COMP0130 Robotic Vision and Navigation Coursework 2: Graph-Based Optimisation and SLAM*

Yuansheng Zhang†, Joseph Rowell‡& Vanessa Igodifo §

28/03/2022

Question 1

1a

The structure of the graph used to support predicting the robot through time and updating its position periodically with GPS is that of a factor graph. The graphical representation of this probability distribution vertices denote the variables or events of interest. The edges specify conditional probabilities between the variables and events. The graph is a line of state estimates, wheres edges are the process model. A general factor graph example is shown in Figure 1, considering the function with vertices $\mathbf{X} = \{X_1, ..., X_3\}$ and edges $\mathbf{f} = \{f_1, ...f_4\}$

$$g(X_1, X_2, X_3) = f_1(X_1)f_2(X_1, X_2) \times f_3(X_1, X_2)f_4(X_2, Y_3)$$

These factor graphs can be used to simplify the implementation of the Bayes filter, by eliminating the integration during the prediction state using the entire state history. The different types of vertices are sensor poses and 3D landmarks, where the edges are physical relationships between the vertices. Landmark vertices are modelled as 3D points (x, y, z) We can project landmarks to the images taken by the camera and get projected image pixels, for example:

$$\hat{x}_{ij} = K[X_{ij}Z_{ij}]$$

Where $\hat{x_{ij}}$ is the simulated image coordinates as a 2D projection of the 3D landmark Z_{ij} onto the image, and the pose of the camera is X_{ij}. Where X_{ij} is a combination of rotation and translation matrix, and K is the intrinsic matrix of the camera. Then we can compute:

$$e_{ij}^2 = (\hat{x}_{ij} - x_{ij})^2$$

as the error between real measurement and simulated measurement based on current estimation. Each observation edge at the k^{th} iteration is given by Eq.1. Where w_k is the zero mean, Gaussian distributed random variable.

$$\mathbf{z_k} = \mathbf{x} + \mathbf{w_k} \tag{1}$$

In a time varying system, the object state is time varying and the estimation edges are represented by Eq. 2 in the process model.

$$\mathbf{x_{k+1}} = \mathbf{F}\mathbf{x_k} + \mathbf{v_k} \tag{2}$$

The oplus method used in the `vehicleStateVertex.oplus` method accounts for angular discontinuities in edge models. The method allows for the updating of states, wrapping the angles from $0 to 2\pi^c$ continously, so an update in the heading of $2\pi^c$ is equal to 0^c.

*This work was submitted in partial fulfillment of the COMP0130 Robotic Vision and Navigation Module

†yuansheng.zhang.18@ucl.ac.uk
‡joseph.rowell.21@ucl.ac.uk
§vanessa.igodifo.21@ucl.ac.uk

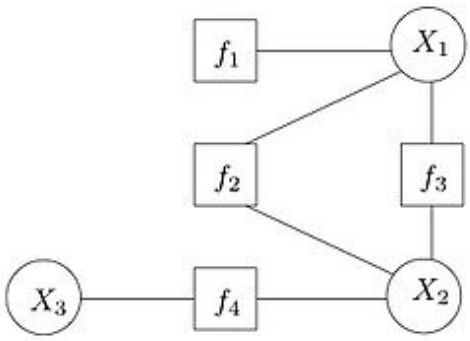

Figure 1: Factor Graph general example [1]

1b

1b - i

To complete the functionality of the predication step within `DriveBotSLAMSystem`, the `handlePredictToTime` was modified, so to predict the state estimate with the estimation process model. The steps undertaken are as follows:

- Predict state with estimation process model
- Add a prediction to the new vertex
- Add a vertex to the graph
- Add edges to the factor graph
- Populated the `VehicleKinematicsEdge.computeError` method

$$Error = \begin{bmatrix} cos(x) & sin(x) & 0 \\ -sin(x) & cos(x) & 0 \\ 0 & 0 & 1 \end{bmatrix} \frac{(x_{k+1} - x_k)}{dT - z} \tag{3}$$

- Populated the `VehicleKinematicsEdge.linearizeOplus` method
- Populated the `VehicleStateVertex.oplus` method

1b - ii

The Figure 2 shows the covariances of the vehicle throughout time, and the x,y covariances are oscillating and increasing in magnitude, this is due to greater uncertainty throughout time, as there is no loop closure. The heading covariances are very low as there is little uncertainty. The Figure 2 shows increasing covariances as there is no closed loop feedback from error correction. The errors shown in position in Figure 3 shows position errors iterating in a pattern due to the square path taken, and the error in the heading is very little.

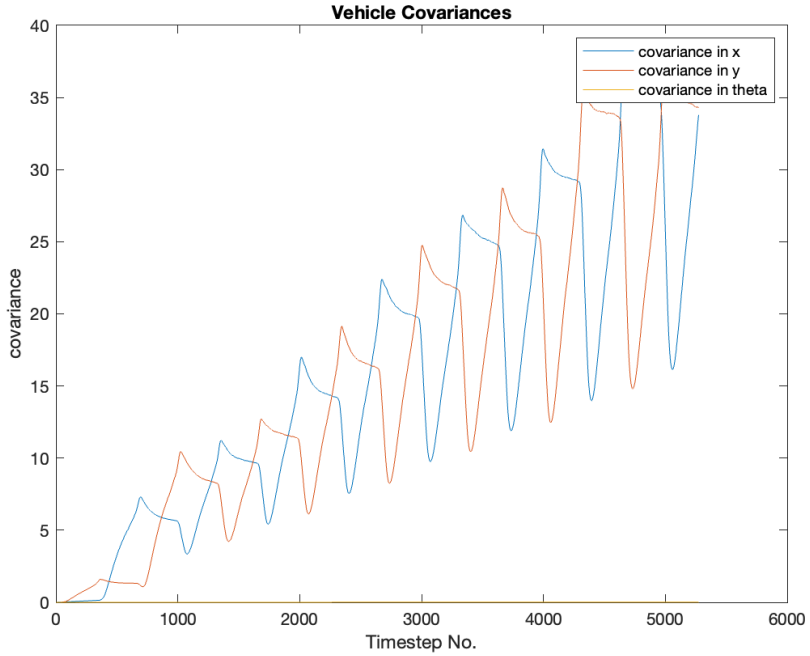

Figure 2: Covariance Graph

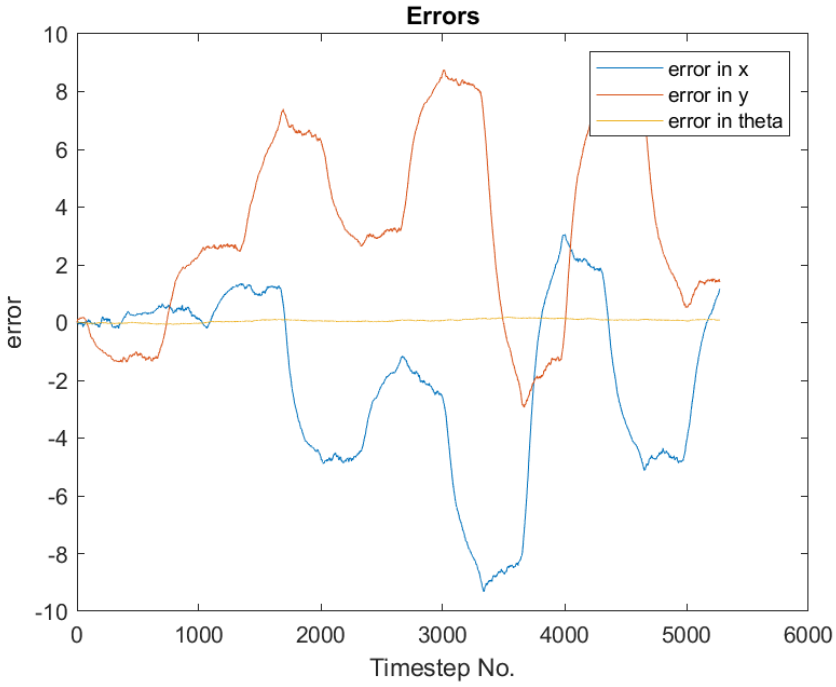

Figure 3: Errors Graph

1c

1c - i

The methods edited in the GPS update step in `DriveBotSLAMSystem` were the `DriveBotSLAMSystem.handleGPSObservationEvent` method, and the `drivebot.GPSMeasurementEdge`

class and internal methods. The `handleGPSObservationEvent` method works as follows:

- Create a GPS measurement edge
- Store event covariance and vehicle vertex ID
- Link the edge so it connects to the vertex we want to estimate
- Set and store the measurement value and measurement covariance
- Add the edge to the graph

The `computeError` method in class `GPSMeasurementEdge` computes the error by finding the difference between the estimate and the known value.

1c - ii

The covariances in Figure 4 are small compared to runs with no GPS data, oscillating at roughy 0.18, as GPS landmark data decreases uncertainty in the model. Covariance in heading theta is almost zero as there is little uncertainty in the heading. There is a large hockey stick overshoot at the end of the run in covariances, this could be due to observing the same landmark for a second time. Errors are also greatly reduced with the introduction of GPS landmark observations, this is expected as GPS provides information about position.

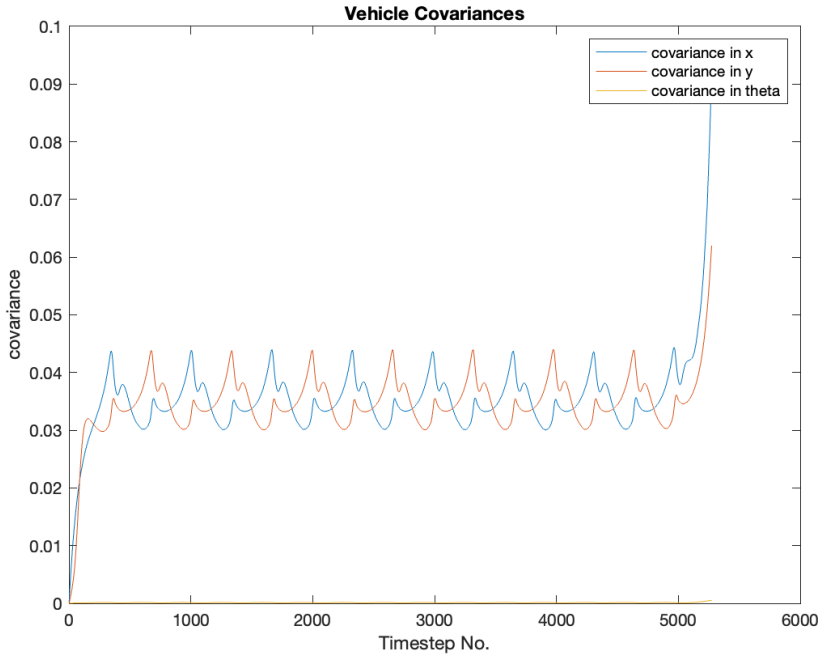

Figure 4: Covariance Graph

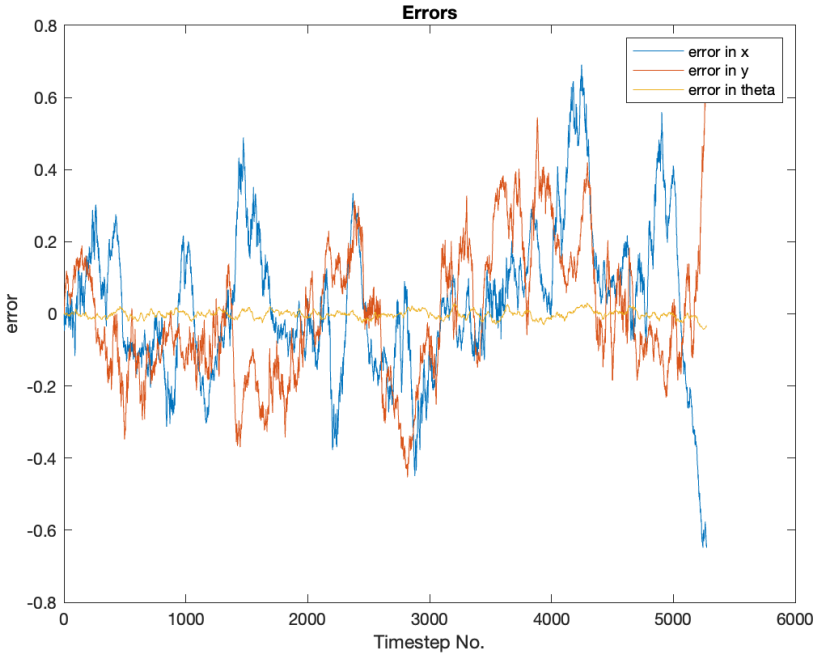

Figure 5: Errors Graph

1d

The Figures 6, 7, 8, 9 show the optimisation times, covariances, errors and Chi2 values for q1d respectively. The script `q1_d.m` was modified to make the optimizer run once every timestep instead of at the end of the run. This gives a lesser oscillation in the covariances and errors, however a similar magnitude. The Chi2 values model the probability of the absolute value of the deviation of the measurement from it's expected value, allowing the modelling of the probability of a certain landmark providing the desired measurement. They increase over each timestep, as do the covariances. The optimisation time increases linearly with timestep, this is due to the complexity of the Levenberg-Marquardt optimisation algorithm increasing as the Hessian is calculated at each timestep. In Figure 6, there are two trends in the optimisation times, a linearly increasing trend due to gradually increasing complexity of inverse hessian calculations, and a much lesser increase in optimisation times when a knwon landmark is observed. As a new landmark requires more complex optimisation, and thus longer optimisation time, however multiple observations of the same landmark do not.

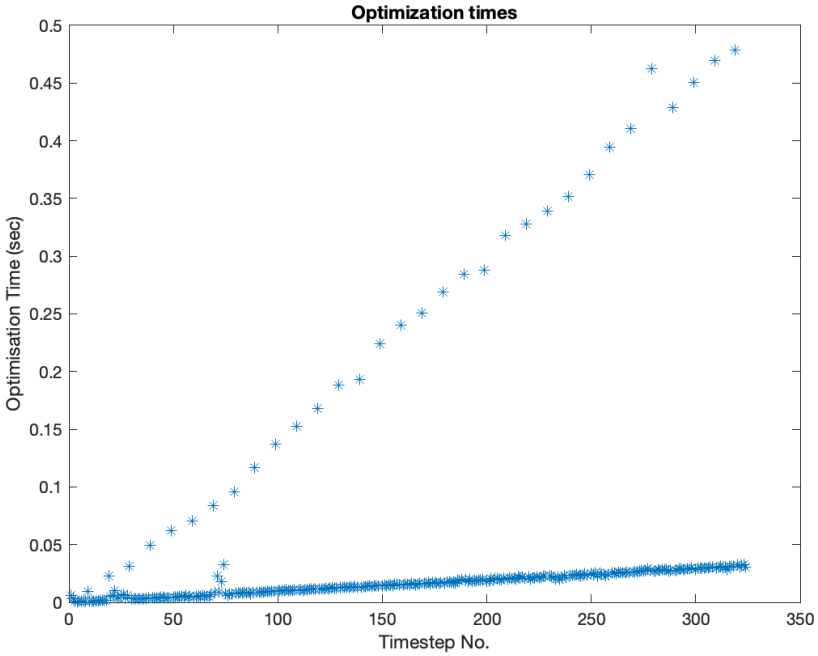

Figure 6: Optimisation times

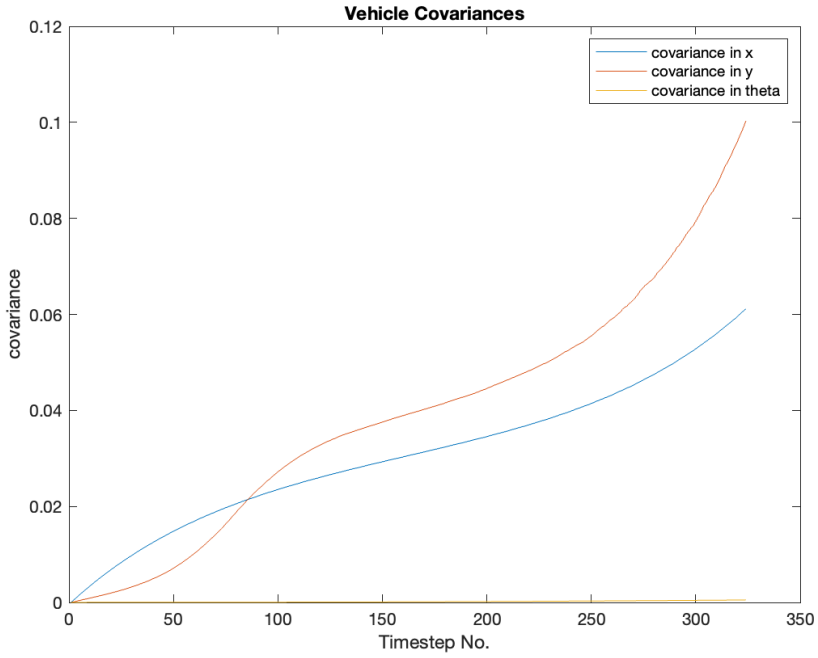

Figure 7: Covariance Graph

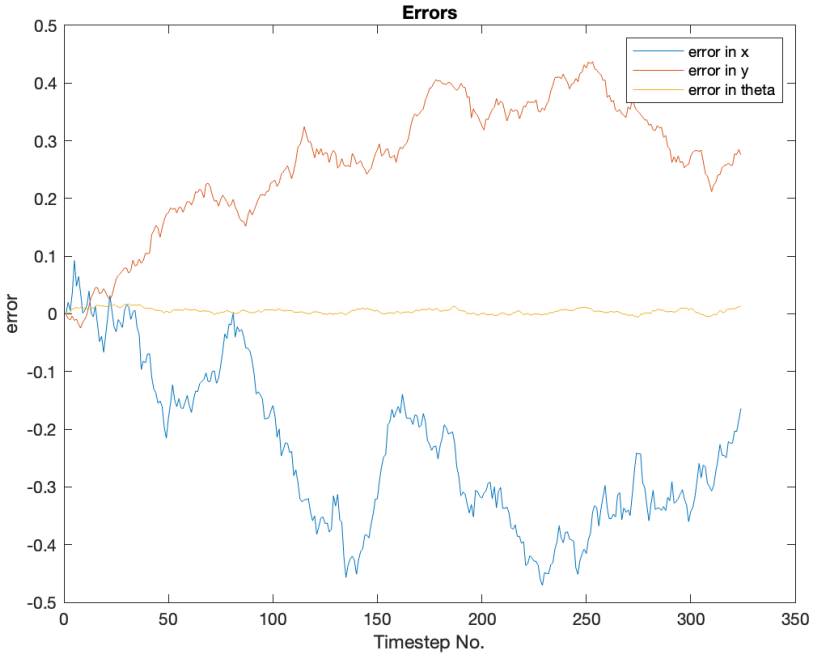

Figure 8: Errors Graph

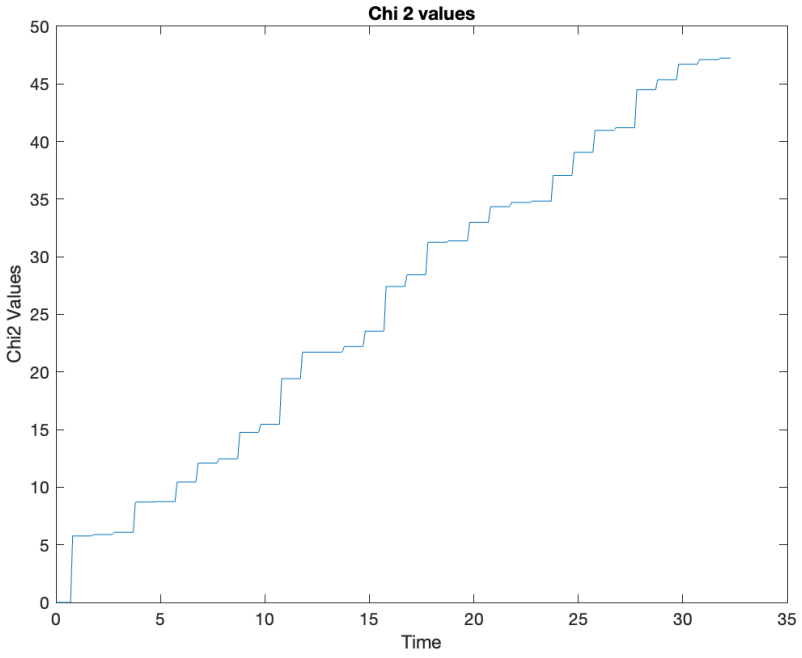

Figure 9: Chi2 Values Graph

Question 2

2a

The graph consists of a sequence of nodes representing the vehicle over time, these vertices are directly connected to one another by prediction edges f_n, and observation edges h_n. The measurement edges are created from GPS sensor inputs, and the observation edges are created by the landmark observations. The error model is calculated in method `LandmarkRangeBearingEdge.computeError`, and consists of the norm of the landmark - states - GPS. Error models in the system are governed by the GPS observation model and the Landmark Observation model, as shown in equations below respectively.

$$\mathbf{z_k}^G = \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \mathbf{w_k}^G$$

Where the covariance of $\mathbf{w_k}^G$ is diagonal and constant.

$$\mathbf{z_K}^L = \begin{bmatrix} r_k^i \\ \beta_k^i \end{bmatrix} + \mathbf{w_k^L}$$

Where

$$r_k^i = \sqrt{(x^i - x_k)^2 + (y^i - y_k)^2}$$

$$\beta_k^i = tan^{-1}(\frac{y^i - y_k}{x^i - x_k}) - \phi_k$$

As the problem is minimised by damped least squares (LM), the error model can be described as follows:

$$F(x) = \sum_{k \in C} \mathbf{e_k}(x_k, z_k)^T \Omega_k \mathbf{e}_k(x_k, z_k)$$

$$\mathbf{x}^* = argmin_x F(x)$$

Error function represented by its first order Taylor expansion around current initial guess $\hat{x}$:

$$\mathbf{e_k}(\hat{x}_k + \Delta x_k) = e_k(x_k \hat{+} \Delta x) = e_k + J_k \Delta x$$

Where J_k is the jacobian needed to implement the error model, shown in the methods in the code listing below.

```
function computeError(this)
            %%%%%%%%%% Q2b %%%%%%%%%%
            x = this.edgeVertices{1}.estimate();
            landmark = this.edgeVertices{2}.estimate();
            dx = landmark(1:2) - x(1:2);

            this.errorZ(1) = norm(dx) - this.z(1);
            this.errorZ(2) = g2o.stuff.normalize_theta(atan2(dx(2), dx(1)) - x(3) -
    this.z(2));

            %warning('landmarkrangebearingedge:computeerror:unimplemented', ...
            %    'Implement the rest of this method for Q1b.');
        end

        function linearizeOplus(this)
            %%%%%%%%%% Q2b %%%%%%%%%%
            x = this.edgeVertices{1}.estimate();
            landmark = this.edgeVertices{2}.estimate();
            dx = landmark(1:2) - x(1:2);
            r = norm(dx);

            this.J{1} = ...
                [-dx(1)/r -dx(2)/r 0;
                dx(2)/r^2 -dx(1)/r^2 -1];
            this.J{2} = - this.J{1}(1:2, 1:2);

            %warning('landmarkrangebearingedge:linearizeoplus:unimplemented', ...
            %    'Implement the rest of this method for Q1b.');
        end
```

Listing 1: Error calculation and Jacobian

The Oplus method is used to counter the discontinuities in error calculation, as this takes account of angle wrapping in the heading angle, so there are not spikes of 2π in the error graphings.

2b - i

The methods implemented in `DriveBotSLAMSystem` to complete functionality to support SLAM are the `drivebot.LandmarkRangeBearingEdge` & `DriveBotSLAMSystem.handleLandmarkObservationEvent` Method `DriveBotSLAMSystem.handleLandmarkObservationEvent` iterates through all landmark measurements, and gets the associated vertex for each measurement. If there is no associated vertex, a new one is created and added to the factor graph. The class `LandmarkRangeBearingEdge` contains methods implemented `initialize`, `computeError` and `linearizeOplus`. The method `initialize` initializes the estiate of the landmark given vehicle pose, and r, β measurements. The method `computeError` computes the error in landmark estimation at each iteration.

2b - ii

The Figures 10, 11, 12, 13 show the optimisation times, covariances, errors and Chi2 values respectively. The time taken to complete and optimisation step is shown to be increasing with each timestep, with the exception at 3 points. At these points, loop closure has occurred, and thus the optimisation time is much less. The heading covariances are very small due to small error. The covariances show a pattern of being greatest when furthest away from the starting point, as the uncertainty is greatest at this point, before loop closure has occured. The chi2 values show are increasing as the the Chi2 values model the probability of the absolute value of the deviation of the measurement from it's expected value, the chi2 values increase in each timestep apart from the timesteps where loop closure has occured and they stay constant, simultaneously the optimisation time is reduced.

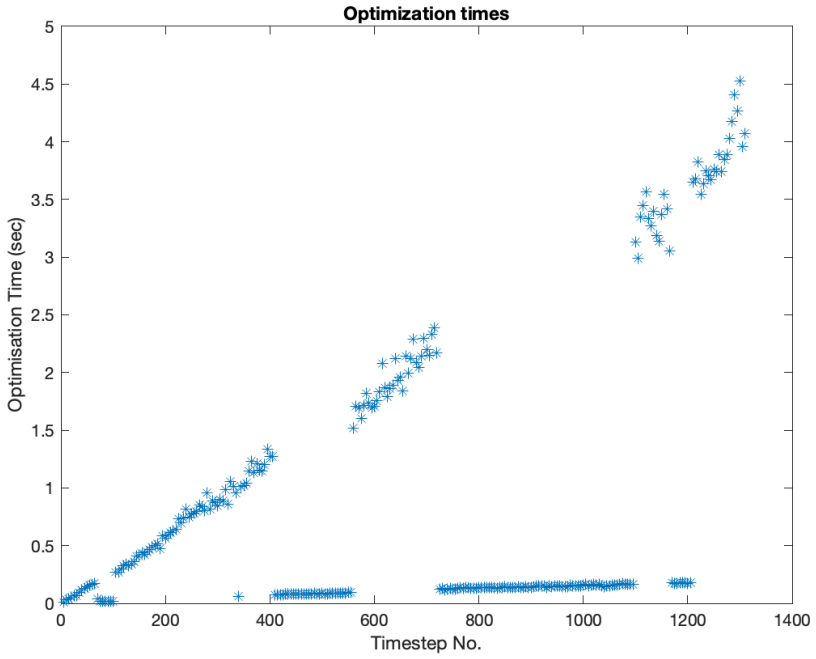

Figure 10: Optimisation times

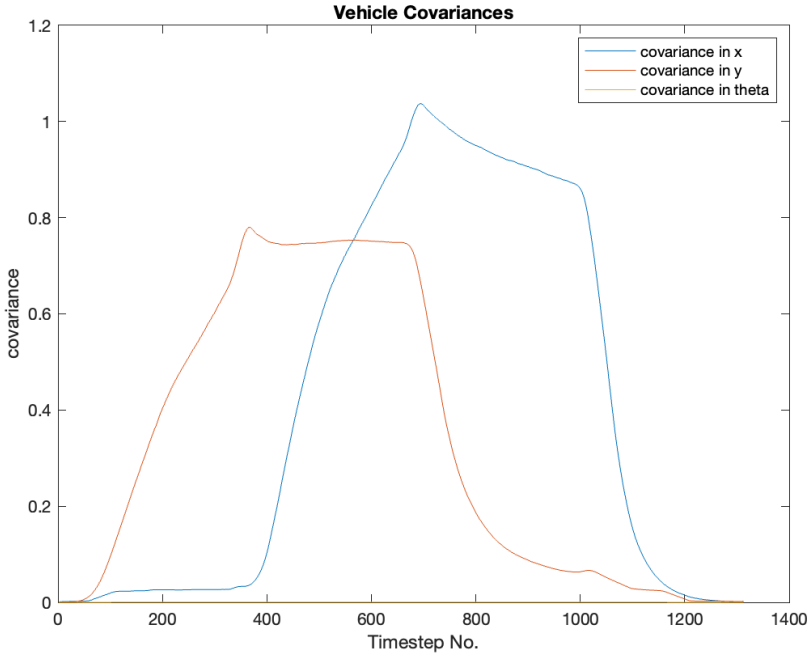

Figure 11: Covariance Graph

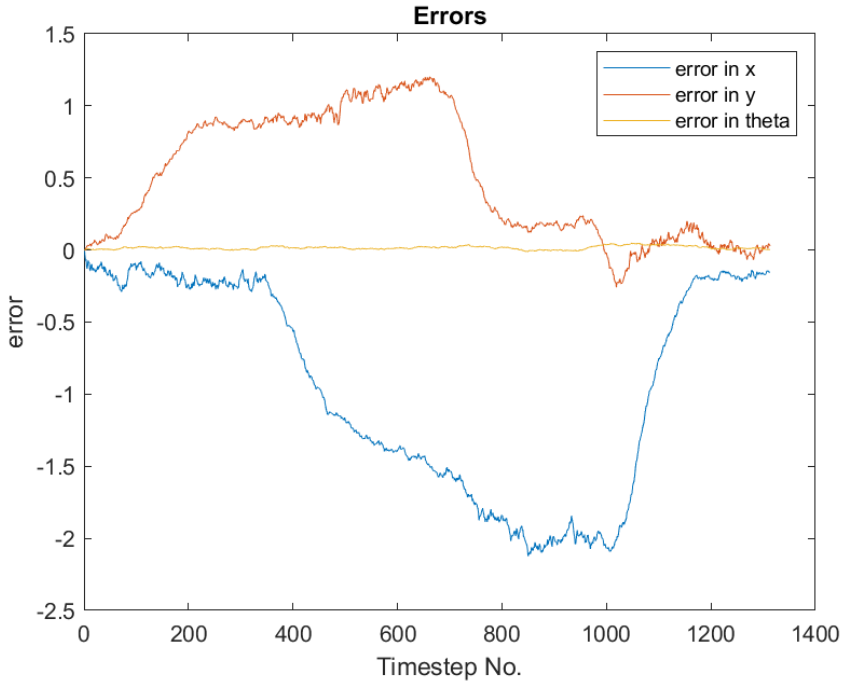

Figure 12: Errors Graph

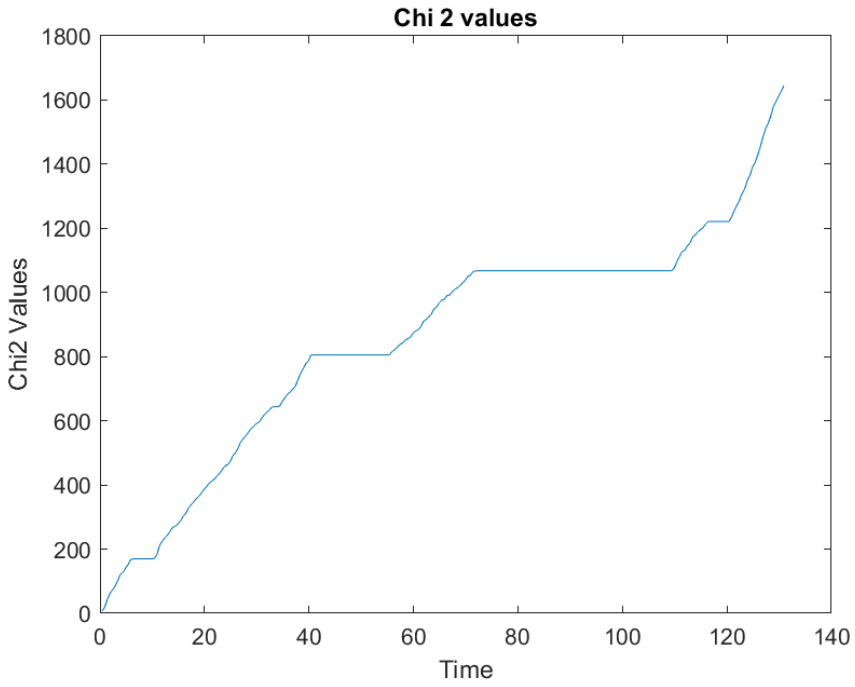

Figure 13: Chi2 Values Graph

2c

2c - i

The number of vehicle poses stored, number of landmarks initialised, average number of observations made by a robot at each time-step and average number of observations received from a landmark were calculated as follows:

```
% total number of vertices
numVertices = length(allVertices);

% total number of edges
numEdges = length(allEdges);
% number of vehicle poses stored
numVehPoses = length(results{1}.vehicleStateHistory');
fprintf('Total number of vehicle poses stored: %.0d. \n', numVehPoses)

% number of landmarks initalized
numLmVertices = numVertices - numVehPoses;
fprintf('Total number of landmarks initalized: %.0d. \n', numLmVertices)

% average number of observations made by a robot at each timestep
numObsVerEdge = numEdges - numVehPoses;
numObsPerDT = numObsVerEdge / numVehPoses;
fprintf(['Average number of observations made by a robot at' ...
    ' each timestep: %.2d. \n'], numObsPerDT)

% the average number of observations received by each landmark
numObsPerLM = numObsVerEdge / numLmVertices;
fprintf(['Average number of observations received by' ...
    ' each landmark: %.2d. \n'], numObsPerLM)
```

Listing 2: 2C

2c - ii

Total number of vehicle poses stored: 5273. Total number of landmarks initialized: 7. Average number of observations made by a robot at each timestep: 0.643. Average number of observations received by

each landmark: 484. These quantities imply that there are not observations made at every timestep, and the timestep is small enough to have sufficient resolution.

2d

The Figures below show the map, vehicle and landmark states before and just after loop closure events, the graphs show that after closure, the vehicle covariances are much smaller than before, as uncertainty is reduced greatly. Determineat of covariance matrix before loop closure is shown in Figure 14, where the large diameter of the covariance determinant is shown in blue. Determinant of covariance matrix after loop closure is shown in 15 where the same landmarks have much smaller diameter covariance determinant. The largest diameter covariance determinant before loop closure is roughly 15 units, whereas after loop closure is 2 units for the same landmark, thus the covariance determinant has reduced by 13 units for that landmark. Therefore the uncertainty for each landmark is reduced when loop closure is performed.

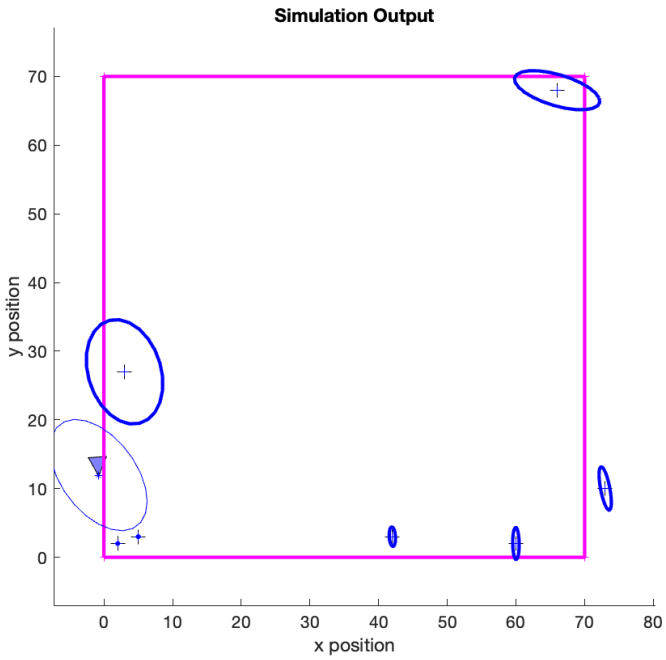

Figure 14: Simulation covariance determinant before closure

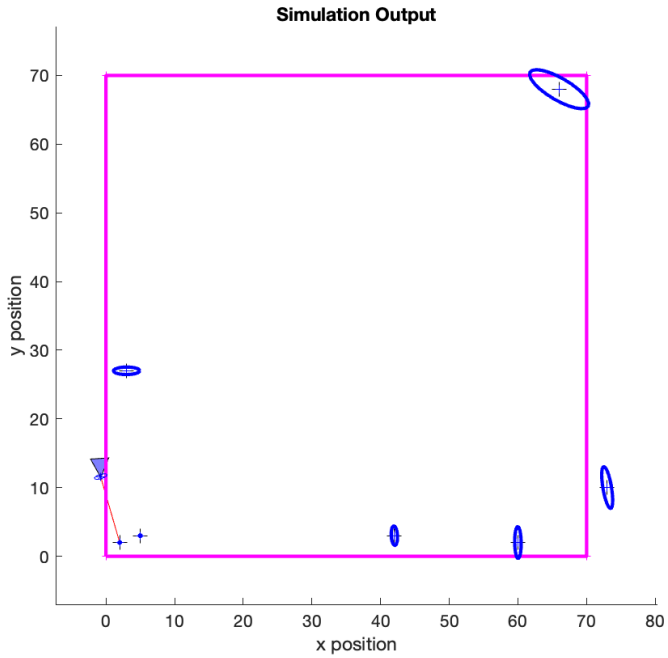

Figure 15: Simulation covariance determinant after closure

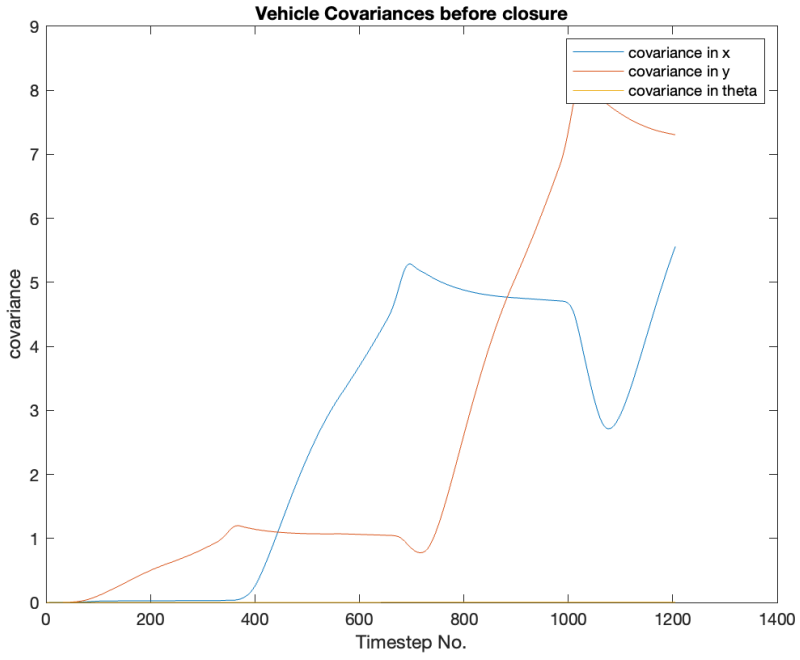

Figure 16: Covariance Graph_before_closure

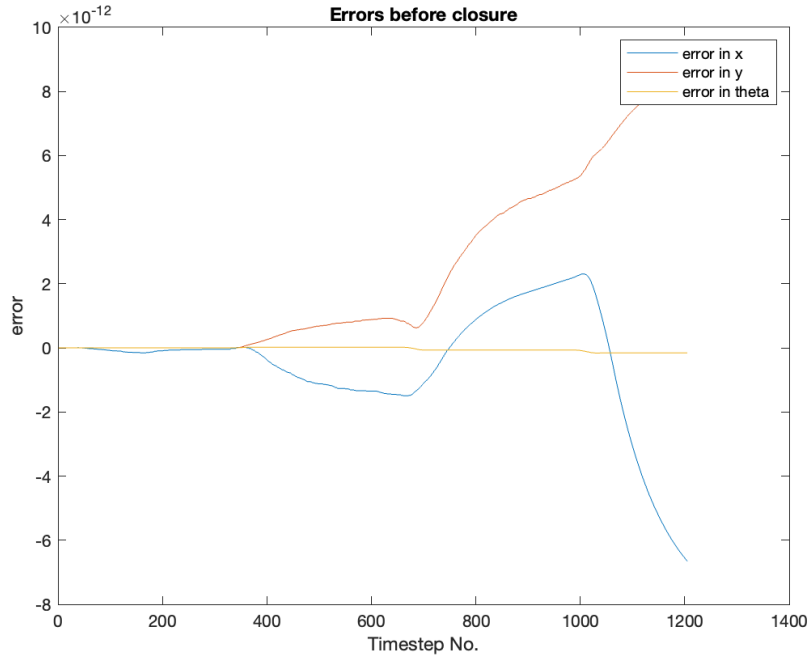

Figure 17: Errors Graph_before_closure

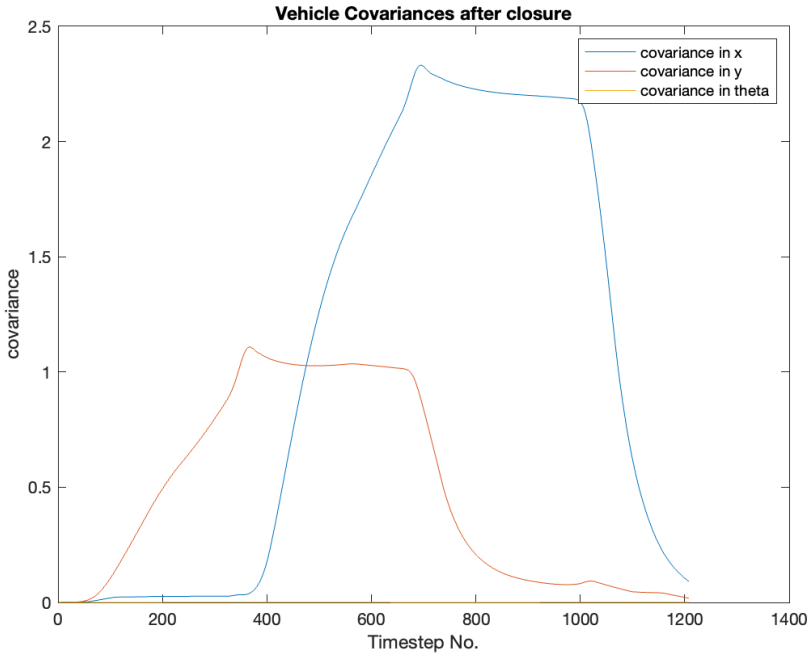

Figure 18: Covariance Graph_aft_closure

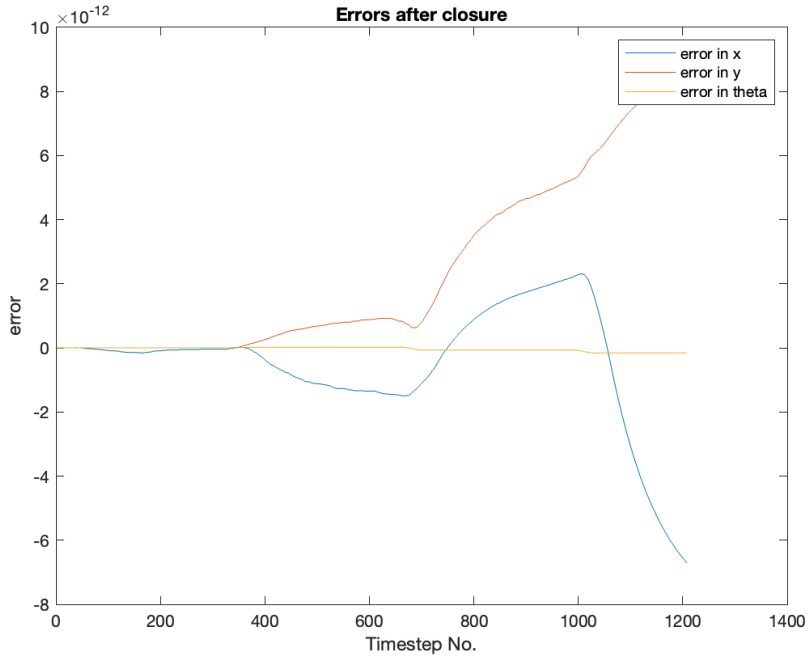

Figure 19: Errors Graph_aft_closure

Question 3

3a

3a - i

Referring to the graph in Figure 20, the prediction edges are labelled f_n. Every edge in the graph has associated prediction or measurement uncertainty. The process of data association involves coupling sensor observations with map elements [2]. Occasionally, data association can be done incorrectly, meaning our system confuses which landmarks are being observed, which can in turn create errors in the platform pose estimates. In these cases, errors must be detected and erroneous prediction edges removed so as to prevent errors from propagating through the system and affecting future vehicle pose estimates. Identification of previously seen landmarks may also be affected and 'new' landmarks may be falsely created. During loop closure, false constraints can arise as a result of these incorrect position estimates. Removal of all vehicle prediction edges (except the prior edge) in this case may be beneficial as we can rely solely on landmark observation measurements to aid in vehicle pose estimation, and false constraints provided by incorrect data association will be detected and removed, so that the information they provide cannot negatively affect the structure of the map being built.

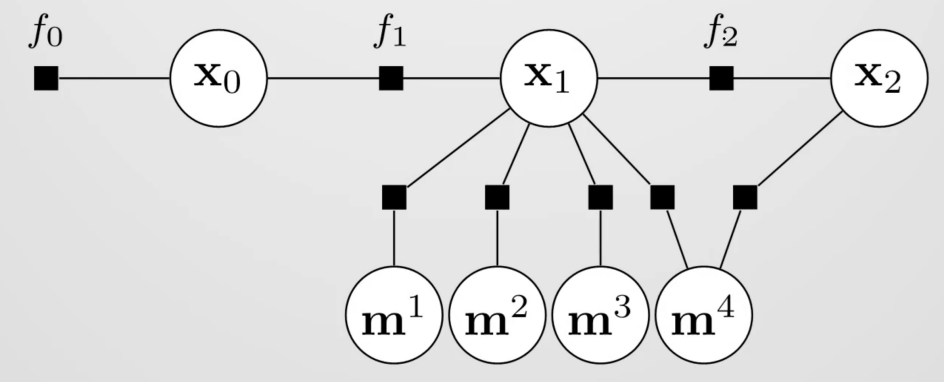

Figure 20: Full Factor Graph

Removal of edges will be unsuccessful if the prior edge is removed, as this is required to compute the posterior trajectory of vehicle pose estimates. It would also be unsuccessful when we have a sparse graph, where there are too few sensor measurements to be able to extract a reasonable vehicle pose estimate from them.

3a - ii

For this question, we edited the `deleteVehiclePredictionEdges()` method of the `DriveBotSLAMSystem` class. The boolean attribute which checks if prediction edges are to be deleted or not has been instantiated outside of this function, so this function is only run if this attribute is set to true — specifying for edges to be deleted. The algorithm first finds the total number of edges in the graph. It then loops through all of these edges and checks if they are a prediction edge or not. Once a prediction edge has been found, a counter is incremented so we can keep track of how many prediction edges have been identified. The next conditional statement checks if the function input requires the first edge to be retained or not. If it needs to be kept, we move onto the next prediction edge and begin to remove each subsequent edge one by one. If we have specified that the first edge is to be removed also, this edge is removed similarly, but the algorithm fails in this case.

3a - iii

In the first case, all prediction edges are maintained (see Figure 21).

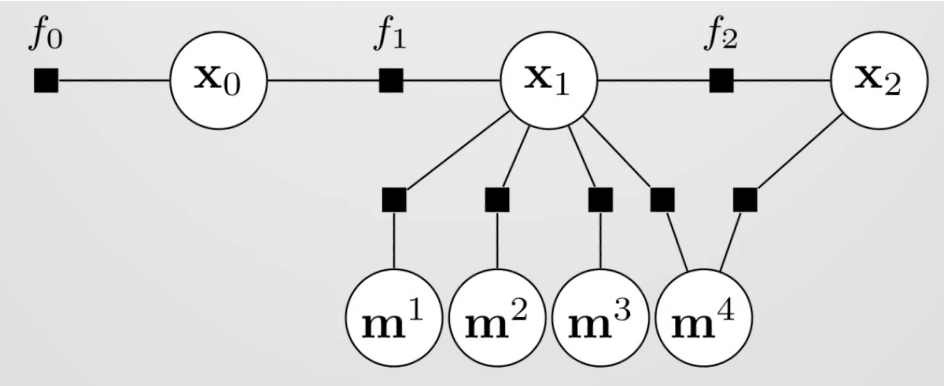

Figure 21: Case 1: all edges maintained

In the second case, all prediction edges are removed (see Figure 22).

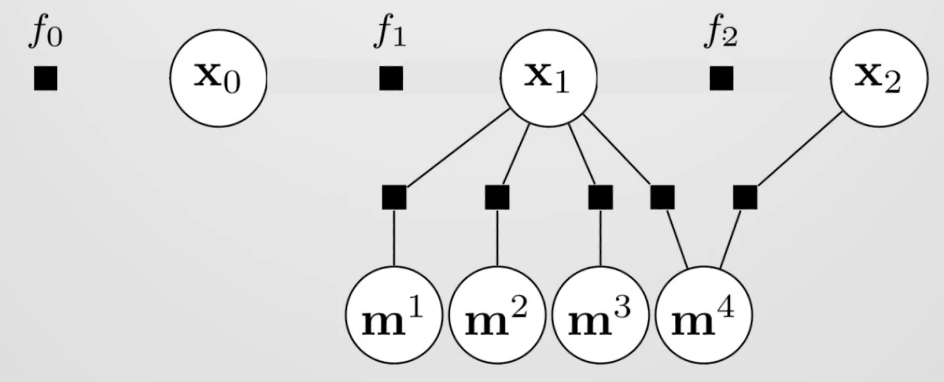

Figure 22: Case 2: all edges removed

In the third case, all but the first prediction edges are removed (i.e. only the first prediction edge is retained - see Figure 23).

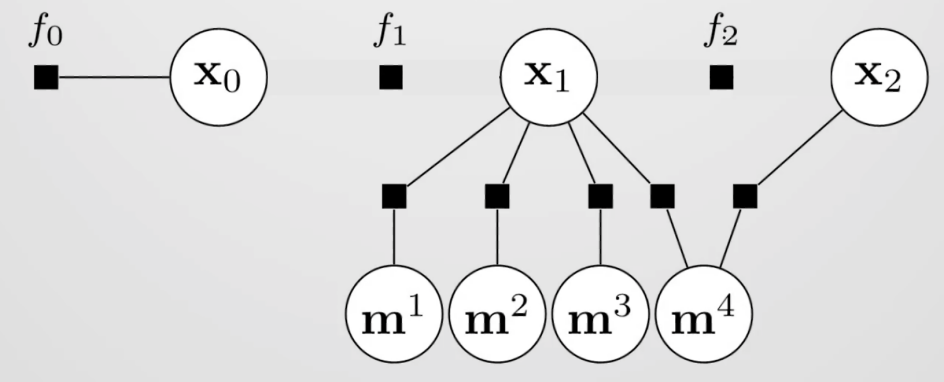

Figure 23: Case 3: all but the first edge removed

The first prediction edge represents the prior on the initial state $\mathbf{x_0}$ — it is a unary edge with a factor, f_0 built into it. $f(\mathbf{x_0})$ is a probability distribution dependent on this single vertex. The goal is to build a graph to describe this starting state, i.e. estimating the posterior probability of the robot's trajectory [3]. This unary edge provides a soft constraint on where this starting position and orientation of the vehicle can be. Prediction edges in general represent a probability distribution of the vehicle's motion from one pose to the next in the subsequent time step [4]. Updates in the vehicle pose are obtained through the minimisation of the error function associated with the edge between two pose vertices. This first (prior) edge, although its initial value is not relevant, provides a sort of 'anchor' for the system, and without which, optimisation in this form cannot take place (hence why case 2 causes the algorithm to fail). This is because the initial constraint on vehicle pose is lost, and the localisation problem becomes largely overcomplicated, if not impossible. Contrary to Kalman filters, graphical models can be implemented to avoid integration in the Bayesian filtering technique by instead performing maximum likelihood estimation for the position estimate in the update step. However, for this to be implemented, knowledge of a prior is necessary to solve the posterior according to Bayes' rule.

In reference to Figures 24 and 25, we can see that removing all prediction edges, except the first, causes an increase in the position errors of vehicle pose, which was to be expected since observations are corrupted by observation noise, so without spatial constraints on the vehicle pose at a given time step (dependent on the previous pose estimates) to describe the transformation between poses, errors can propagate through the system [3]. These constraints can come from odometry measurements, which is why the system does not fail when all prediction edges are removed as long as the measurement edges are retained. However, with more constraints, a higher certainty in vehicle pose at a given time step can be obtained, and the task of optimisation and determination of the most likely configuration of

vehicle pose given edges present is more accurate.

The low error in the bearing estimate of the vehicle pose is assumed to be due to a low error in the 2D range bearing sensor.

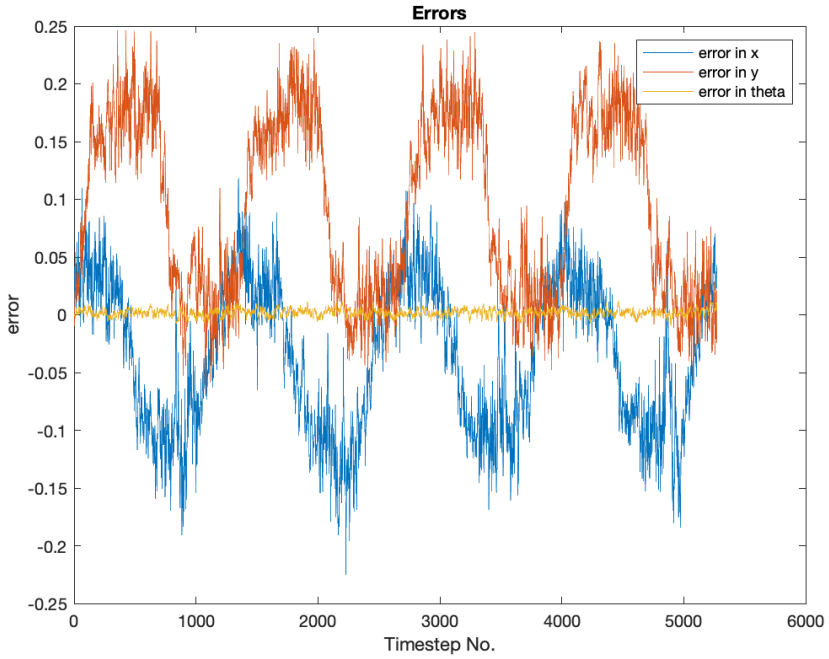

Figure 24: Case 1: Errors

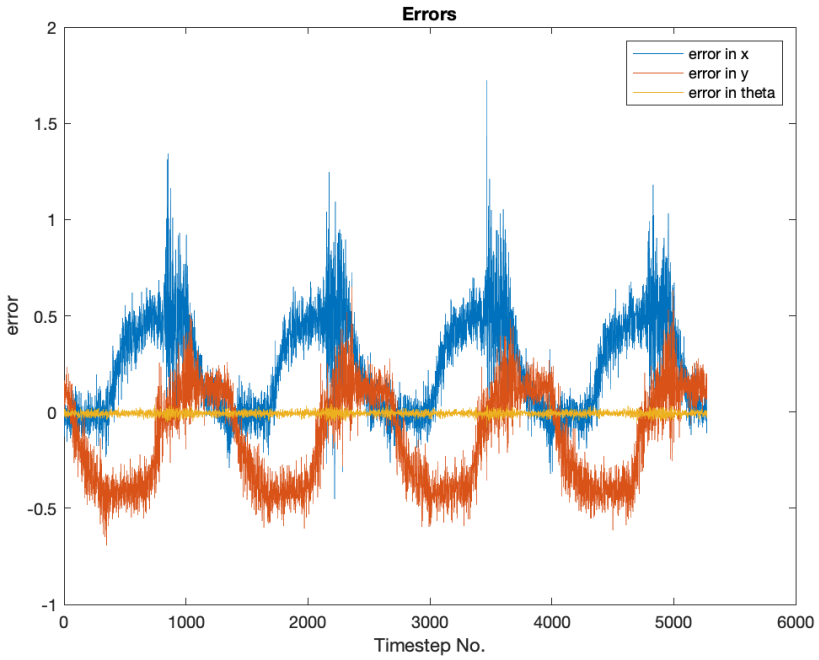

Figure 25: Case 3: Errors

3a - iv

Optimisation times in both cases grows linearly over time steps as we consider the computation of the Levenberg-Marquardt equation during optimisation. After removal of edges and decreasing the density of the graph, optimisation time at each time step was shorter than the case of the full SLAM system.

We observe regular pulses in the vehicle covariance, where it increases in between optimisation steps and falls again where observations are recorded. The uncertainty builds up gradually between time steps due to process noise, measurement noise, and inherently present noise in the sensor, which is shown by an increase in covariance. This prediction error builds into the system, but can only be seen in this way when we observe infrequently, where in this case, we observe every 500 time steps. A high covariance means we are more uncertain about the platform's pose at that particular time step, and it can be seen that the highest uncertainty is recorded in between observations. A smoothing algorithm as such allows landmark observation information effectively from both the past and the future to be used to estimate the current pose, which is why the covariance decreases when observations are observed, as when we receive an observation we have high certainty regarding the pose. In Figure 21 we can see that the maximum covariance recorded in between observations is significantly lower than that seen in Figure 23, where all except the first prediction edges have been removed. We can also see that for Figure 4 the covariance increases and doesn't fall at the end of the experiment, as there is no final observation to consider here.

The Chi2 values show by how much the measurement of a value (vehicle pose) deviates from its expected value, which allows us to assess the probability of the observation measurement we receive from a particular landmark being in line with what we expect. It can be seen in Figure 27 that the values are very similar that of the full SLAM system shown in Figure 26.

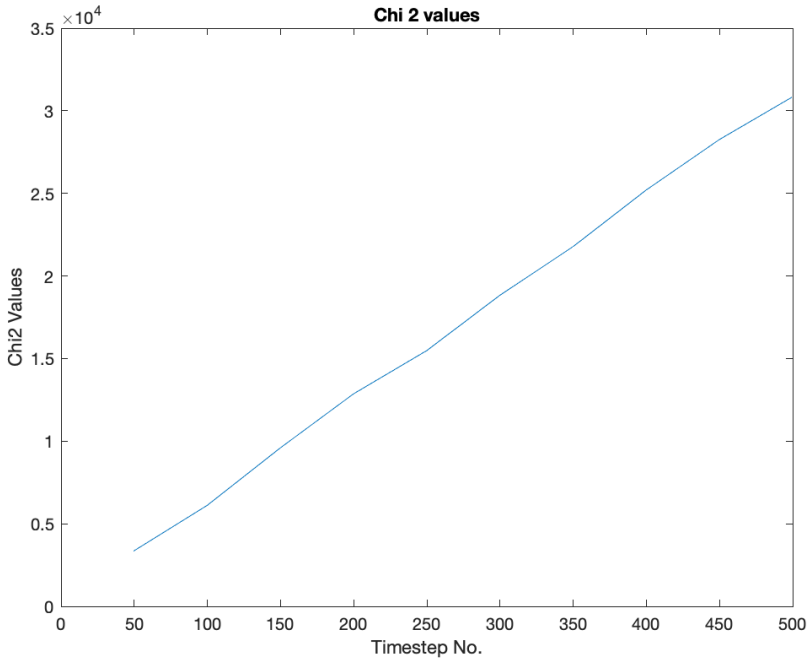

Figure 26: Case 1: Chi2 Values

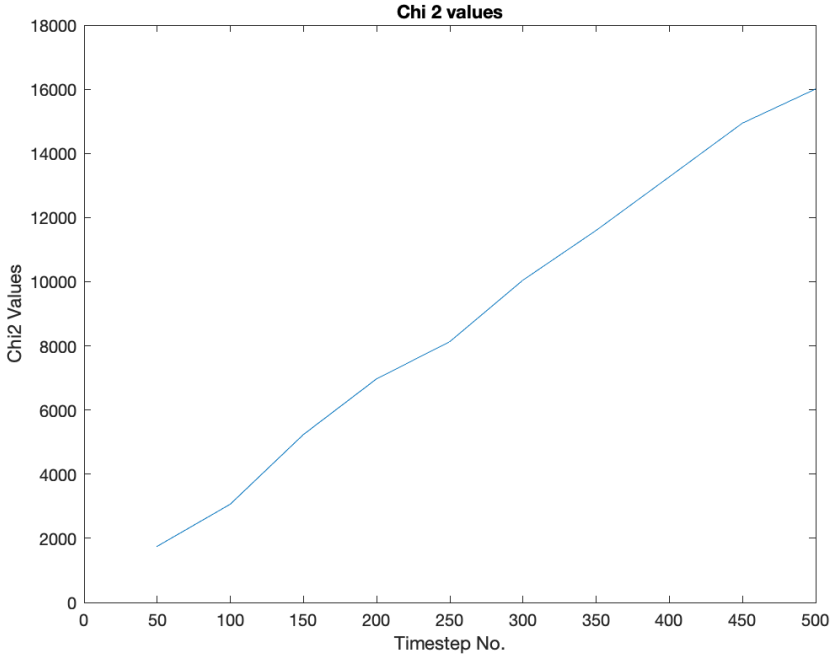

Figure 27: Case 3: Chi2 Values

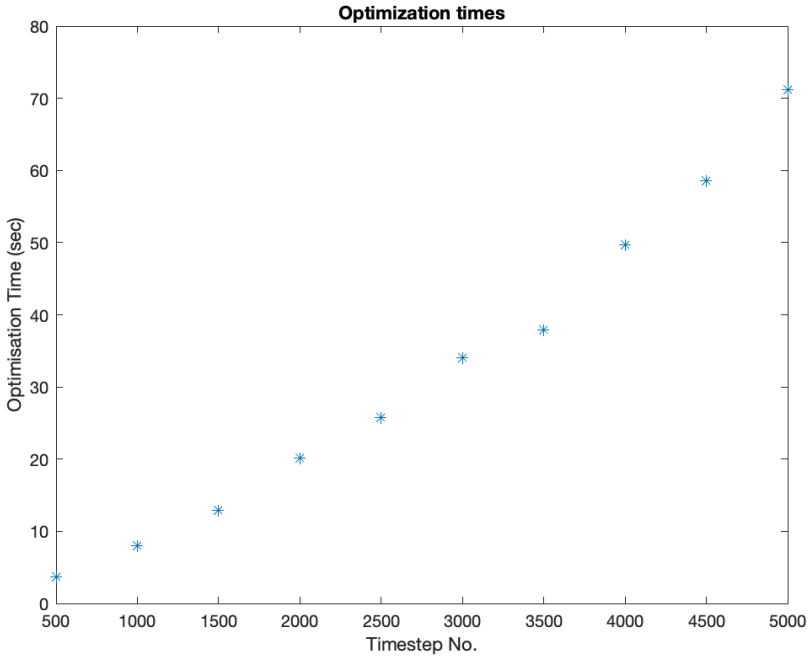

Figure 28: Case 1: Optimisation Times

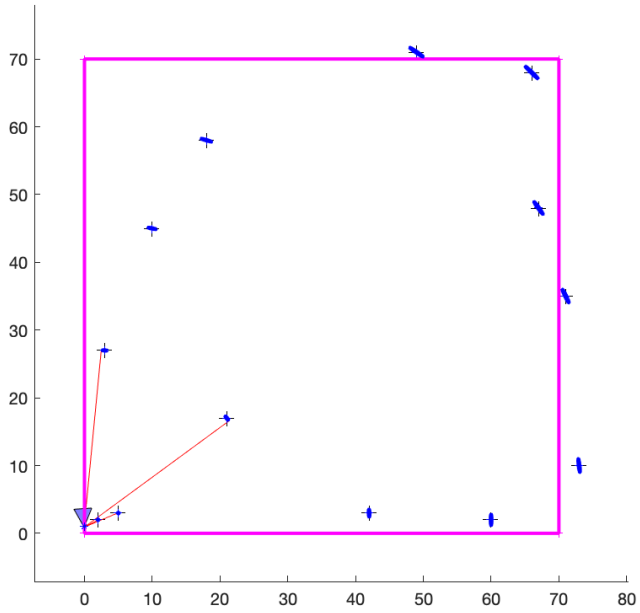

Figure 29: Case 1: Simulator Output at the end of trajectory

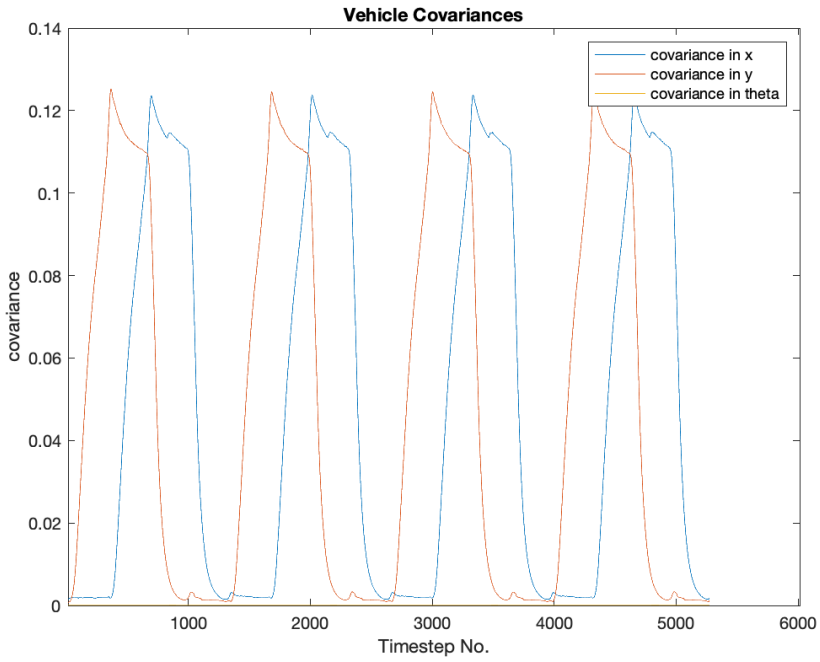

Figure 30: Case 1: Vehicle Covariances

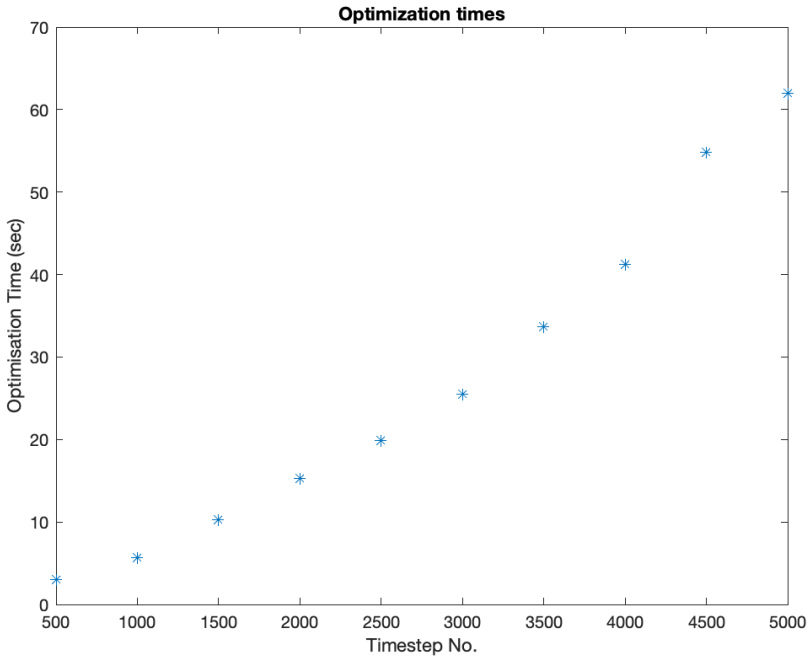

Figure 31: Case 3: Optimisation Times

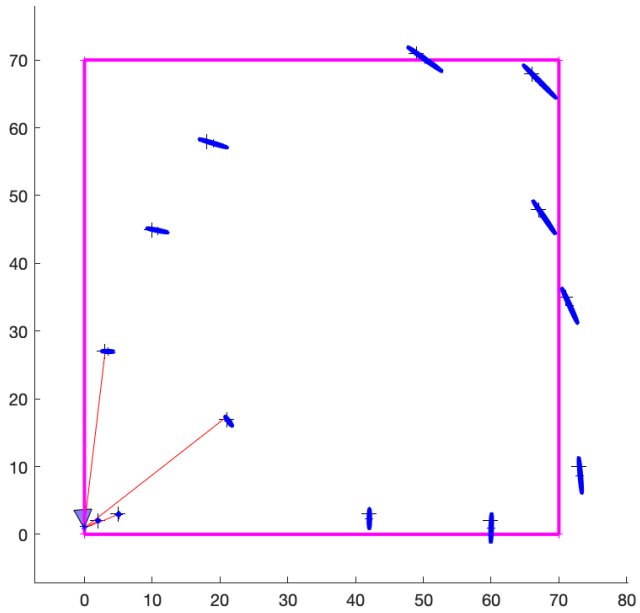

Figure 32: Case 3: Simulator Output

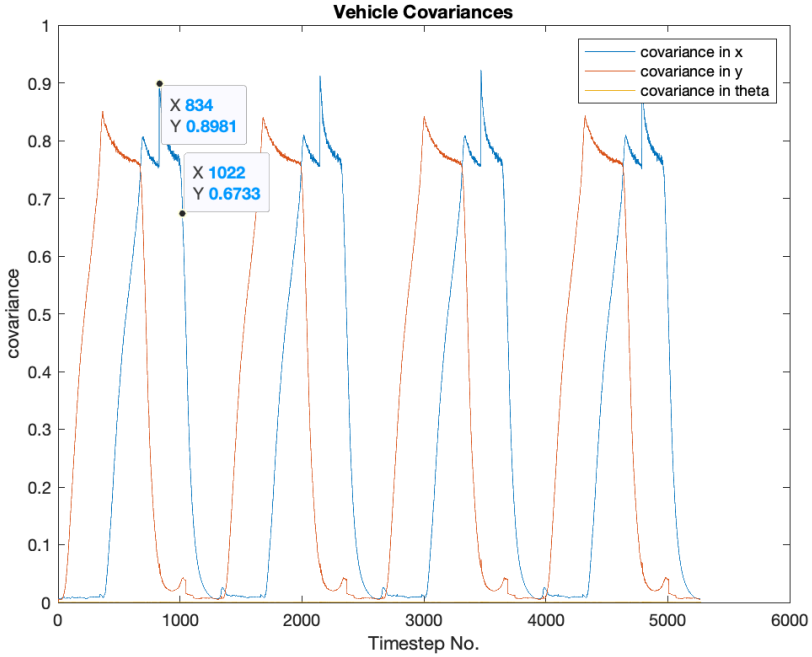

Figure 33: Case 3: Simulator Output at the end of trajectory

3b

3b - i

Graph pruning is a technique used to reduce the size of a graph in a SLAM system through the removal of edges and vertices. As the size of the graph grows linearly with respect to time [5] and has a large effect on the speed of computation and memory limits, it's important to maintain the graph at a manageable size, keeping only necessary information if memory or time constraints are present.

During the process of loop closing, the platform may observe a particular landmark at multiple time steps. The covariance associated with a landmark's position and the number of updates to this position estimate are inversely proportional to one another, therefore, more observations and updates decreases the uncertainty associated with the landmark's position estimate. However, as we consider more observations, eventually the value associated with each update decreases, and we can settle for a position estimate that has a low covariance, even if it is not the lowest we could possibly achieve. At this point, perfect accuracy from further observations is not necessary, and we can delete the edges associated with these observations, as well as the platform pose vertices.

The issue with this method comes from selecting which edges to delete. If an edge associated with important prior information, for example, is deleted, global information important to the position estimates of landmarks and the vehicle pose is lost, and can degrade the performance of the entire SLAM system. Vertex marginalisation can be used to compensate for this, although this method increases the density of graphs, making construction of and computation with them more difficult [6].

Key-frame-based SLAM similarly works to reduce the amount of data stored by limiting the size of the graph through the implementation of front-end and back-end systems. The front-end does real-time tracking of the vehicle's pose and landmark position estimates, receiving all information the vehicle obtains as it traverses the environment [5]. Only the subset of frames it deems important (where importance is determined by the nature of the task at hand) will be sent to the back-end, where a SLAM system is run, containing just platform poses and landmark position estimates. No prediction is undergone in the back-end. It uses the information it receives to build a feature map, the maximum likelihood estimation of the graph [7], which the front-end assumes is a perfect model of the environment and uses for tracking, i.e. solving the localisation problem. This method effectively reduces the sparsity of graphs by partially bounding the optimisation window, increasing the speed of computation [8], however, quantitatively determining the most important frames to be processed by

the back end may be a difficult task, and important information from frames could be accidentally missed.

3b - ii

We have decided to implement graph pruning as quantitatively determining which observations to select as key frames as well as the implementation of a front and back end system, is a much more complicated task, and a difficult one to do correctly.

3b - iii

To implement this algorithm, we used the model of the script from `3_a.m` and set the method to run in the case that we remove prediction edges, but keep the first edge. We also introduced a new attribute to the `DriveBotSLAMSystem` class which specifies whether or not graph pruning should take place, and initialised this value to true. When the system now goes to begin deleting prediction edges, it first checks if the graph needs to be pruned. If so, the `graphPruning` method of this class is called and we break out of the `deleteVehiclePredictionEdges` function (hence maintaining all edges initially).

In `graphPruning`, we loop through all edges of every vertex in the graph and a conditional statement aids in finding the number of observation edges that are present. We divide this number by the number of vertices in the graph to find a rough estimate of the average number of observation edges per vertex. We then check each vertex of the graph again and see which ones have fewer observation edges than this average number. The vertices which do have all observation edges deleted.

3b - iv

The decision process in selecting which vertex to remove observation edges from was guided by the idea that a vertex could be considered to be of low importance to the system's performance overall if few observation edges are attached to it [9]. Following on from this, vertices attached to fewer observation edges will not degrade the quality of the system significantly in such a way that errors are magnified. The goal is to prune the system in such a way that reduces computation time through the reduction in the overall density of the graph (so the optimisation step is faster), while preserving the quality of the system.

To evaluate the performance of the SLAM system after the process of graph pruning is carried out, we compare the graphs of chi2, error performance, and optimisation times for the pruned system (Case 4) and the full SLAM system with all prediction edges maintained from the previous question (Case 1).

Chi2 values for both SLAM systems were very similar, albeit very high, likely due to noisy perturbations in the process model that caused the measured values of vehicle pose and landmark position to deviate from the expected values. The chi2 value increased as the number of time steps went on as these errors accumulated over time and propagated through the system to affect the measured values at later time steps.

As was to be expected, the error performance of the system in Case 4 was slightly greater than that obtained for Case 4. We suppose that too many observation edges were removed to maintain as accurate of a vehicle pose estimate throughout the entire state trajectory in comparison to Case 1. In hindsight, perhaps a more mathematically sound method of selecting which observation edges to prune should be selected. After 1000 observations for example, over 600 edges were removed out of 3921 edges in total (20% of the total edges were removed) which caused the first spike in error seen in the graph at this time step. After the final observation, it can be seen that a further 70 out of 19,020 edges were removed (0.4% of the total edges).

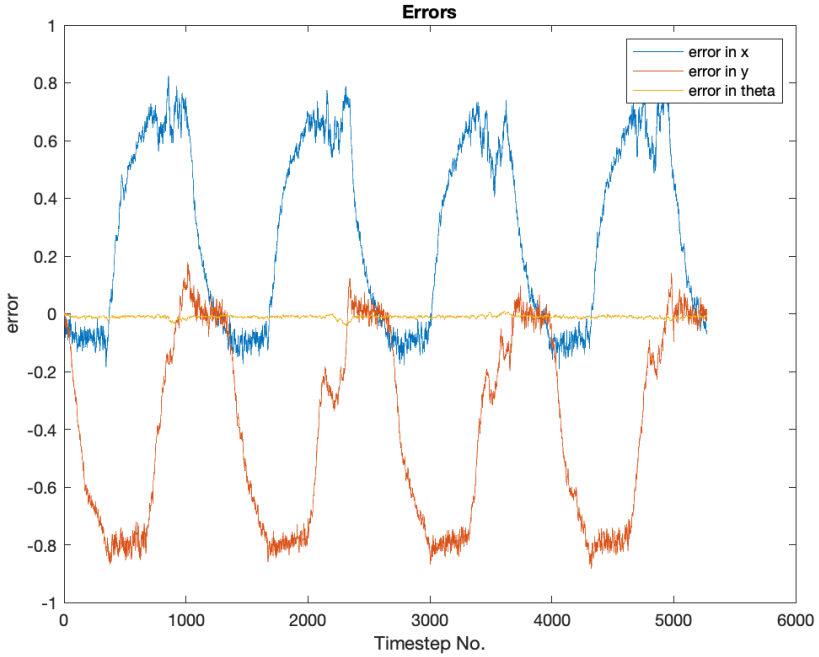

Figure 34: Case 4: Errors

The optimisation time in Case 1 is shown in Figure 24, and that of Case 4 in Figure 35. It can be seen that pruning edges has resulted in a reduction in the optimisation times. This is due to the fact that there are fewer observation edges to take into account during the optimisation process so the computational complexity of this process is lower as a result, and quicker to carry out.

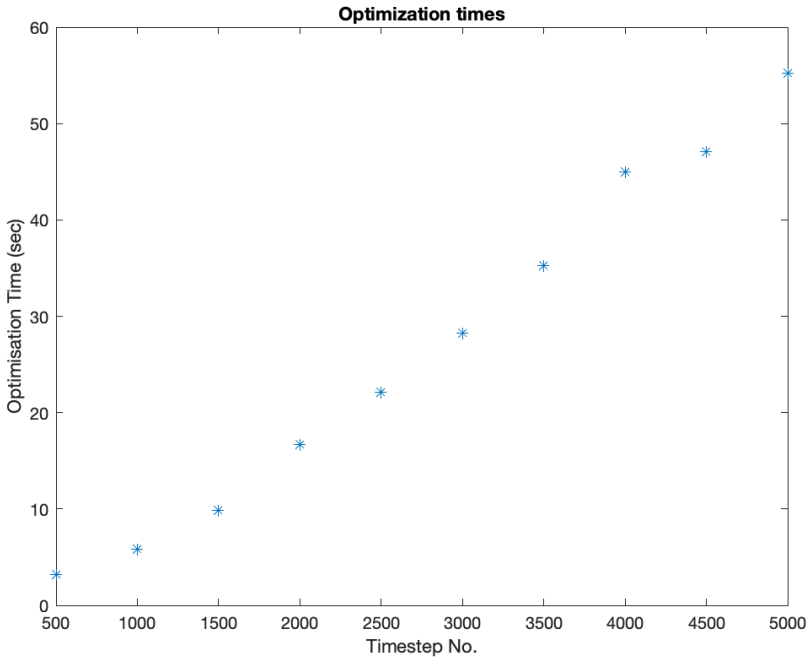

Figure 35: Case 4: Optimisation Times

We attempted to prune entire vertices and all associated edges with them (see `graphPruningOld`

function), however, due to issues with marginalisation of the vertices and difficulty with avoidance of breaking the graph due to incorrectly attached prediction edges between vehicle state vertices, this approach was abandoned for a simpler approach to graph pruning.

Ultimately, while this method was successful in decreasing the complexity of the computations by reducing the density of the graph, the 100 fold increase in error was not a desirable outcome, so a more quantitatively sound method of choosing which edges or vertices to prune should be considered to avoid potentially pruning useful observation edges.

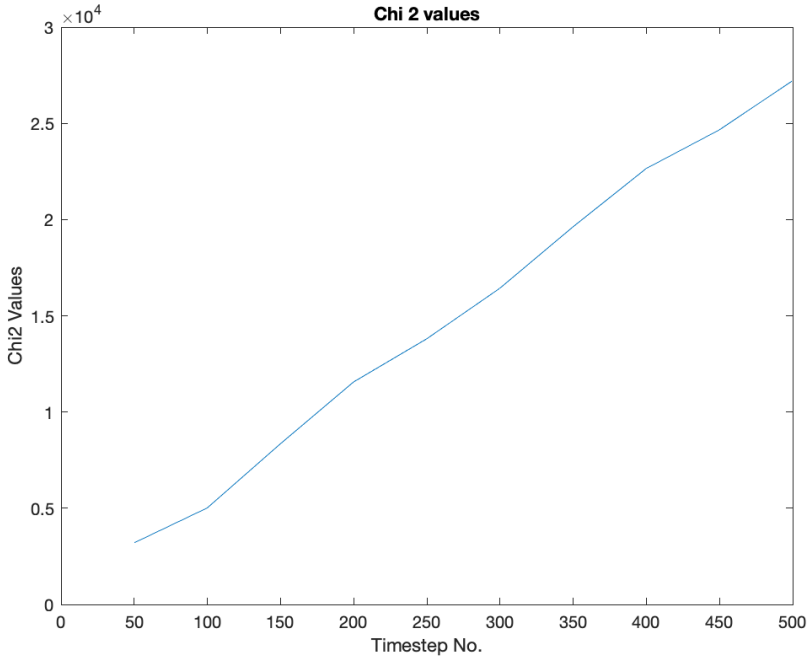

Figure 36: Case 4: Chi2 Values

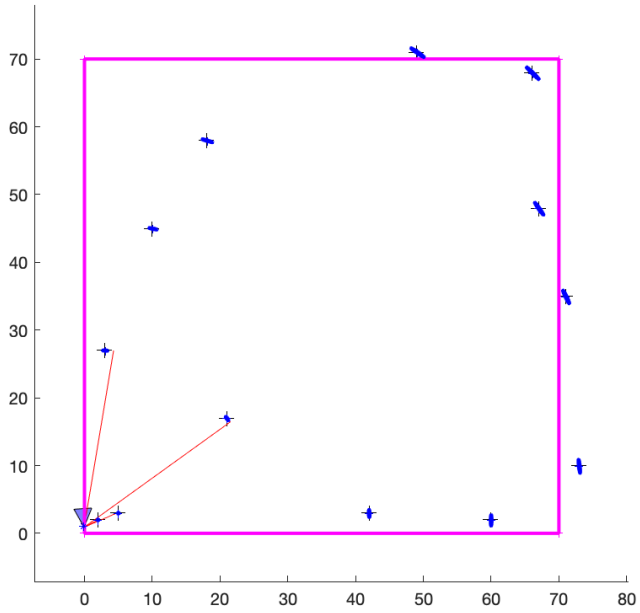

Figure 37: Case 4: Simulator Output at the end of trajectory

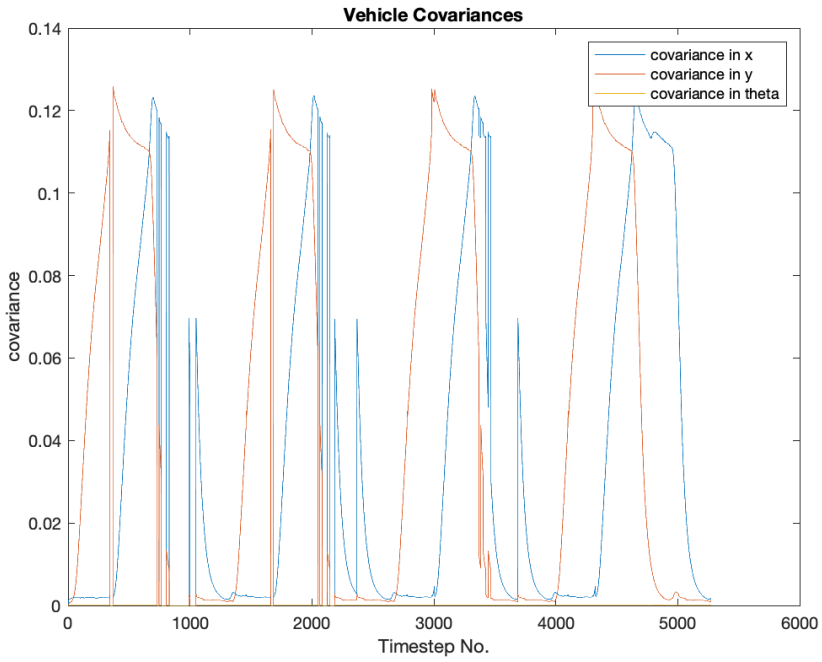

Figure 38: Case 4: Vehicle Covariances

References

[1] Wikipedia, "Factor graphs," 2022.

[2] J. Neira, "Data association in slam," 2004.

[3] G. G. et. al, "A tutorial on graph-based slam," 2010.

[4] A. Schabert, "Integrating the use of prior information into graph-slam with ndt registration for loop detection," 2017.

[5] D. S. Julier, "7b: Practical implementations with graphical models and slam," 2022.

[6] H. K. et. al, "Efficient information-theoretic graph pruning for graph-based slam with laser range finders," 2010.

[7] P. C. et. al, "Performance evaluation of graph-reduction in slam through pose rejection," 2017.

[8] S. L. et. al, "Keyframe-based visual-inertial slam using nonlinear optimization," 2013.

[9] G. K. et. al, "Geometry-based graph pruning for lifelong slam," 2021.