomena can be seen quite consistently as the frequency of GPS measurements decreases. Firstly, the vehicle covariance increases with increased periods of GPS measurement. This is an expected result as there are fewer edges created in the g²o implementation. The Kalman filter's covariance does not increase as significantly as the g²o implementation. Secondly, the error increases with the increased period which once again is an expected result and is largely attributed to the same reasons as the increased covariance. Thirdly, the trajectory variation from the true trajectory is very significant as the period increases; this is particularly visible when comparing figures 5e and 7e, where a clear distinction of the trajectory lines is present.

Question 3. g²o SLAM Implementation

Question 3a) **System Description**

For implementation of the full SLAM system, the class structure detailed in table 1 was implemented, specifically, the classes LandmarkMeasurementEdge and LandmarkVertex were created. Landmarks are denoted as m^i where i denotes the landmark ID and the positional state is defined in (35).

$$\mathrm{m}^i = \begin{bmatrix} x^i \\ y^i \end{bmatrix} \tag{35}$$

As the landmarks have a positional state, and as shown in table 1, they were modelled as vertices. A full visual representation of the structure of the factor graph is shown in figure 5. Landmark observations were modelled as binary edges where a connection between the vehicle and the landmarks were made. The option to create a separate edge for the 2D range bearing (laser) sensor measurements was considered but as the observations were to be absorbed by the landmark-vehicle edge, this was not implemented as it would reduce the number of edges that would be optimised.

The landmark observation model using the range-bearing sensor is detailed in (36-38) where r_k^i is the range, and β_k^i is the azimuth and elevation of the landmark relative to the platform frame and ϕ_k represents the orientation of the laser. The environment was given as populated with landmarks that already have assigned IDs eliminating the need for a landmark numbering system.

$$\mathrm{z}_k^L = \begin{bmatrix} r_k^i \\ \beta_k^i \end{bmatrix} + \mathrm{w}_k^L \tag{36}$$

$$r_k^i = \sqrt{(x^i - x^k)^2 + (y^i - y^k)^2} \tag{37}$$

$$\beta_k^i = \tan^{-1}\left(\frac{y^i - y^k}{x^i - x^k}\right) - \phi_k \tag{38}$$

As before with the creation of any edge, error functions and Jacobians must be defined for the g²o system. The error was defined as shown in (39) with the computed Jacobians shown in (42-43) which use the definitions from (40-41) and was adapted from [4].

$$E[\mathrm{z}_k^L] = \begin{bmatrix} r_k^i \\ \beta_k^i \end{bmatrix} - z_k^L \tag{39}$$

$$dx = \begin{bmatrix} x^i - x^k \\ y^i - y^k \end{bmatrix} \tag{40}$$

$$r = \sqrt{dx^2} \tag{41}$$

$$J_1 = \begin{bmatrix} -\frac{x^i - x^k}{r} & -\frac{y^i - y^k}{r} & 0 \\ \frac{y^i - y^k}{r^2} & -\frac{x^i - x^k}{r^2} & 0 \end{bmatrix} \tag{42}$$

$$J_2 = \begin{bmatrix} \frac{x^i - x^k}{r} & \frac{y^i - y^k}{r} \\ -\frac{y^i - y^k}{r^2} & \frac{x^i - x^k}{r^2} \end{bmatrix} \tag{43}$$

The code was modelled to recognize landmarks and add them to the graph as the edges and vertices shown in figure 3 by the implementation of two functions: createOrGetLandmark and handleLandmarkObservationEvent which are detailed in the next section.

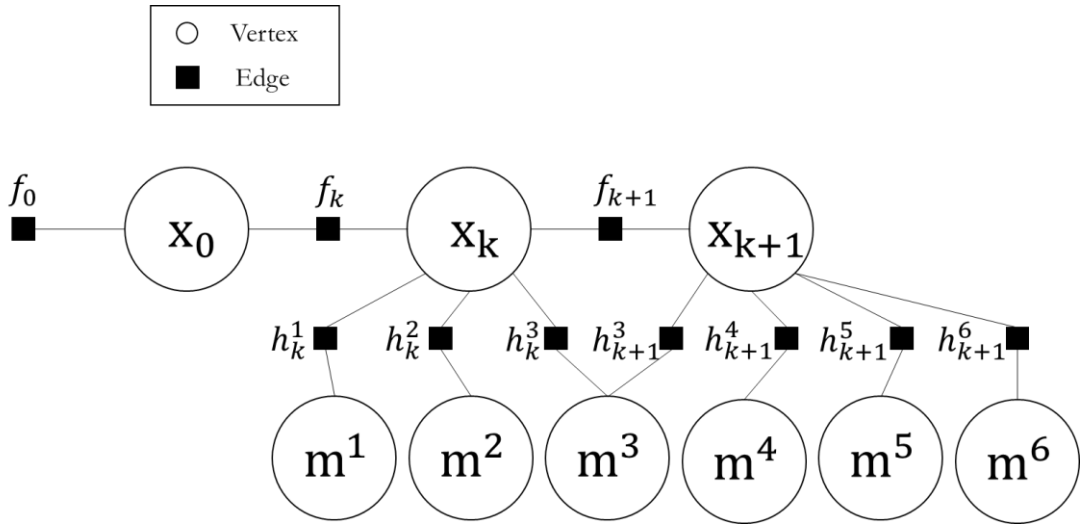


Figure 5: Visual representation of the factor graph with the extension of landmarks and edges that connect the landmarks to the robot for the SLAM system.

Question 3b) **g²o SLAM Solution**

Following initialisation, the vehicle was designed to observe all of the landmarks which the range-bearing sensor could sense. If the landmark was new, a new vertex was created and added to the graph, if however, a landmark was recognized the system would return it to the SLAM system and no vertex would be recreated. This is detailed in more in the pseudocode algorithm 6.

Algorithm 6 createOrGetLandmark - g²o system

Input: Current Vehicle Vertex, Landmark ID
1: **if** *observed landmark* is in *Robot Landmark Map*
2: | **return** the landmark
3: | newVertexCreated = **false**: Do not create vertex
4: **end if**
5: **Else**
6: Create Landmark Vertex
7: newVertexCreated = **true**: create vertex (*Boolean*)
8: Add created vertex to Landmarks map
Output: landmarkVertex, newVertexCreated *(Boolean)*

Algorithm 7 describes the process when a landmark observation event was triggered. It can be seen that it calls algorithm 6 within it and then proceeds to create a vertex for the landmark and an associated vehicle-landmark binary edge.

Algorithm 7 handleLandmarkObservationEvent - g²o system

Input: Current Vehicle Vertex, Landmark Vertex, Range-Bearing Observation, Range-Bearing Covariance

1: **for each** *observed landmark* in *Robot Landmark Map*
2: [landmarkVertex, newVertexCreated] = createOrGetLandmark(observed landmark)
3: **if** newVertexCreated is **true**
4: Graph.addVertex(landmarkVertex): add Vertex to the graph
5: set vertex to origin
6: **end if**
7: LandmarkEdge = LandmarkMeasurementEdge(): Create BinaryEdge Landmark-Vehicle Edge
8: LandmarkEdge.setVertex(1, currentVehicleVertex): set the edge to connect to the current vehicle vertex
9: LandmarkEdge.setVertex(2, Landmark Vertex): set the edge to connect to the landmark vertex
10: LandmarkEdge.setMeasurement(Range-Bearing Observation): set the edge to absorb the measurement of the Range-Bearing sensor
11: LandmarkEdge.setInformation($Range - Bearing\ Covariance^{-1}$): setting information matrix omega (Hessian) to edge
12: Graph.addEdge(LandmarkEdge): Add the new Landmark-Vehicle edge to the graph
13: **end for**

Output: None

Question 3c) **Performance of Graph**

When running the g²o SLAM the properties detailed in table 2 were investigated to evaluate the algorithm's performance. The performance of the SLAM in terms of the error and the covariances are presented in figure 9 from the next section. The landmark initialisation locations are plotted in figure 6 where there is high agreeability between the actual landmark locations and the estimated locations.

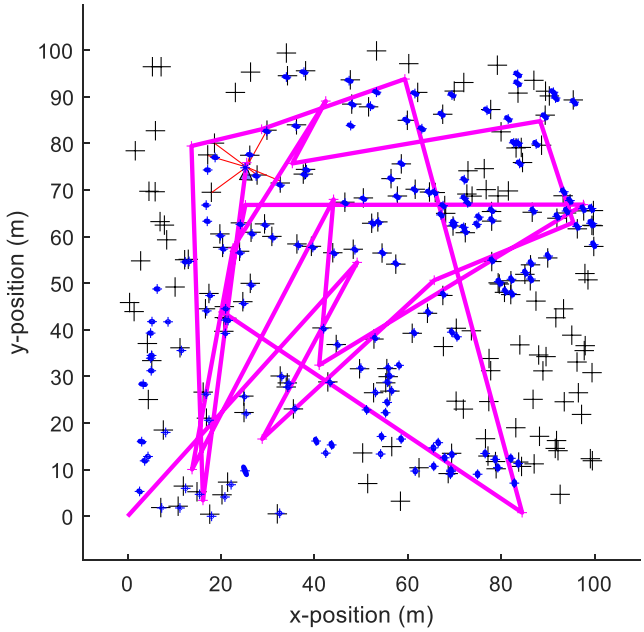


Figure 6: Landmark observations modelled by the robot vehicle are shown in blue dots relative to the actual landmark locations shown by the black crosses.

Table 2: Performance Properties of the SLAM algorithm

Property	**Value**
Number of Landmark Vertices	174
Number of Vehicle Vertices	5667
Number of Observations	4792
Average Number of Observations For Each Landmark	27.54
Number of Observations at Each Time Step	0.845

Question 3d) **Change Standard Deviation**

The performance properties detailed in table 2 of the SLAM algorithm with a larger standard deviation did not change and are subsequently not included for conciseness. However, qualitatively there are very large differences in the landmark location initializations errors. This is an expected result as the larger standard deviation for the heading implied that there would be more noise which the system was inherently unable to correct for. This also implies that with the same number of observations (as shown in table 2) and increased standard deviation error the location error would be greater which is a logical result. Figure 7 shows the increased error in landmark observation locations produced by the robot. Further, the red lines from the vehicle indicating the observation heading at that timestep clearly shows why the results are worse than in figure 6, as expected.

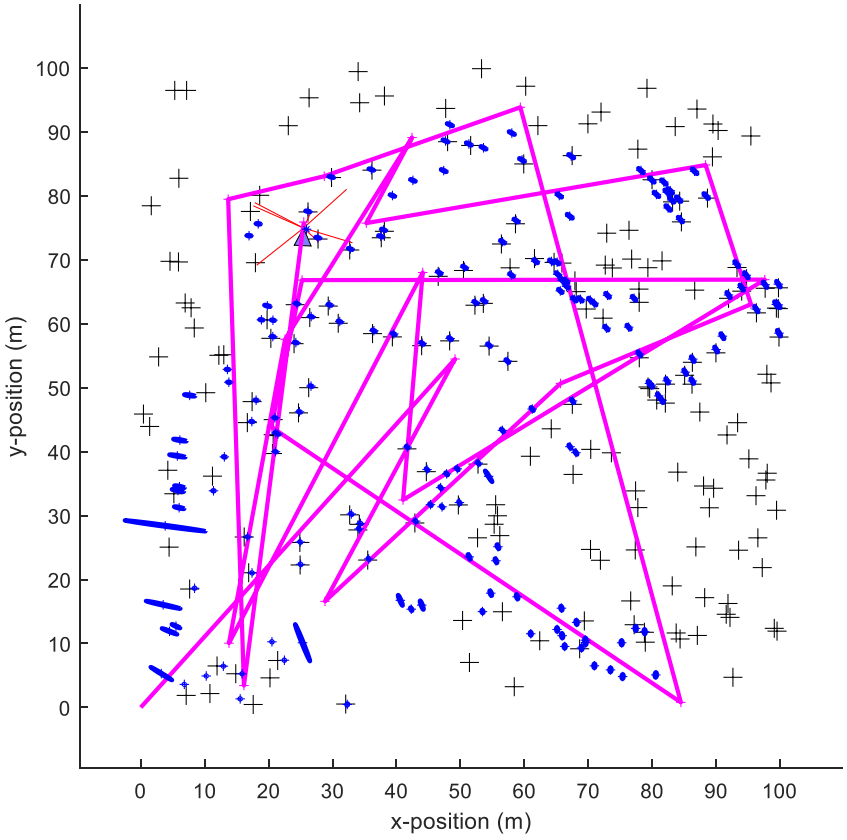


Figure 7: Landmark observations modelled by the robot vehicle are shown in blue dots relative to the actual landmark locations shown by the black crosses with increased standard deviation error.

Question 4. Performance Improvement

Ways to improve the performance of the SLAM system were investigated in this section following the structure of the coursework brief. Performance was assessed based on accuracy, computational time and amount of data stored depending on the questions.

Question 4a) **SLAM GPS extension**

By activating the GPS sensor to produce edges as detailed in question 2b, it was hypothesized that the SLAM system would perform better. However, the trade-off for improved accuracy is increased computational time as the system stores and optimises for a higher number of edges and vertices.
For this question, it was sought to minimize the number of GPS measurements taken while maintaining a good accuracy. This was conducted by iteratively changing the GPS measurement parameter and evaluating the total computational time and the solution accuracy.
The baseline was established as the system performance performed in question 3c. All cases were run on an identical Windows OS, with 10th Gen Core i7 intel processor.

Table 3: Computational time and system performance for various test cases.

Test Case	Comp. Time (s)	Average Trajectory Error	Standard Deviation Error	Average Covariance	Standard Deviation Covariance
1 Baseline No GPS measurement	917.571	x: -1.9382	x: 2.2201	x: 0.0054	x: 0.0038
		y: -1.5683	y: 1.6706	y: 0.0036	y: 0.0038
		ψ: 0.2767	ψ: 1.9375	ψ: 1.94e-06	ψ: 1.90e-07
2 Measurement Period=1s	1684.741	x: -0.2443	x: 0.4885	x: 0.0076	x: 0.0062
		y: -0.3998	y: 0.4774	y:0.0057	y: 0.0055
		ψ: 0.0133	ψ: 0.2998	ψ: 2.67e-06	ψ: 7.07e-07
3 Measurement Period=5s	1996.222	x: -1.3898	x: 1.5442	x: 0.0050	x: 0.0035
		y: -0.6888	y: 1.2314	y:0.0034	y: 0.0036
		ψ: 0.5102	ψ: 1.9034	ψ: 1.88e-06	ψ: 2.01e-07
4 Measurement Period=10s	1168.670	x: -0.6819	x: 1.4698	x: 0.0056	x: 0.0039
		y: -0.4680	y: 0.9370	y: 0.0038	y: 0.0040
		ψ: 0.5245	ψ: 1.7801	ψ: 2.04e-06	ψ: 2.07e-07
5 Measurement Period=20s	1046.149	x: -0.2958	x: 1.0356	x: 0.0057	x: 0.0041
		y: -2.2799	y: 1.1560	y: 0.0041	y: 0.0043
		ψ: 0.4139	ψ: 1.5948	ψ: 2.14e-06	ψ: 2.09e-07

Based on table 3 the determined optimal measurement period was found to be 10s, as the computational time was not significantly greater than the baseline and it provided a decrease in trajectory error of 44% with an increase in computational time of 11% relative to the 20s measurements. 5s as the GPS measurement period did not provide significantly better results and seemed to perform worse error-wise. However, the computational time was very significant taking 70.9% longer for the computer to produce the results. The estimated and true trajectories are plotted in figure 8 where the trajectory performs better than in figure 9c with no GPS observations (baseline case).

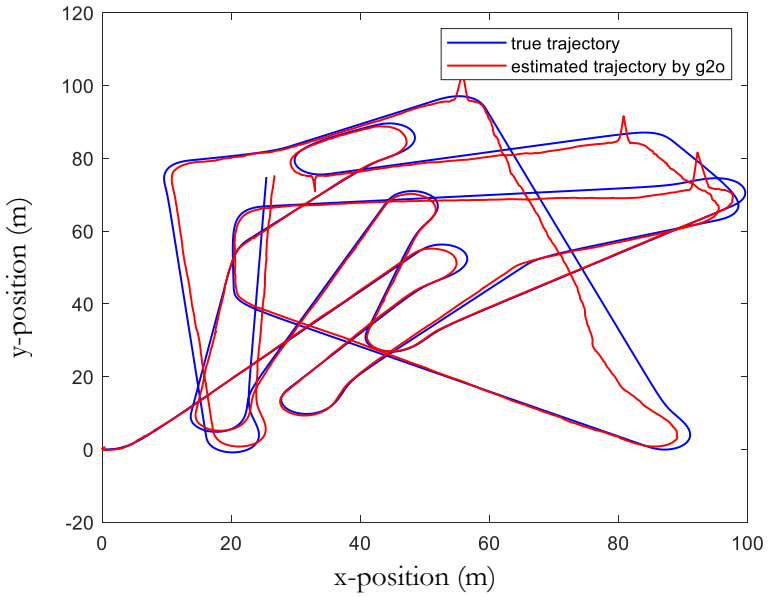


Figure 8: A comparison of the true and estimated trajectories with GPS measurements produced every 10s (deemed as optimal).

Question 4b) **Data storage reduction**

In this question, we implement an algorithm that could either remove all the vehicle edges or remove all but the first vehicle edge. To compare the performance of a SLAM system, we maintain the sensor observation data and open both GPS and Laser sensors. We will set GPSMeasurementPeriod to 10s.

The error, covariance and trajectory plots of original setting are shown below:

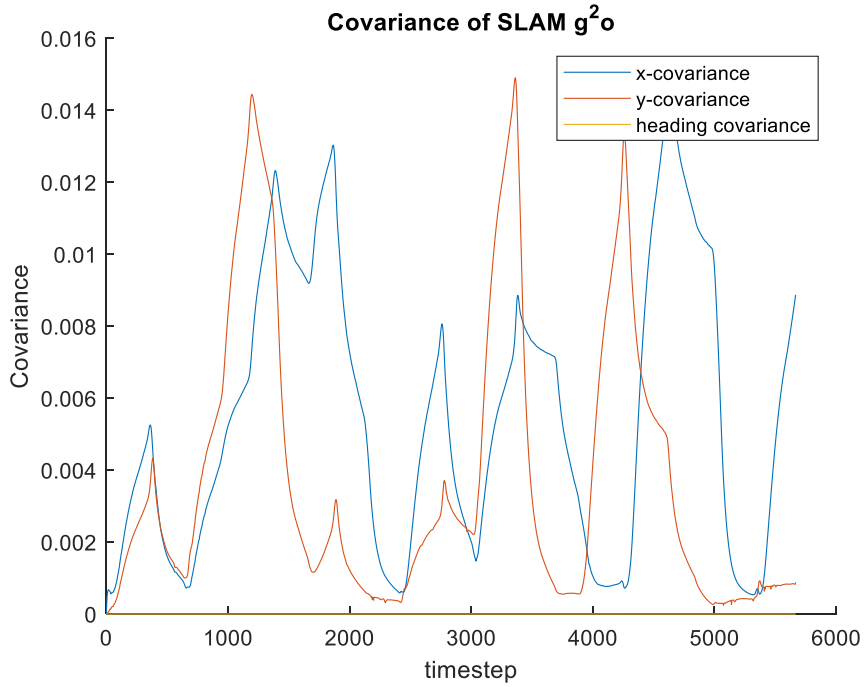


Figure 9 (a): Covariance plot for original setting with no data storage reduction.

Figure 9 (b) : Error plot for original setting with no data storage reduction.

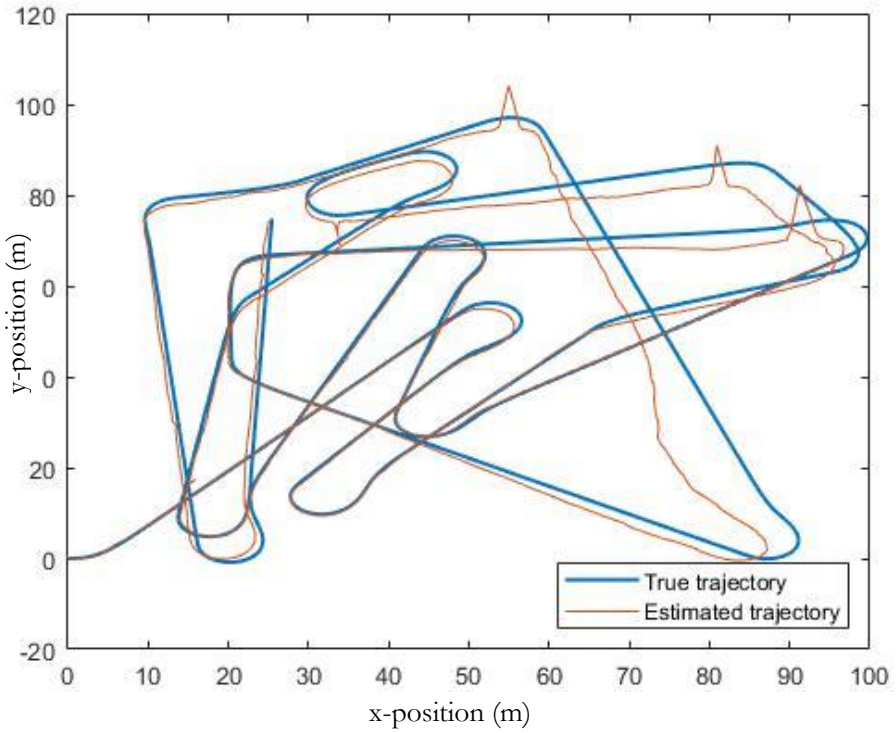


Figure 9 (c): Trajectory plot for original setting with no data storage reduction.

The error, covariance and trajectory plots of all-removed setting is shown below:

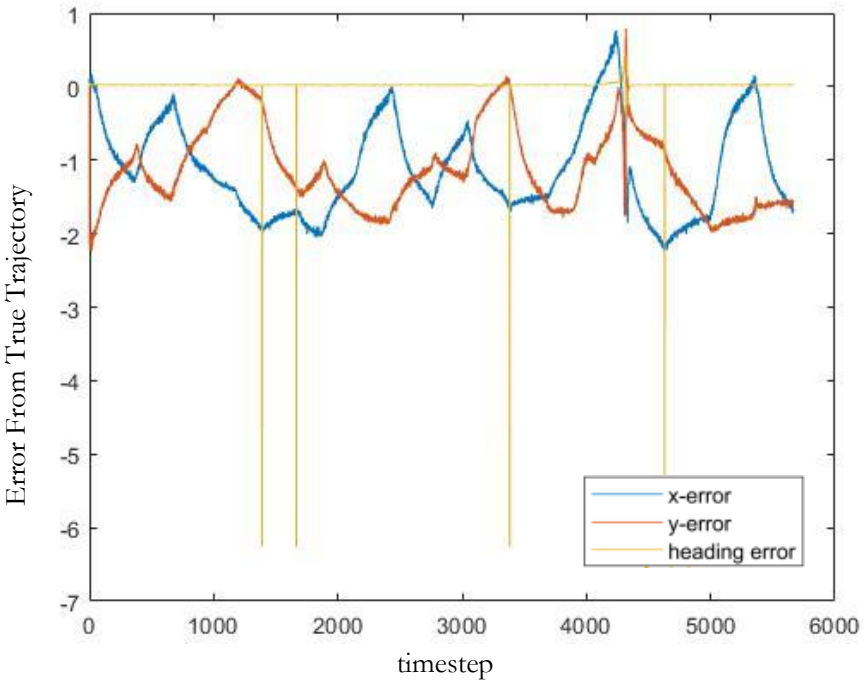


Figure 10 (a): Errors for removing all vehicle edges.

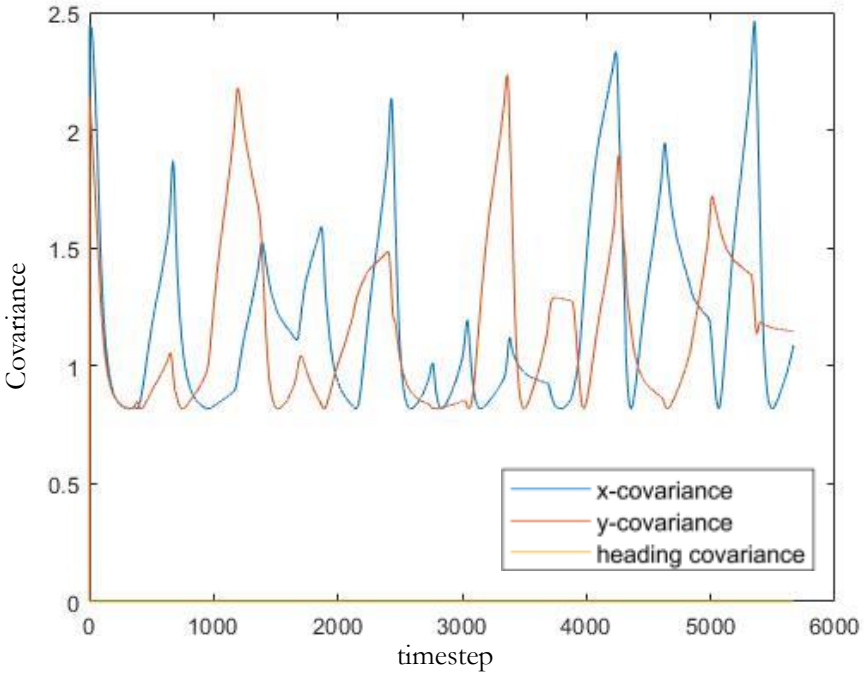


Figure 10 (b): Covariance when removing all vehicle edges.

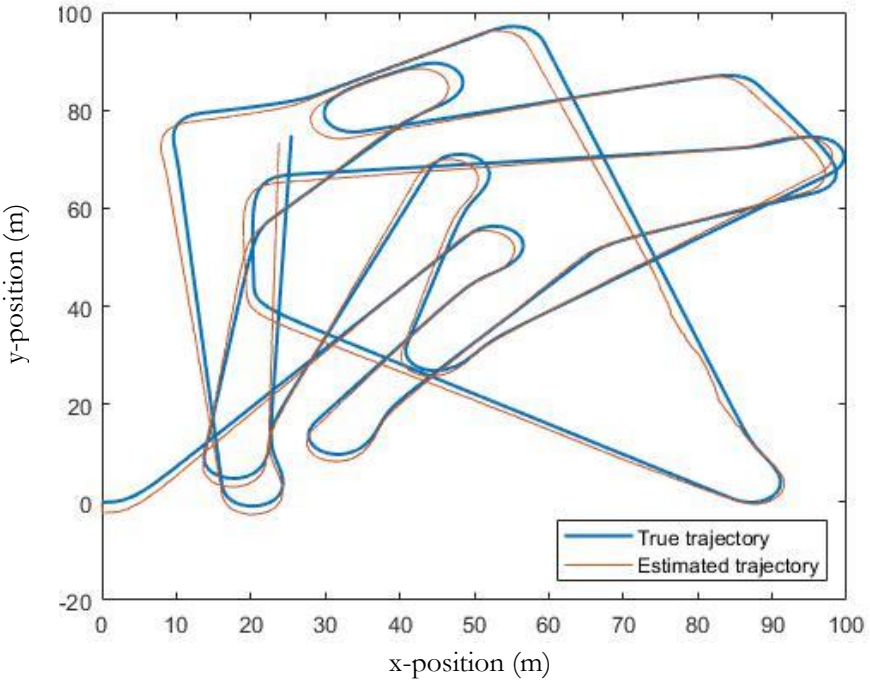


Figure 10 (c): Trajectory for removing all vehicle edges.

The error, covariance, and trajectory plots of all-removed but remain one setting is shown below:

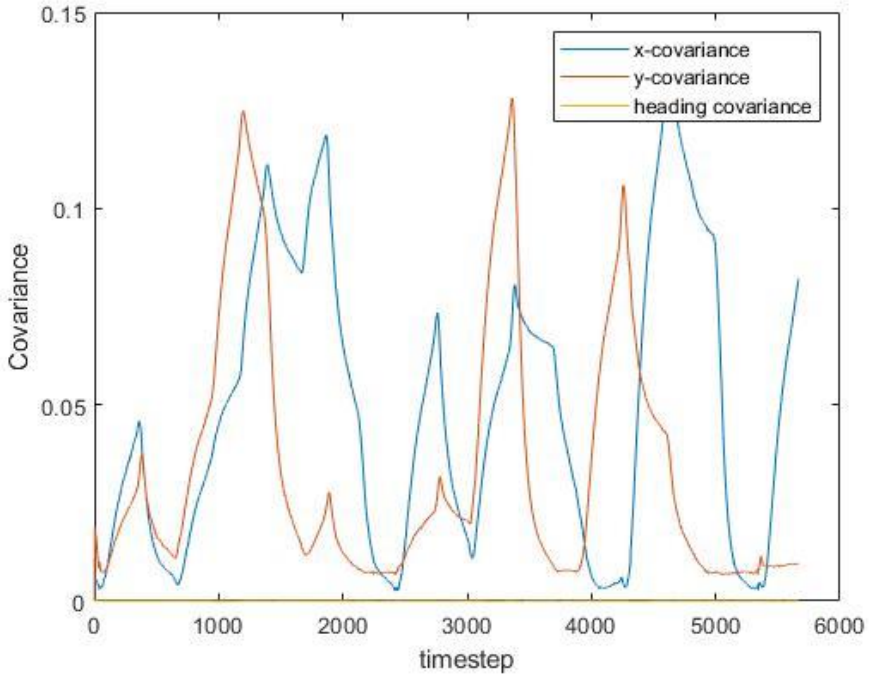


Figure 11 (a): Covariance for remaining first vehicle edge.

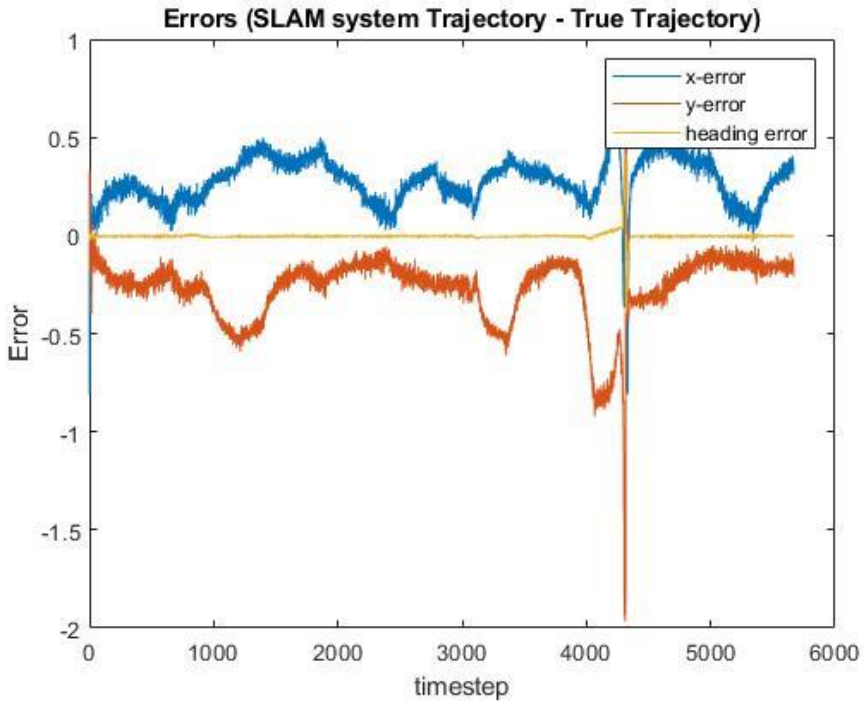


Figure 11 (b): Error for removing all but the first vehicle edge.

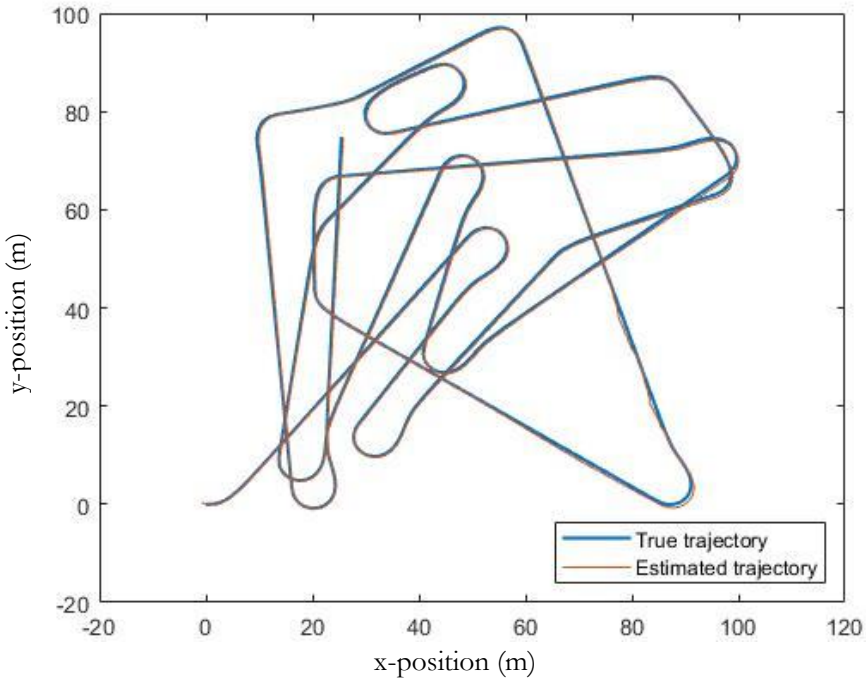


Figure 11(c): Trajectory for removing all but the first vehicle edge.

Based on the figures shown above, we can clearly see the improvement that this algorithm has made. Also, by comparing two cases in this algorithm, remove-all case will fail because when all vehicle prediction edges, are deleted, the initial estimation the SLAM system made is replaced by a vertex. Then the whole estimation will be affected and fail to do a more accurate calculation.

Algorithm 8 Remove all or remove all but first vehicle edge - g²o system

Input: graph, vehicle edges, number of edge remained (remain)

```
numEdges = this.graph.edges():  We first introduce the whole number of edges
for i = 1:length(numEdges)
    current_edge = numEdges{i}
    if current_edge is in vehicle edge:
        vehicle_edge_count += 1
        if vehicle_edge_count == 1 and remain == 1: check if it's first edge and we want to remain.
            disp('First vehicle edge removed')
        Else
            This.graph.removeEdge(current_edge): remove vehicle edge
        end if
    end if
end for
```

Output: None

Question 4c) **Graph Simplification Methods**

Definition of two approaches for simplifying graphs:

Sparse keyframe based SLAM: reconstructs the map over selected frames, which are called "keyframes", and uses an optimization technique called "sparse" which aims to reduce the frames and provides a more accurate localization and semantically rich maps.

Graph Pruning: the reduces the size of the graph through selecting the unnecessary edges that could make decisions on calculations and thus improve the efficiency and accuracy.

In this question, graph pruning was selected as it was hypothesized that it would process much faster, and due to higher familiarity with this algorithm.

To select the successor of the edges of each landmark vertex, two ways were considered: random selection and another a more complex algorithm which selects edges when certain conditions are met. [5] For the randomly selected edges part, the algorithm was designed to randomly delete most of the edges for each landmark to reduce the computational time and some noise. In this coursework, the algorithm was selected to randomly keep two edges for each landmark and remove the remaining edges.

The original plots of error, covariance and trajectory are provided in the previous question and the random graph pruning plots for enable GPS (GPSMeasurementPeriod = 10) and laser (LaserMeasurementPeriod = 0.1) are shown below:

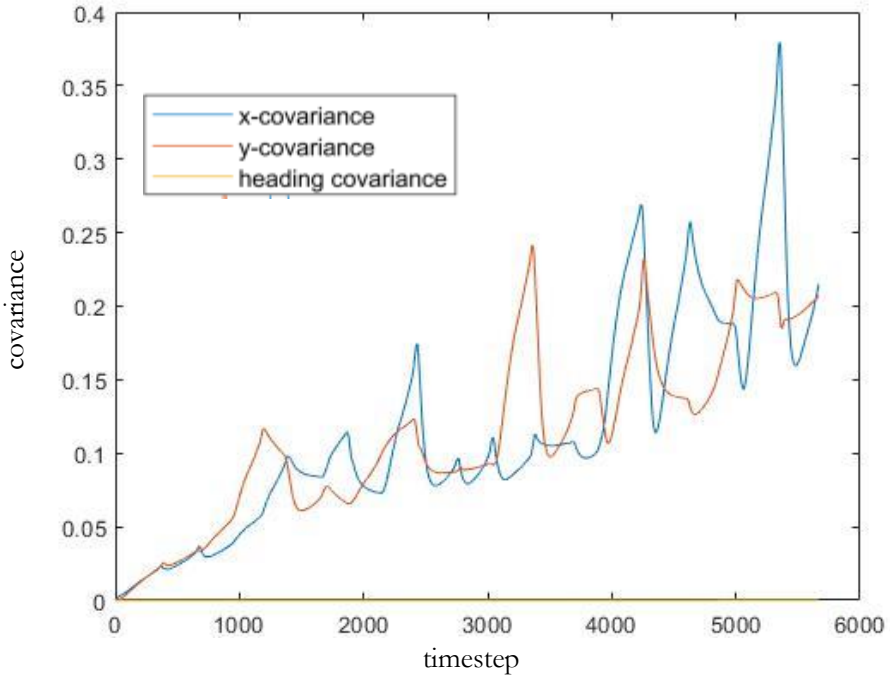


Figure 12(a): Covariance plot for random graph pruning.

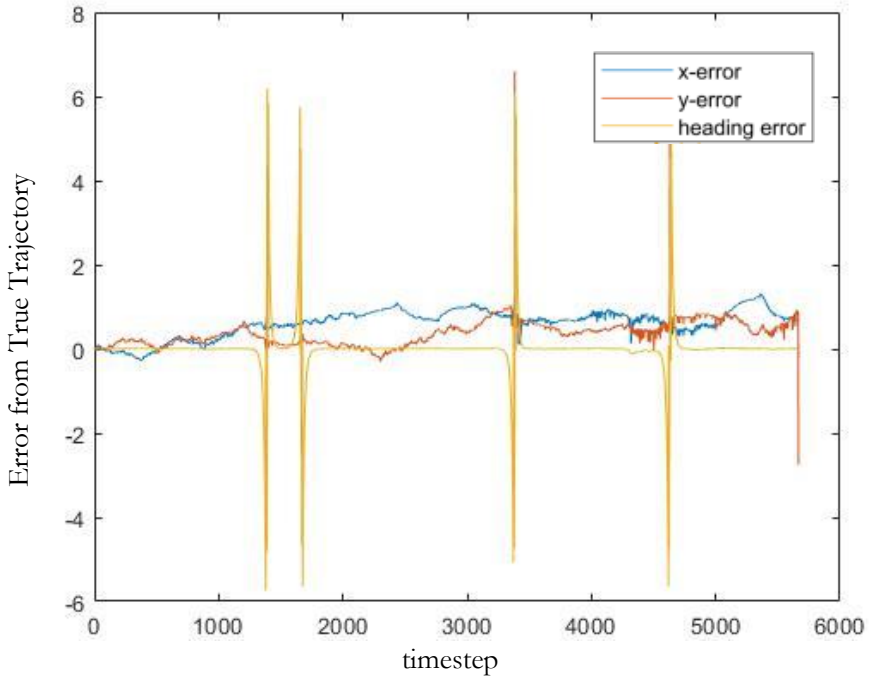


Figure 12(b): Error plot for random graph pruning.

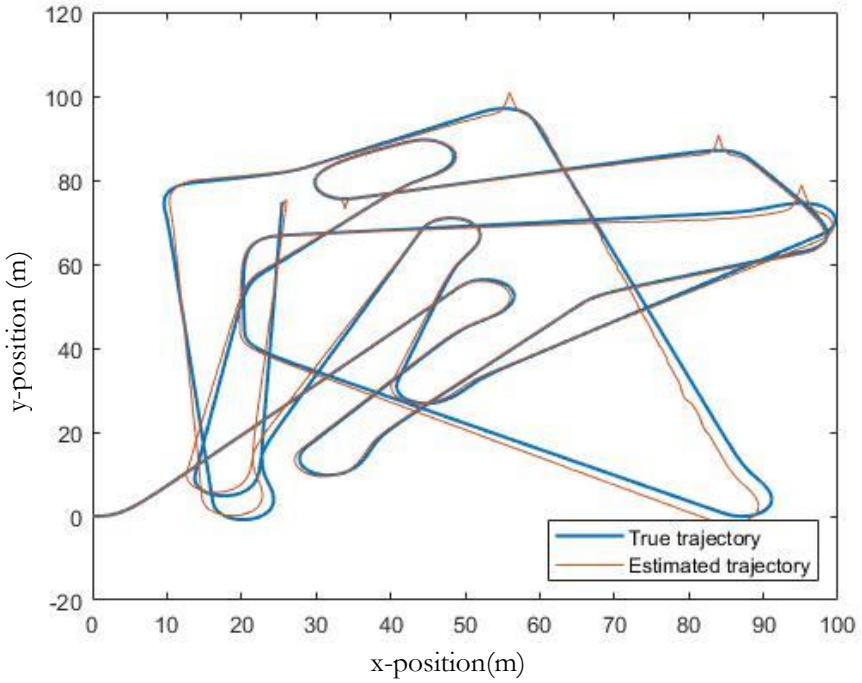


Figure 12(c): Trajectory plot for random graph pruning.

Compared to the original plots in previous question, there is a clear improvement in the random graph pruning performance in terms of average error however the covariance was found to be higher which is expected as there are less edges for the algorithm to optimise.

Algorithm 9 Graph Pruning simple random version (Finished Algorithm)- g²o system

Input: graph, landmark edges, number of edge remained (remain=2)

```
for i = 1:length(this.landmarksMap) : loop through number of landmarks
    current_landmark_edges = this.landmarksMap(i).edges: find each landmark's edges
    if length(current_landmark_edges) < remain:
        deleted_count = length(current_landmark_edges) - remain
        deleted_edge = randsample(current_landmark_edges, deleted_count) : randomly select
        for j = 1:deleted_count
            this.graph.removeEdge(current_landmark_edges{j}) :delete the randomly selected edge
        end for
    end if
end for
```

Output: None

The more complex version of graph pruning is only a proposed algorithm that could filter to keep the most useful edges. However, during implementation of the code, when transferring to the large-map code instability was experienced when iterating. Due to this unresolved inconsistency when running, the results were omitted from the report however the pseudocode for this proposed algorithm is shown below and the unstable code is included in the submission.

Algorithm 10 Graph Pruning complex version (Proposed Algorithm)- g²o system

Input: graph, landmark edges, remain = 1: we only remain the smallest error edge

```
for i = 1:length(this.landmarksMap) : loop through number of landmarks
    current_landmark_edges = this.landmarksMap(i).edges: find each landmark's edges
    for j = 1:length(current_landmark_edges)
        lowest_error = Inf
        if norm(LandmarkMeasurementEdge.ComputeError(current_landmark_edges{j})<
        lowest_error : try to find the lowest error edge for each landmark
            norm(LandmarkMeasurementEdge.ComputeError(current_landmark_edges{j}) =
             lowest_error
            index_of_lowestErrorEdge = j
        end if
    end for
    for k = 1:length(current_landmark_edges)
        if k != index_of_lowestErrorEdge
            this.graph.removeEdge (current_landmark_edges{k}): Remove all the edges except for the
              lowest error one
        end if
    end for
end for
```

Output: None

References

[1] Simon. J, "COMP0130 Coursework 02: Graph-based Optimisation and SLAM", *University College London*, Department of Computer Science, 2021, pp. 7-8.

[2] Simon. J, "COMP0130 Workshop 04: Introduction to Graph-based Optimisation and SLAM", *University College London*, Department of Computer Science, 2021.

[3] Simon. J, "COMP0130: Robotic Vision and Navigation: Lecture Notes", *University College London*, Department of Computer Science, 2021.

[4]P. Cork e, J. Malzahn and J. Haviland, *robotics-toolbox-matlab/RangeBearingSensor.m*. 2017.

[5]Harabor, D. and Grastien, A. , "Online Graph Pruning for Pathfinding on Grid Maps". 2011.