Department of
Computer Science

COMP0130: Robot Vision and Navigation

Coursework 2: Graph-based Optimisation and SLAM

Student Names – **Student Numbers –** **Email ID:**	**Ahmed Adamjee – 21134046 –** **Ahmed.Adamjee.21@ucl.ac.uk** **Abdulbaasit Sanusi – 21086402 –** **Abdulbaasit.Sanusi.21@ucl.ac.uk** **Ikedinaekpere Kennedy Dike – 21125081 –** **Ikedinaekpere.Dike.21@ucl.ac.uk**
Date:	**03 April 2022**

Table of Contents

List of Figures

Question 1
(a)

In the prediction and update step, there are different vertices and edges. The vertices describe the state of the event while the edges describe the conditional probabilities between events. Depending on which vertex it is connected to, the edges can be unary or binary. A binary edge connects two vertices and therefore possesses two Jacobian functions. However, a unary edge is only connected to a single vertex and as such, only has one Jacobian function.

The first edge is initialized using the initialPriorEdge class which is a unary edge and set to the vehicle initial state. A new vertex is then created using the result of *equation 1*. A binary edge of class, VehicleKinematicsEdge, is created and connects the two vertices together. The vehicleKinematicsEdge encodes the vehicle process model which is given by:

$$X_{k+1} = X_k + \Delta T_k M_k (u_k + v_k) \quad (1)$$

$where, X_k = \begin{bmatrix} x_k \\ y_k \\ \psi_k \end{bmatrix}$ is the vehicle's current state

$$M_k = \begin{bmatrix} cos\psi_k & -sin\psi_k & 0 \\ sin\psi_k & cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ΔT_k is the length of the prediction interval,

$u_k = \begin{bmatrix} s_k \\ 0 \\ \dot{\psi}_k \end{bmatrix}$, is the control input

$v_k = \begin{bmatrix} v_k \\ v_y \\ v_{\dot{\psi}} \end{bmatrix}$, is the process noise zero mean.

Furthermore, the system must also calculate the error, which is generated by the vehicle kinematics and the posterior model can be computed using the process

model. After defining the posterior model, the initial process noise can then be computed.

The error, e_k, required to compute this edge is expressed in *equation 2*

$$e_k = \frac{M_k^{-1}}{\Delta T_k} \times (X_k - X_{k-1}) - u_k \quad (2)$$

Following the calculation of the error, the Jacobians, $J_k \; and \; J_{k+1}$, which is the derivative of the error function, needed to compute this edge is computed by *equation 3*

$$J_k = \frac{\partial e_k}{\partial X_k} = \begin{bmatrix} \frac{\partial e_{k,1}}{\partial x_k} & \frac{\partial e_{k,1}}{\partial y_k} & \frac{\partial e_{k,1}}{\partial \psi_k} \\ \frac{\partial e_{k,2}}{\partial x_k} & \frac{\partial e_{k,2}}{\partial y_k} & \frac{\partial e_{k,2}}{\partial \psi_k} \\ \frac{\partial e_{k,3}}{\partial x_k} & \frac{\partial e_{k,3}}{\partial y_k} & \frac{\partial e_{k,3}}{\partial \psi_k} \end{bmatrix} \quad (3)$$

After simplification,

$$J_k = \begin{bmatrix} \frac{-cos\psi_k}{\Delta T_k} & \frac{-sin\psi_k}{\Delta T_k} & \left(-(x_k - x_{k-1}) \times \frac{sin\psi_k}{\Delta T_k}\right) + \left((y_k - y_{k-1}) \times \frac{cos\psi_k}{\Delta T_k}\right) \\ \frac{sin\psi_k}{\Delta T_k} & \frac{cos\psi_k}{\Delta T_k} & \left(-(x_k - x_{k-1}) \times \frac{cos\psi_k}{\Delta T_k}\right) - \left((y_k - y_{k-1}) \times \frac{sin\psi_k}{\Delta T_k}\right) \\ 0 & 0 & -\frac{1}{\Delta T_k} \end{bmatrix} \quad (4)$$

$$J_{k+1} = \frac{\partial e_k}{\partial X_{k+1}} = \begin{bmatrix} \frac{\partial e_{k,1}}{\partial x_{k+1}} & \frac{\partial e_{k,1}}{\partial y_{k+1}} & \frac{\partial e_{k,1}}{\partial \psi_{k+1}} \\ \frac{\partial e_{k,2}}{\partial x_{k+1}} & \frac{\partial e_{k,2}}{\partial y_{k+1}} & \frac{\partial e_{k,2}}{\partial \psi_{k+1}} \\ \frac{\partial e_{k,3}}{\partial x_{k+1}} & \frac{\partial e_{k,3}}{\partial y_{k+1}} & \frac{\partial e_{k,3}}{\partial \psi_{k+1}} \end{bmatrix}$$

After simplification,

$$J_{k+1} = \frac{M_k^{-1}}{\Delta T_k}$$

(5)

To process GPS measurements, unary edges are created where the observations represent the measurements in the GNSS system. The GNSS provides direct measurement of the robot's position, so no additional vertex is required. The edge is connected to the vehicle state vertex whenever a GPS event is called. This process repeats itself until the end of the odometry. The GPS observation model is given as

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

(6)

The covariance of w_k^G, R_k^G is diagonal and constant.

The error required to compute this edge is the difference between the x and y positions of the robot and the GPS x and y measurements.

$$e_k = \begin{bmatrix} x_k \\ y_k \end{bmatrix} - \boldsymbol{z}_k^G$$

(7)

The Jacobian required to compute this edge is given in the *equation 8* below

$$J = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

(8)

(b)

(i)

The methods that were edited to complete the prediction step functionality are:

- ***computeError*** in the ***VehicleKinematicsEdge*** class which computes the error vector of the active vector in the process model
- ***linearizeOplus*** in the ***VehicleKinematicsEdge*** class which calculates the numeric jacobians
- ***oplus*** in the ***VehcileStateVertex*** class to account for angular discontinuities
- ***computeError*** in the ***LandmarkRangeBearingEdge*** class which computes the error vector of the active vector in the measurement model
- ***linearizeOplus*** in the ***LandmarkRangeBearingEdge*** class which calculates the numeric jacobians

- ***handlePredictToTime*** of the ***DriveBotSLAMSystem*** class which is the method that actually creates the predicted vertices and updates the graph by connecting it with the prior edges with an edge

(ii)

Following the computation of the prediction steps, figure shows the error and covariance between the predicted vehicle states and the true vehicle states

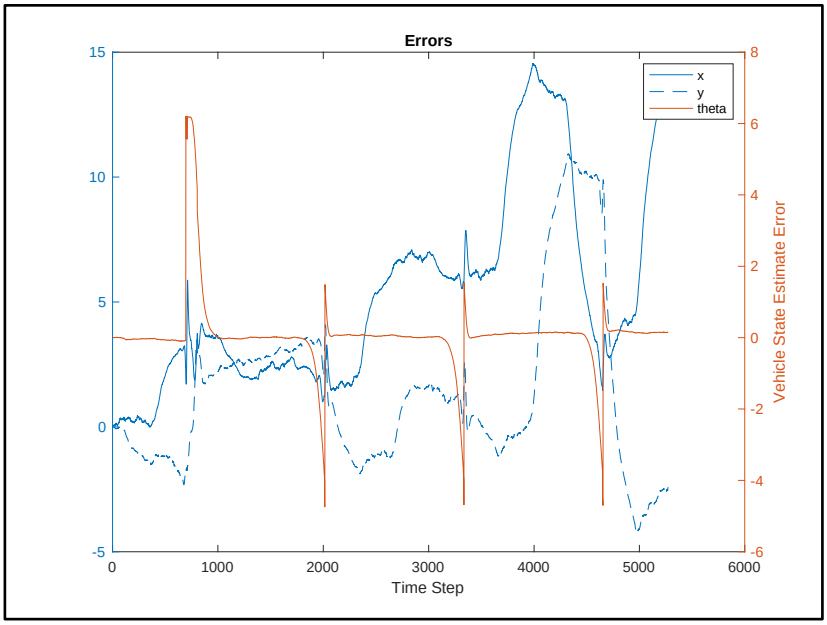

Figure 1 - showing the error plot in the prediction using only vehicle process model

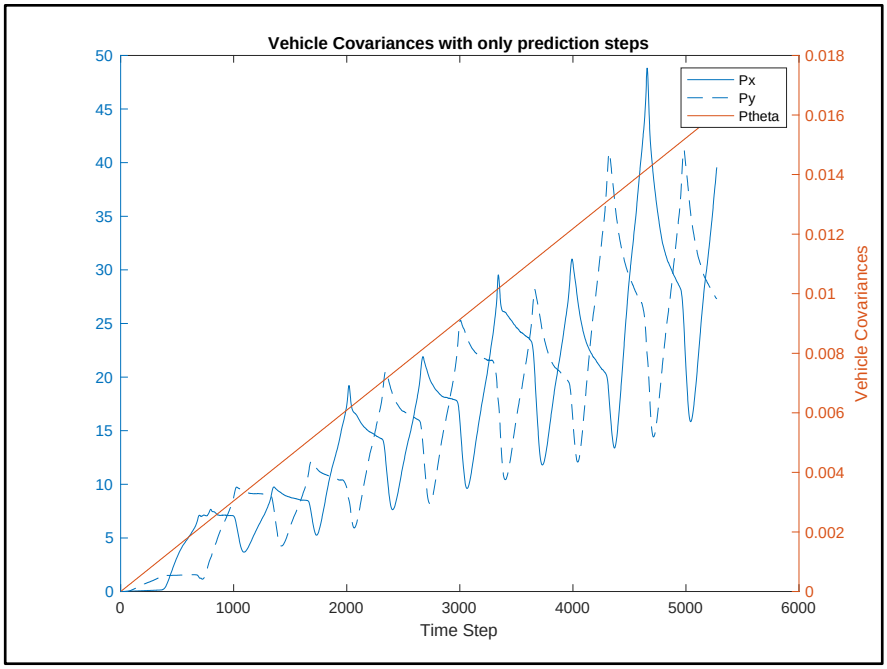

Figure 2- showing the vehicle covariance plot using only vehicle process model

As seen from *Figure 2*, as the timestep increases, the vehicle covariance, along all three state parameters, increases over time. This is because whenever the vehicle observes a landmark, the error produced by the vehicle each time step propagates over time. Hence, the error constantly increases to give the covariance plot shown in *Figure 2.* As there is no GPS update information, the vehicle does not have an estimate of the true states. This further causes the vehicle covariance to grow proportionally over time.

Similarly for the error plots in *Figure 1*, there is sort of a sinusoidal pattern present in state parameters of the error plots. This is because observations only occur at certain timesteps and as such predictions made between the previous and current observations will generate errors with the largest errors occurring at the middle as that is the midpoint. With the theta parameter, due to angular discontinuities, there are steep spikes present which can be assumed to be the point where the angle is wrapped between pi and -pi. This assumption is based on the fact that the values of the vehicle state theta parameter was approximately the value of pi which indicates that wrapping may have taken place. Also, the plot shows there is a deviation from the true vehicle state and the estimated vehicle state indicating the estimation is not entirely accurate for all timestep. This is because the process model used to predict the vehicle state is non-linear and the vehicles use just the last vehicle state to predict the current vehicle state. Also, there is process noise associated with the last vehicle state which leads to more errors being propagated as the vehicle predicts its current state.

(c)

(i)

To incorporate the GPS update step, the following methods were edited:

- ***computeError*** in the ***GPSMeasurementEdge*** class, this computes the error based on the current vertex estimate by calculating the difference between the current vertex estimate and the GPS measurement
- ***linearizeOplus*** in the ***GPSMeasurementEdge*** class, this computes the Jacobian for the most recent error calculation. It returns the numeric Jacobian computation.

- ***HandleGPSObservationEvent*** in the ***driveBotSLAMSystem*** class, this function is used to create a GPS Edge and add to the appropriate place in the graph

(ii)

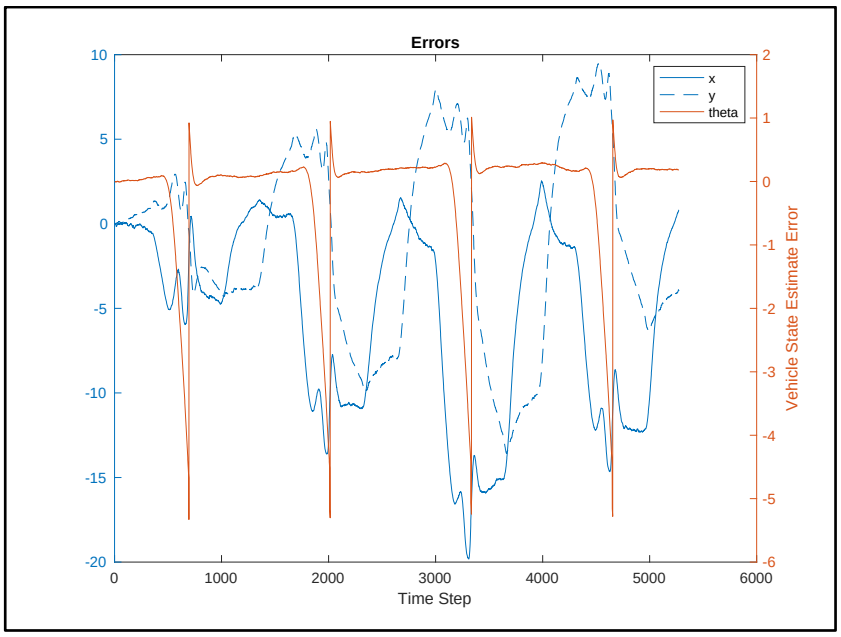

Figure 3 - showing the error plot in the prediction using vehicle process model and GPS measurement

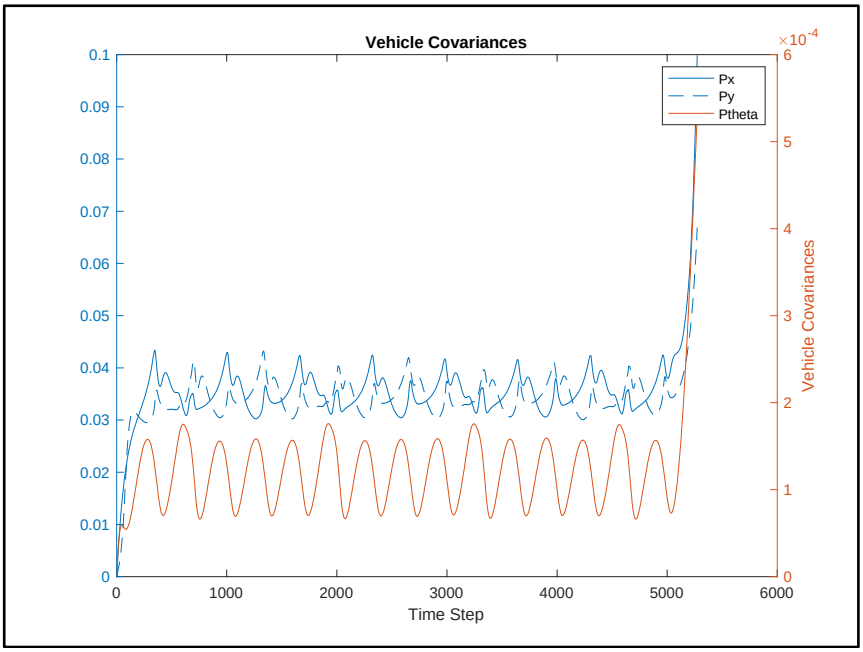

Figure 4 - showing the vehicle covariance plot using vehicle process model and GPS measurement

Similarly, as explained for *Figure 2*, as observation measurements only occur at certain timesteps, there is an oscillatory movement in the vehicle covariance plots. It can be seen that the covariance is largest when there is no GPS measurement

update indicating that there is a higher uncertainty in measurement whenever there is no GPS measurement update. Likewise, the covariance appears minimal when there is a GPS measurement update indicating there is more certainty in the estimated vehicle state. It is noteworthy that there appears to be spikes at the Start and End of the graph as there is no posterior GPS measurement update edge hence at the end of the graph the vehicle is highly uncertain about it position. Similarly, at the start, no prior update which increases uncertainty. Also, it can be noticed the magnitude of the vehicle covariance also reduces and stays within a certain range as compared to when there was GPS measurement update.

Similar to the error plot in *Figure 1*, the error plot does not significantly improve in accuracy despite the incorporation of the GPS measurement. This is because the process model used to predict the vehicle state is non-linear and the vehicles use just the last vehicle state to predict the current vehicle state. Also, there is process noise associated with the last vehicle state which leads to more errors being propagated as the vehicle predicts its current state. This shows that incorporating GPS measurement may not significantly improve the accuracy of the graph-based optimiser.

(d)

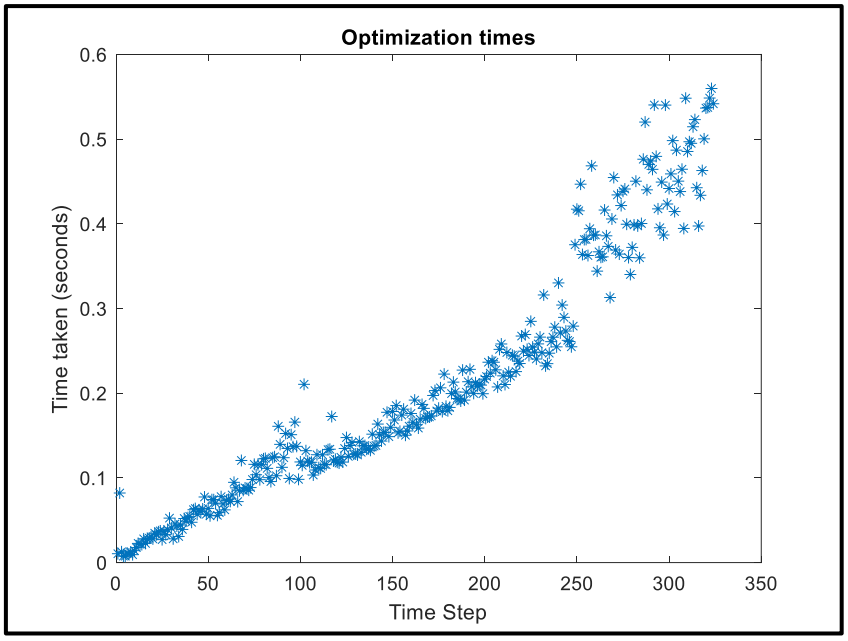

Figure 5 - Showing the time taken to optimize the graph each time step

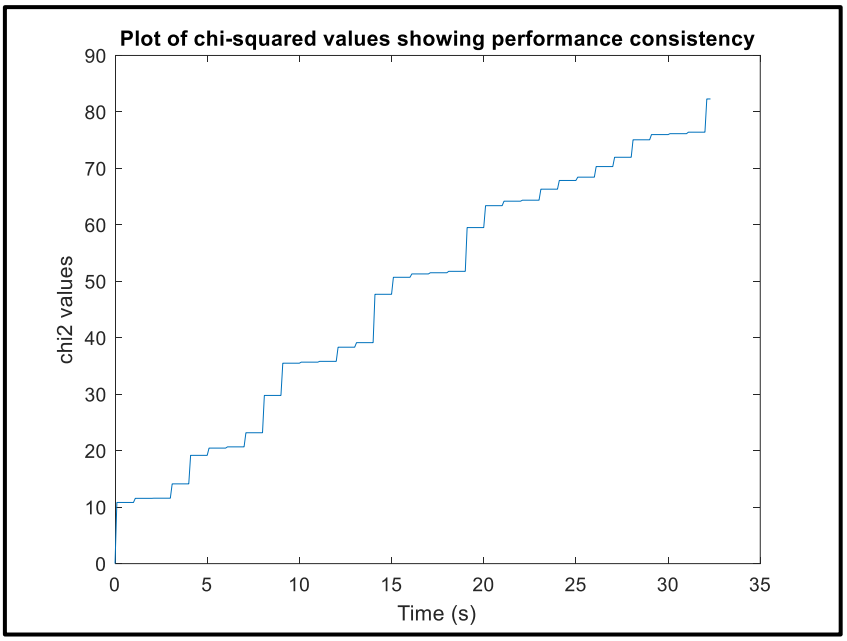

Figure 6 - showing the plot of the chi2 value over time

As we can see in the optimization graph, it shows an increase in the time taken to optimize the graph as time the timesteps increases, because as time increases the graph becomes more complex and takes a longer time to optimize, as the optimization now occurs at every timestep.

The plot of chi-squared is a measure of consistency of performance of the graph optimizer and shows how far away the vehicle is from the mean of the distribution. As the time step increases, the difference between the observed data and the expected data increases constantly, which makes sense as this shows the difference in the error terms is due to a relationship between the states as the vehicle advances. This increases as the covariance in the vehicle state estimate increases over time to account for noise and uncertainty as it has not yet seen features it has seen before as part of loop closure.

Question 2

(a)

(i)

This SLAM system is made up of different types of vertices and edges. The vertices describe the state of the event while the edges describe the conditional probabilities between events. As described earlier, edges could either be binary or unary. A binary edge connects two vertices and therefore possesses two Jacobian functions. However, a unary edge is only connected to a single vertex and as such, only has a single jacobian function.

The first edge in the graph structure is the initialPriorEdge which is a unary edge that is going to be connected to the initial vehicleStateVertex that would be initialised to be the state origin. This vehicleStateVertex describes the states of the vehicle in a particular timestep. In order to account for angular discontinuities in the vehicleStateVertex, the oplus method is implemented to handle non-linear manifolds that arises especially as one of the state parameters, x(3), is angular. This oplus method is computed by

$$x = x \boxplus \delta x \tag{3}$$

Where δx is an update from one step in the optimizer. After predictions and updates, the prior vehicle state vertex is connected to new state vertex with a vehicle kinematics edge. The vehicleKinematicsEdge encodes the vehicle process model which is given by

$$X_{k+1} = X_k + \Delta T_k M_k (u_k + v_k) \tag{4}$$

$where, X_k = \begin{bmatrix} x_k \\ y_k \\ \psi_k \end{bmatrix}$ is the vehicle's current state

$$M_k = \begin{bmatrix} cos\psi_k & -sin\psi_k & 0 \\ sin\psi_k & cos\psi_k & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ΔT_k is the length of the prediction interval,

$u_k = \begin{bmatrix} s_k \\ 0 \\ \dot{\psi}_k \end{bmatrix}$, is the control input

$v_k = \begin{bmatrix} v_k \\ v_y \\ v_{\dot{\psi}} \end{bmatrix}$, is the process noise zero mean.

If the vehicle state is described by X_k, where

$$X_k = \begin{bmatrix} x_k \\ y_k \\ \psi_k \end{bmatrix}$$

The error, e_k, required to compute this edge is expressed in *equation 7* and in the computeError method

$$e_k = \frac{M_k^{-1}}{\Delta T_k} \times (X_k - X_{k-1}) - u_k \quad (5)$$

Following the calculation of the error, the Jacobians, $J_k \; and \; J_{k+1}$, which is the derivative of the error function, needed to compute this edge is computed by *equation 8* in the linearizeOplus method

$$J_k = \frac{\partial e_k}{\partial X_k} = \begin{bmatrix} \frac{\partial e_{k,1}}{\partial x_k} & \frac{\partial e_{k,1}}{\partial y_k} & \frac{\partial e_{k,1}}{\partial \psi_k} \\ \frac{\partial e_{k,2}}{\partial x_k} & \frac{\partial e_{k,2}}{\partial y_k} & \frac{\partial e_{k,2}}{\partial \psi_k} \\ \frac{\partial e_{k,3}}{\partial x_k} & \frac{\partial e_{k,3}}{\partial y_k} & \frac{\partial e_{k,3}}{\partial \psi_k} \end{bmatrix}$$

After simplification,

$$J_k = \begin{bmatrix} \frac{-cos\psi_k}{\Delta T_k} & \frac{-sin\psi_k}{\Delta T_k} & \left(-(x_k - x_{k-1}) \times \frac{sin\psi_k}{\Delta T_k}\right) + \left((y_k - y_{k-1}) \times \frac{cos\psi_k}{\Delta T_k}\right) \\ \frac{sin\psi_k}{\Delta T_k} & \frac{cos\psi_k}{\Delta T_k} & \left(-(x_k - x_{k-1}) \times \frac{cos\psi_k}{\Delta T_k}\right) - \left((y_k - y_{k-1}) \times \frac{sin\psi_k}{\Delta T_k}\right) \\ 0 & 0 & -\frac{1}{\Delta T_k} \end{bmatrix} \quad (6)$$

$$J_{k+1} = \frac{\partial e_k}{\partial X_{k+1}} = \begin{bmatrix} \frac{\partial e_{k,1}}{\partial x_{k+1}} & \frac{\partial e_{k,1}}{\partial y_{k+1}} & \frac{\partial e_{k,1}}{\partial \psi_{k+1}} \\ \frac{\partial e_{k,2}}{\partial x_{k+1}} & \frac{\partial e_{k,2}}{\partial y_{k+1}} & \frac{\partial e_{k,2}}{\partial \psi_{k+1}} \\ \frac{\partial e_{k,3}}{\partial x_{k+1}} & \frac{\partial e_{k,3}}{\partial y_{k+1}} & \frac{\partial e_{k,3}}{\partial \psi_{k+1}} \end{bmatrix}$$

After simplification,

$$and\ J_{k+1} = \frac{M_k^{-1}}{\Delta T_k}$$

(7)

The range bearing measurement of the landmarks are stored in the ***LandmarkRangeBearingEdge***. This edge connects the vehicle state vertices is connected to another vertex which is the landmark state vertex. The Landmark state vertex describes the states of the landmark at a particular timestep. The landmark observation model, which measures the range, azimuth, and elevation of a landmark relative to the platform frame is given as

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

(8)

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

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

$x^i and\ y^i$ indicate the states of the landmarks

The error required to compute this edge is expressed in *equation 11* and in the computeError method is given by

$$e_k = \begin{bmatrix} r_k^i \\ \beta_k^i \end{bmatrix} - z_k^L$$

(9)

The Jacobian required to compute this edge is given in equa*tions 12-13* and is in the linearizeOplus method.

$$J^i = \begin{bmatrix} -\frac{x^i - x_k}{r_k^i} & -\frac{y^i - y_k}{r_k^i} & 0 \\ \frac{y^i - y_k}{{r_k^i}^2} & -\frac{x^i - x_k}{{r_k^i}^2} & -1 \end{bmatrix}$$

(10)

$$J^{i+1} = \begin{bmatrix} \frac{x^i - x_k}{r_k^i} & \frac{y^i - y_k}{r_k^i} & 0 \\ -\frac{y^i - y_k}{{r_k^i}^2} & \frac{x^i - x_k}{{r_k^i}^2} & 1 \end{bmatrix}$$

(11)

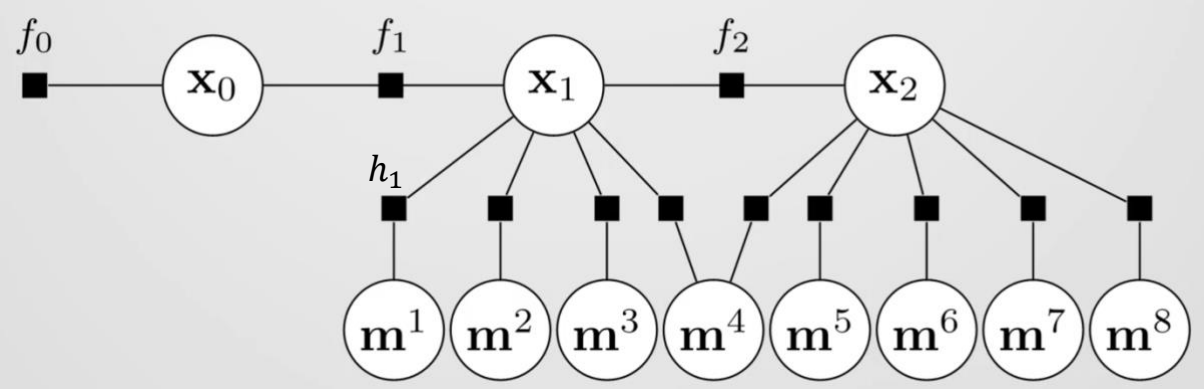

Figure 5 - A pictorial representation of the SLAM system after two timesteps. f_k represents the vehicle kinematics edges with f_0 representing the initial prior edge, x_k represents the vehicle state vertices. m^k represents the landmarks state vertices with h_k represents the landmark range bearing edges [3].

(b)

(i)

The only method edited to complete the functionality of this task is ***handleLandmarkObservationEvent***. This method iterates over all the measurements and if a new measurement is found, creates a new landmark vertex, and attach it to the SLAM graph structure.

(ii)

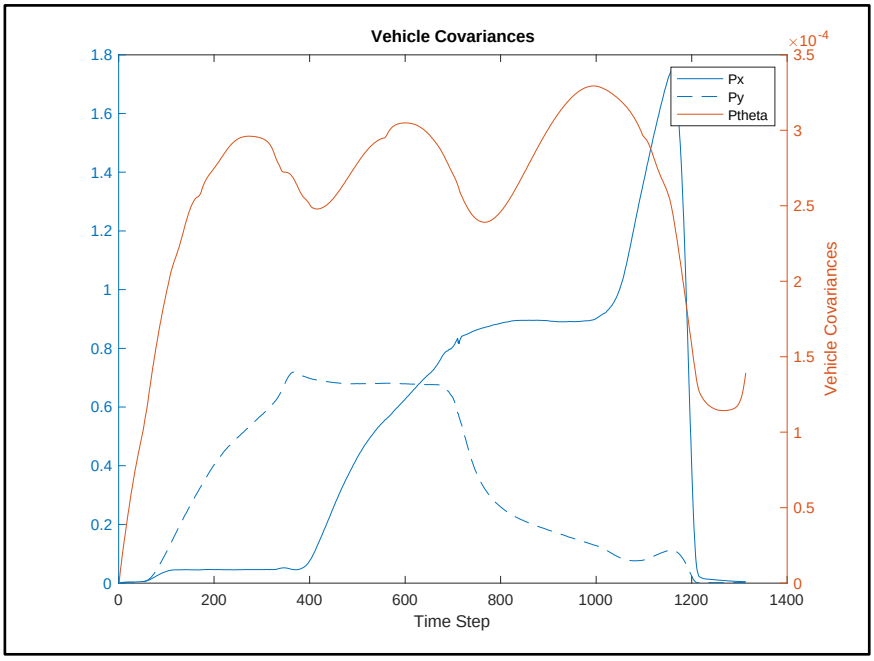

Figure 6 - showing the vehicle covariance plot of the graph when SLAM is implemented

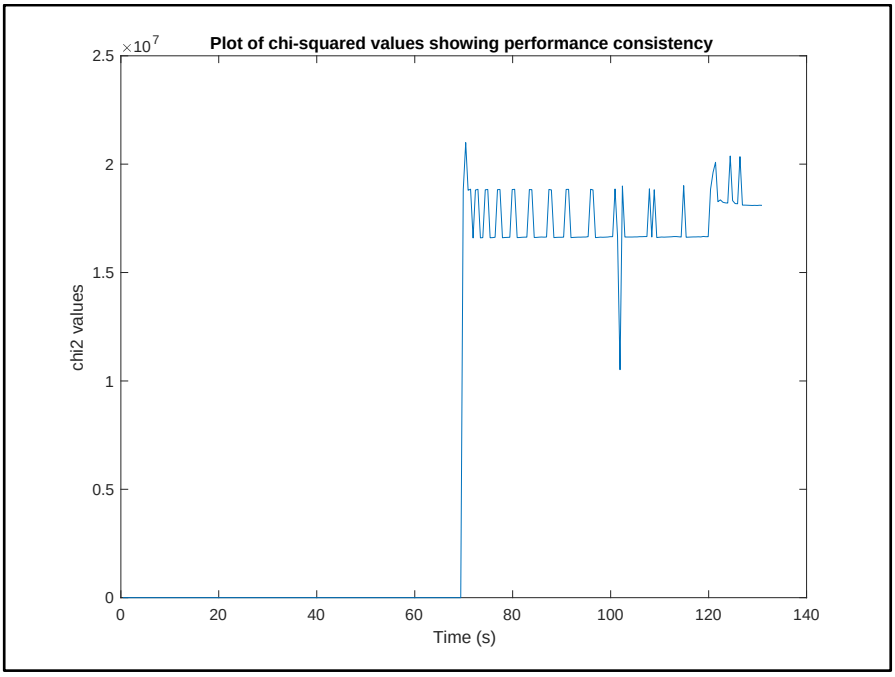

Figure 7 - showing the plot of the chi2 value over time when SLAM is implemented

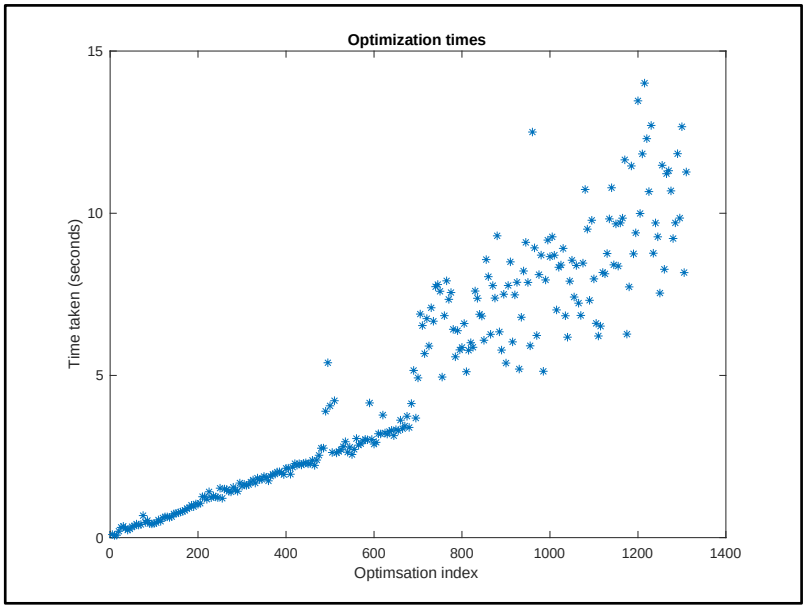

Figure 8 - Showing the time taken to optimize the graph each time step when SLAM is implemented

Again, the trend on the optimization graph shows an increase in the time taken to optimize the graph because as time increases the graph becomes more complex and takes a longer time to optimize. The magnitude of the optimisation time increases significantly when SLAM is implemented as it the complexity of the graph is higher. As can be observed from the graph, the time taken to optimize gradually increases in a linear and tightly packed fashion however as the time approaches the 800^{th} timestep, the trend after this timestep shows the optimization graph being sparsely packed and non-linear increase in the time taken to optimize the graph. This indicates that the graph is getting more complex with more landmarks. At timestep 1250, the optimization time is highest, and curve grows in a straight-line fashion because loop closure occurs.

As for the chi2 curve, this tells us that the converged value of chi2 is much bigger or much smaller than the sum of the error vector dimensions in the graph, and it indicates that the covariance is not a good match for the residuals involved in the optimization process.
It appears correlated with the optimization time curve, and it is 0 when the optimization curve is linear and spikes up when the time taken to optimize the graph increases indicating that the time graph is getting more complex.

As for the vehicle covariance plot, the first time the vehicle detects the landmark, it will produce large covariance moreover when the vehicle observes the registered landmarks again, close loop system will reduce the landmark platform uncertainty and largely decrease the covariance of the vehicle state, and this can be observed from the vehicle covariance plot above. Loop closure occurs at timestep 1200, and it can be seen that that the vehicle covariance reduces to almost 0 at this time step indicating that the vehicle has seen the landmark before and is more confident about its true position now.

(c)

(i)

To investigate the properties of the structure of graph, the following information were obtained by querying the constructed graph,

- The Number of vehicle Poses Stored
- The Number of Landmark Initialized
- The Average Number of observations made by a robot each timestep
- The Average Number of observations received by each Landmark

In the script, to calculate the Number of vehicle poses stored, a variable num_vehicle_poses was initialized to keep track of the vehicle state vertex. A for loop is then initial to go through all vertices in the graph, within this loop, a conditional statement is written to increment the count of the num_vehicle_poses if the current vertex in the loop belongs to the class vehicleStateVertex. MATLAB *isa* function was used to do this. At the end of the loop, the num_vehicle_poses gives the total number of vehicle poses stored in the constructed graph.

To calculate the Number of Landmark initialised, similar to the process of calculating number of vehicles poses, a variable num_landmarks was initialized to keep track of landmark state vertex. Using the same for loop as above to go through all vertices in the graph, a conditional statement was also written using the MATLAB isa function. The condition was to increment the num_landmark variable anytime the current vertex belonged to the LandmarkStatevertex class. At the end of the loop, the variable num_landmark gives the total number of Landmark initialized in the constructed graph.

To calculate the average number of observations made by a robot each timestep, the total number of observations was computed first by initializing a variable num_observation to keep track of all observations made and then using a for loop to go through all edges in the graph. A conditional statement is then used to increment the num_observation variable whenever the current edge belongs to the class LandmarkRangeBearingEdge using the MATLAB isa function. The total number of observations is then divided by the total time step to get the average number of observations made by a robot each timestep.

Lastly, to calculate the average number of observations received by each Landmark, we divide the total number of observations obtained above by the number of landmarks initialized. This gives the average number of landmarks received by each Landmark.

(ii)

The table below shows the values associated with the information obtained by querying the constructed graphs

Properties of constructed Graph	Value
Number of vehicle Poses Stored	5273
Number of Landmark Initialised	7
Average Number of observations made by a robot each timestep	0.643
Average Number of observations received by each Landmark	484.143

The table above shows that although there were 5280 vertices in the graph constructed, only 7 were Landmark vertices while the remaining were just vehicle poses. The average number of observations made each timestep is 0.643 and the average number of observations received by each Landmark is about 484.

(d)

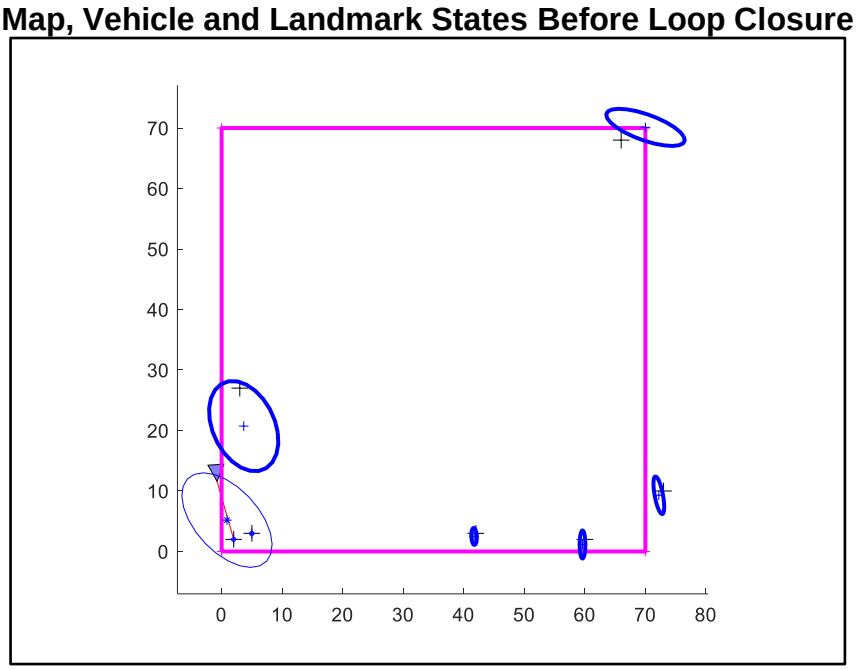

Figure 9 - showing simulator output with map, vehicle, and landmark state before loop closure

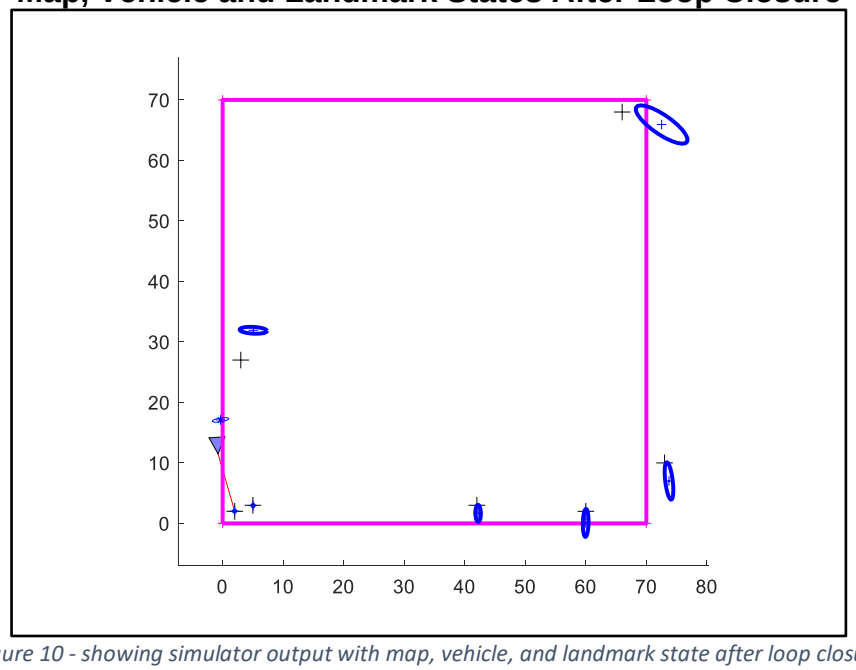

Figure 10 - showing simulator output with map, vehicle, and landmark state after loop closure

Loop closure happens between timesteps 1206 and 1207.

Loop closure happens when the robot has seen a feature it has seen before, this causes a change in the covariance matrices seen on the landmark and vehicle estimates after the next update. The robot knows the pose of the landmark it has seen before with a very small uncertainty, therefore observing this landmark again

helps reduce the covariance of the vehicle state estimates. This information is also used to consider its previous belief and reduces the vehicle covariances for the previous estimates as well, which causes the uncertainty of each landmark observed previously to decrease as well as they are connected on the graph.

This can be showcased in the calculation below signifying the amount by which the uncertainties in the vehicle state estimates have changed:

Determinant Of the Covariance of Vehicle State Estimate Before Loop Closure:
1.0229×10^{-6}
Determinant Of the Covariance of Vehicle State Estimate After Loop Closure:
9.8228×10^{-7}

Amount By Which the Uncertainty Has Changed for the Covariance of Vehicle State Estimate:
4.0665×10^{-8}

Question 3

(a)

(i)

The size of graphs grows over time as it encounters new features and puts them in the graph which makes it harder to solve. These prediction Edges keep getting added every single timestep which makes the number of edges grow linearly with time. So eventually this may lead to large graph that may be impossible or computational expensive to compute. Hence Removing vehicle Prediction Edges is a way of controlling the size of the graph.

This strategy is likely to be successful when it is done carefully such that the edge that is deleted is not carrying valuable information about the map that can degrade performance.

This strategy is likely to be unsuccessful when an edge that carries valuable prior information is deleted. E.g., if the first vehicle prediction edge is deleted.

(ii)

The only method edited to delete all vehicle prediction edges is:

- ***deleteVehiclePredictionEdges-*** This goes through all the edges in the graph and checks if the Edge is a vehicle prediction edge. This edge is then removed from the graph as long as it is not the first edge in the graph.

(iii)

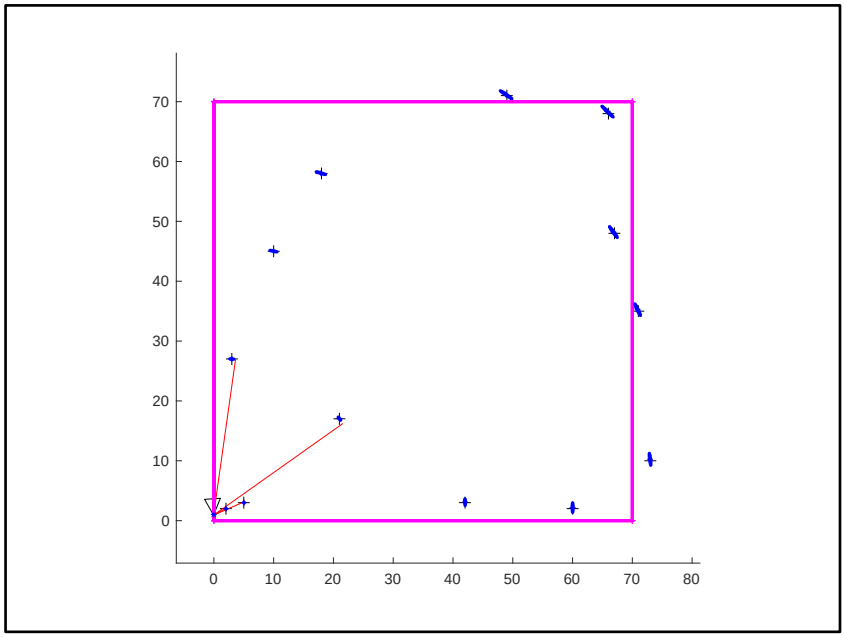

Figure 11 - Showing simulator output before deleting the vehicle prediction edges

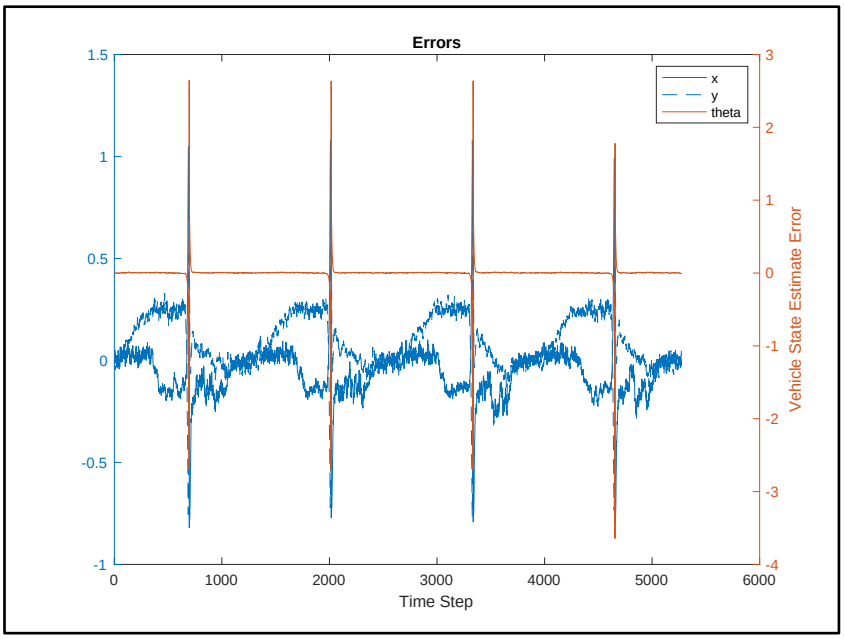

Figure 12 - showing Error Plots before deleting vehicle prediction edges

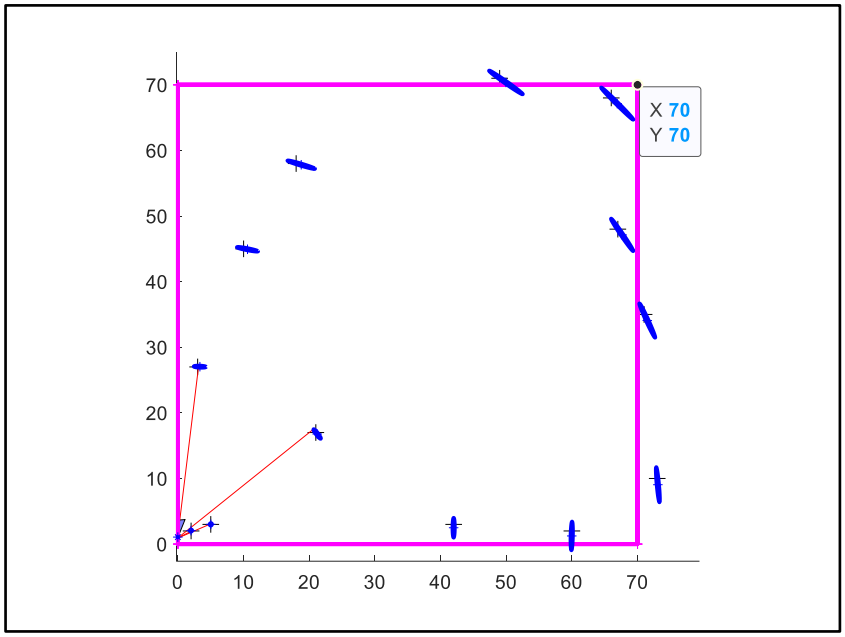

Figure 13 - Showing simulator output after deleting all but the first vehicle prediction edges

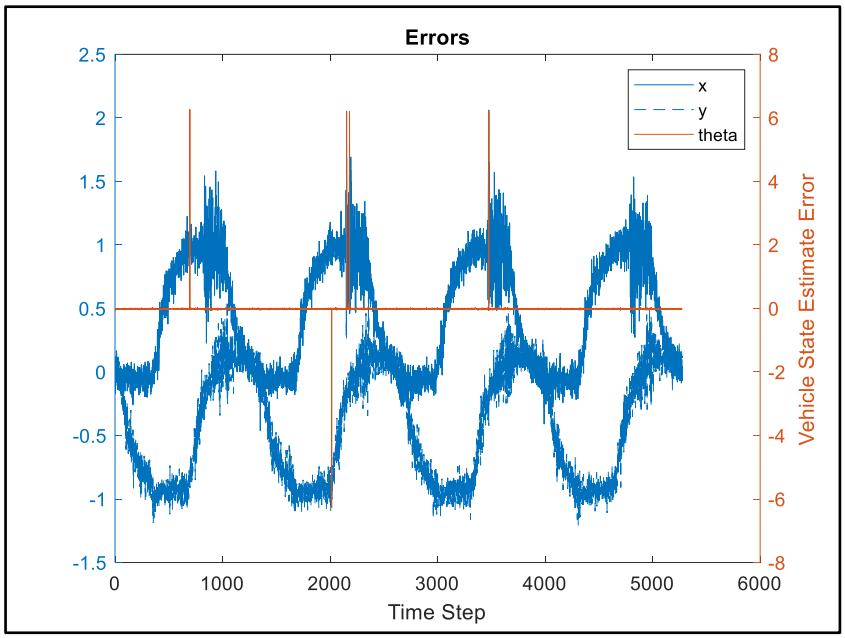

Figure 14 - showing Error Plots after deleting all but the first vehicle prediction edges

Looking at the plots, it can be seen from the two plots that the covariance of the landmarks is slightly smaller when all the prediction edges are removed. This is because some parts of the graphs are removed and therefore, it is prone to more error. However, although the covariances were slightly higher when all but the first prediction edges were removed, it can be noted that it was also quite low. Its low covariance could be attributed to the fact that some prediction edges do not carry

enough important information or are redundant. So, removing these edges would not significantly affect performance.
Similarly, the magnitude of the error increases slightly when the prediction edges are removed for the same reason. Also, more noise can be seen when those edges were removed.
It should be noted that the behaviour of the system cannot be analysed when all the prediction edges were deleted as the system fails to run. This failure is because the first edge, which was also deleted, contains some important prior information. Without this, the graph structure of the system cannot be created.

(iv)

Looking at the covariance plots, it can be seen the patterns are very similar to that of the full SLAM system. However, the magnitude of the covariance of the SLAM system with all but the first prediction edges removed is higher when compared to when to the full SLAM system.
As always, the optimisation times increases every time step because as times goes on, the graph structure gets more complex. Furthermore, when compared to the full SLAM system, the optimisation times are lower due to lower complexity as some edges were removed. And optimization occurs only at specfie
The magnitude of the chi2 values, when all but the first prediction edges were removed, is lower when compared to the full SLAM system. Also, the chi2 plot can be described as a linear relationship between the timestep and the chi2values. Whereas in the full SLAM system, after the 100^{th} time step, a non linear pattern occurs.
In general, this strategy can be considered successful as even though the errors and covariances increased, the magnitude of the increase is not significant enough to seriously hamper the performance while also reducing optimisation time.

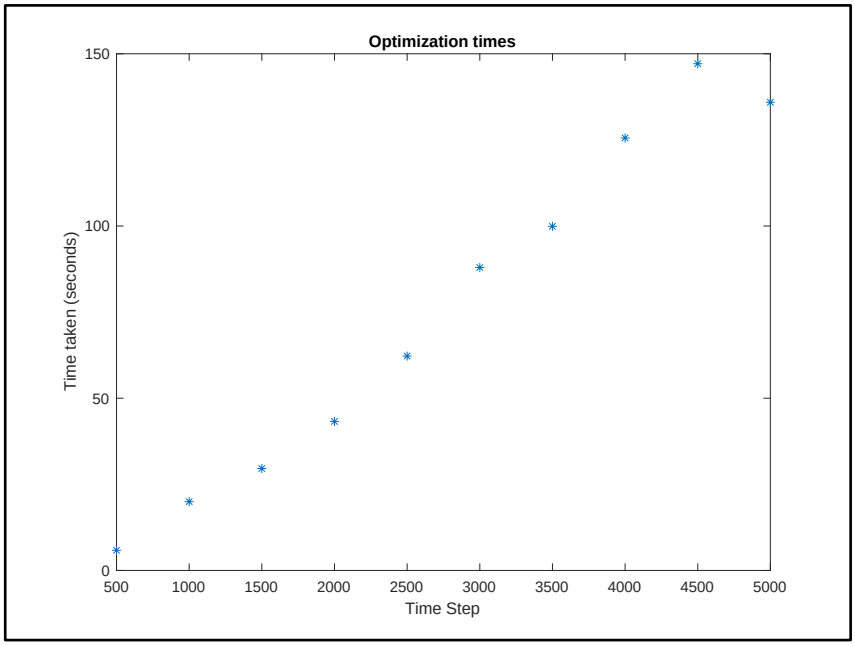

Figure 15 - Showing the time taken to optimize the graph each time step when all the prediction edges are maintained

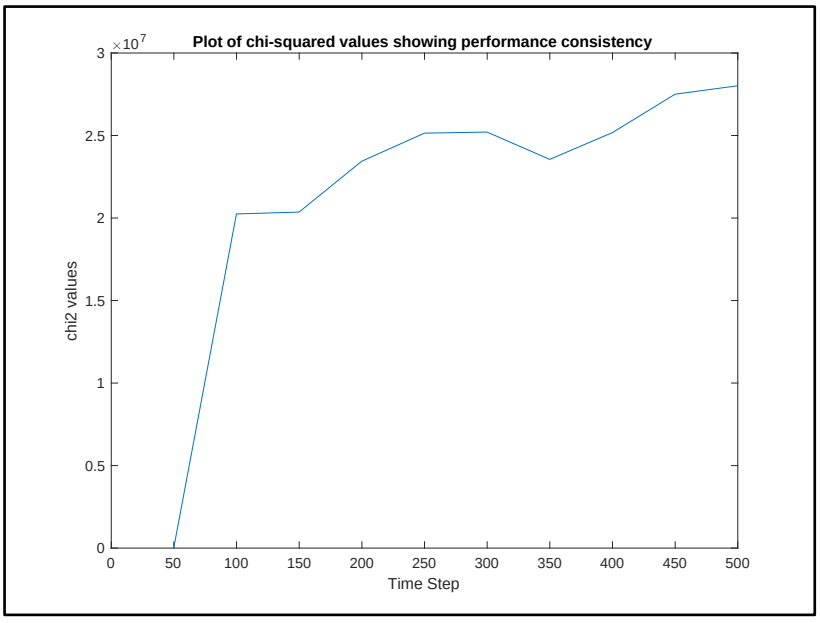

Figure 16- showing the plot of the chi2 value over time when all the prediction edges are maintained

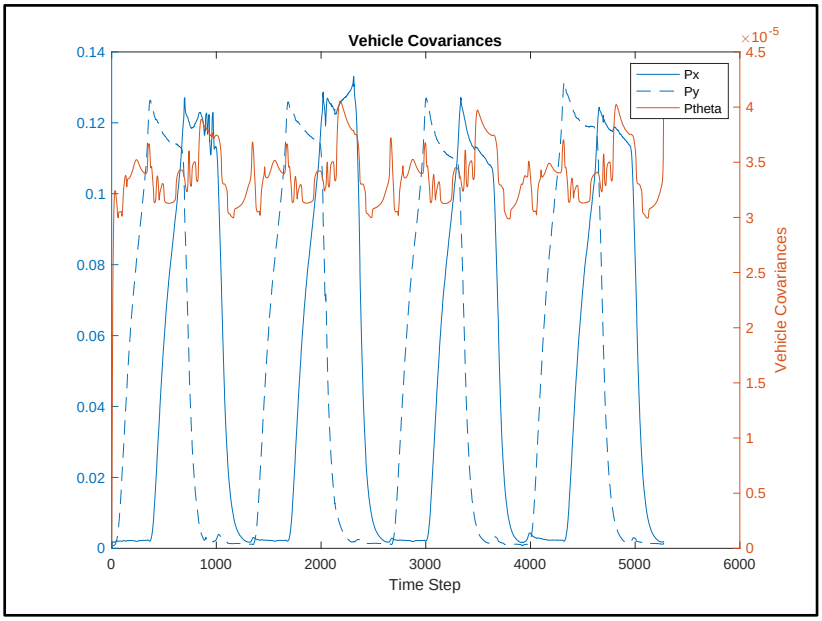

Figure 17- showing the vehicle covariance plot of the graph when all the prediction edges are maintained

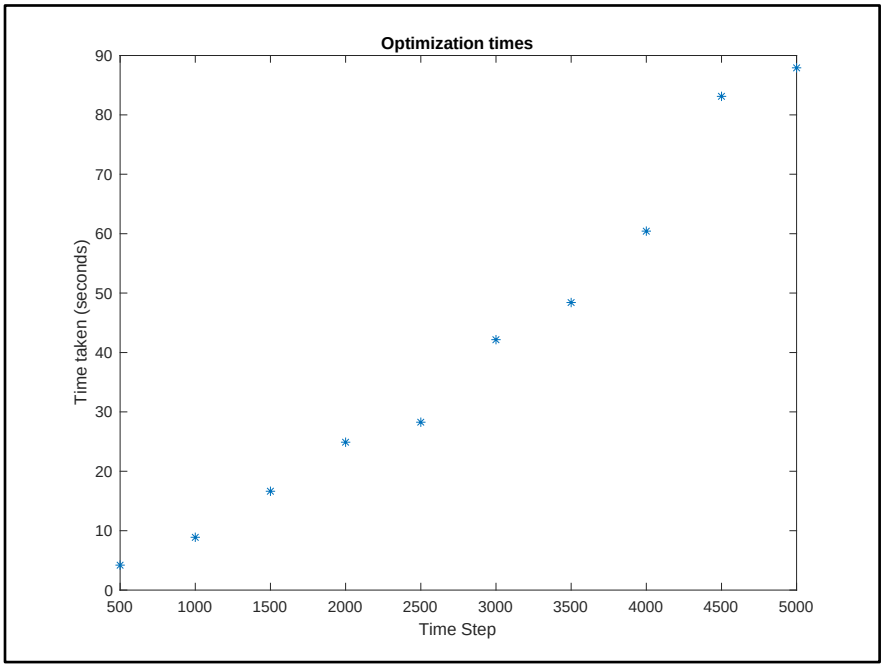

Figure 18 - Showing the time taken to optimize the graph each time step after deleting all but first vehicle prediction edges

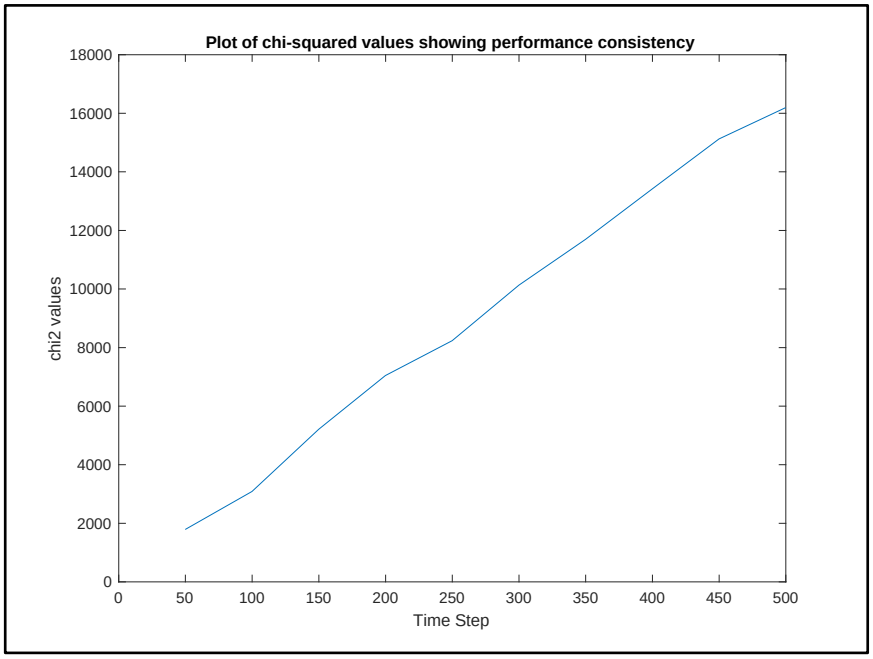

Figure 19 - showing the plot of the chi2 value over time after deleting all but first vehicle prediction edge

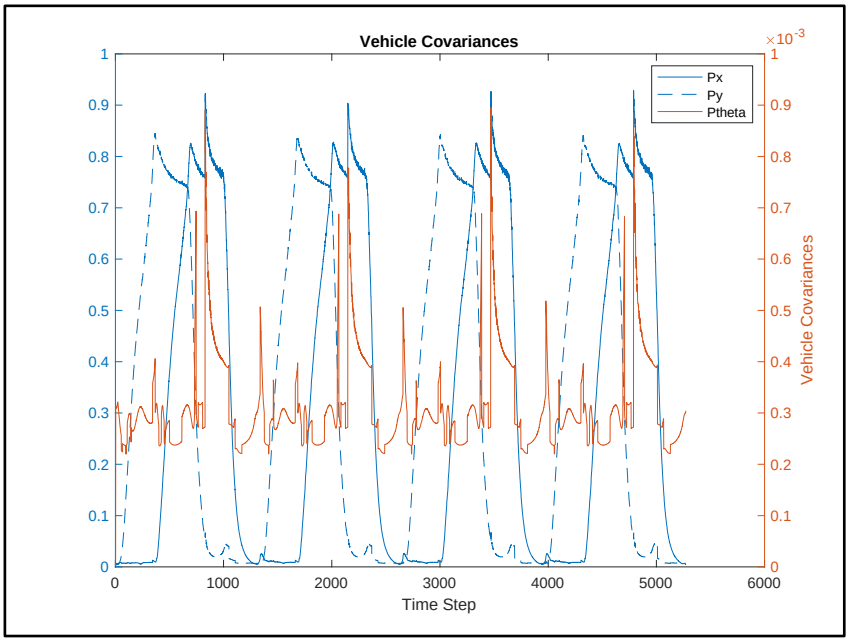

Figure 20 - showing the vehicle covariance plot of the graph after deleting all but the first vehicle prediction edge

(b)

(i)

As observed in the graphical model without implementing ways to improve the performance of the SLAM system, the size of the graph grew at a constant rate over time, which leads to an extremely slow performance and requires high computational power, and in the long term the robot can run out of resources to build the map.

Approaches for simplifying graph such as Sparse Keyframe based SLAM and Graph pruning methods are different strategies used to control the size of the graph and reduces the storage required. However, these methods do not reduce the computation power required to optimize the graph.

Sparse keyframe based SLAM methods is an approach which restricts what data is put into the map in the first place [4]. They use a frontend and a back end, where the front end performs real-time tracking and data association, whereas the backend has consisted of the platform pose and map points and runs the SLAM system, using structure from motion algorithms. The frontend sends a set of valuable information, using global optimizers, to the backend in frames regularly. The backend optimizes the graph structure provided by the front-end. The ideology behind this is if the vehicle places landmark estimates in the environment which have several observations and are fairly accurate, the measurements for the same landmarks do not need to be put in the map again if the landmark is observed again after approaching it again. A strength of this approach is that it can handle events when data is missing or is interrupted as these would not be added to the graph by performing optimisation over multiple keyframes [5]. Additionally, the computational power required reduces as the optimization of the estimates do not have to be performed over all points in the graph [5]. However, a complication with this approach is that a suitable algorithm to select the information, which requires minimal computational power needs to be designed.

Graph pruning methods work by removing vertices and edges from a map after they have been added [2]. The principle behind this approach is that as more landmarks are detected, each landmark detected caries increasingly lesser value as not all landmark observations are needed to efficiently localize the vehicle state estimates. Graph pruning methods exploit this idea by deleting vertices and edges from the graph when information observed while building the map may not be extremely valuable. This is generally found by setting a threshold for the information gain expected from each scan as the world state estimate is given by the posterior probability distribution of the map, and scans with minimal information gain may not influence the posterior probability distribution of the map greatly [1]. However, the limitation with this approach is that the information regarding the vertex and

landmark observations is removed entirely and can affect the performance of the system. To minimize this loss of information, many approaches marginalize out the vertices instead, which summarises the deleted information by incorporating them into factor nodes. Doing an exact marginalization, however, also creates additional issues in the graph by affecting its sparsity structure as it introduces new edges between vertices which were previously connected to the deleted vertices. This is not sought after as it creates a dense covariance matrix which requires more memory resources and furthermore also increases the computation costs for optimization of vehicle pose estimate. There are a few approaches, namely the Chow-Liu tree approximation, which treats some pairs of vertices as conditionally independent [1].

(ii)

Graph pruning method is chosen; as to implement a Sparse keyframe method, a front-end and backend system must be designed, along with a selection process regarding which data needs to be sent to the backend. The algorithm to select which data should be added to the graph can also be computationally expensive.
Therefore, the graph pruning method is a simplistic and efficient algorithm to implement and its complexities relating to the sparsity structure are not significant in this application as each vehicle state vertex only has 2 prediction edges associated with them.

(iii)

To implement the graph pruning functionality, a script named ‘q3_b.m’ was created, which enables graph pruning while executing the SLAM system.
During execution it inspects all the edges attached to each vertex and checks weather it is a landmark observation edge, which increments a counter on the total number of landmarks observed by the vehicle. If the number of landmarks observed by a vehicle are very small (with the implemented set threshold as 3), it is decided that this vertex does not provide valuable information on the execution of the SLAM algorithm, and all the landmark edges associated with this vertex are removed.

Algorithm 3.1 – *Algorithm to implement a Graph pruning method.*	
1	**FOR** VERTEX_ID **in** ALL_VERTEX_IDS
2	**FOR** EDGE_ID **in** ALL_EDGES_ON_VERTEX
3	**IF** EDGE **isa** “LandmarkRangeBearingEdge”
4	totalLandmarkEdgesOnVertex += 1
5	**IF** ((totalLandmarkEdgesOnVertex < 3) && (VERTEX_ID > 1))
6	**FOR** EDGE_ID **in** ALL_EDGES_ON_VERTEX
7	**IF** EDGE **isa** “LandmarkRangeBearingEdge”
8	this.graph.removeEdge(vertexEdges{ EDGE_ID });

The main drawback with this approach is that the vertices are still retained as deleting them may create complications associated with creating new edges between the neighbouring vertices, and they must be marginalized appropriately to set the correct measurement and information associated for the new edges being created and is complicated as integration over several vertices needs to be performed. Whilst the vertices were attempted to be deleted, there were still some vertices left unconnected in the graph, which responded with errors while executing the SLAM system.

(iv)

Key characteristics:

Total number of vertices in the graph:

- 5286

Total number of edges in the graph:

- 19449

Total number of edges removed from the graph:

- 2430

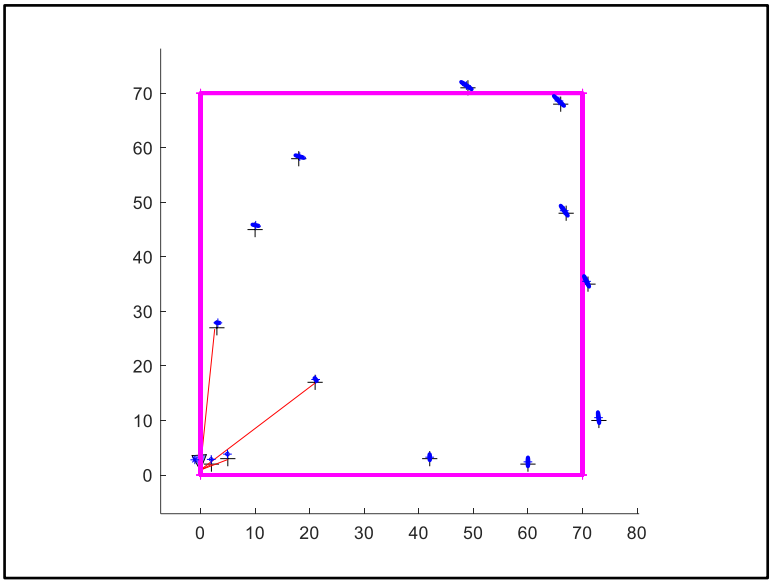

Figure 21 - Showing simulator output after implementing the graph pruning algorithm

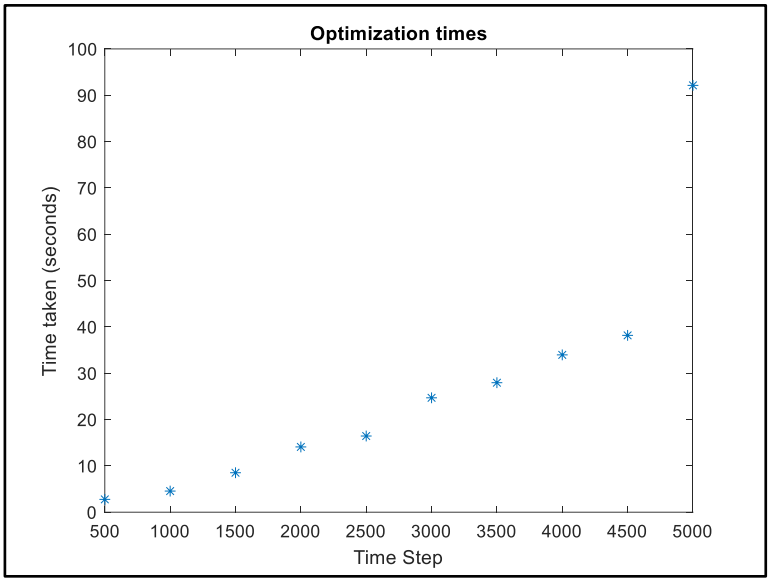

Figure 22 - Showing the time taken to optimize the graph each time step after implementing the graph pruning algorithm

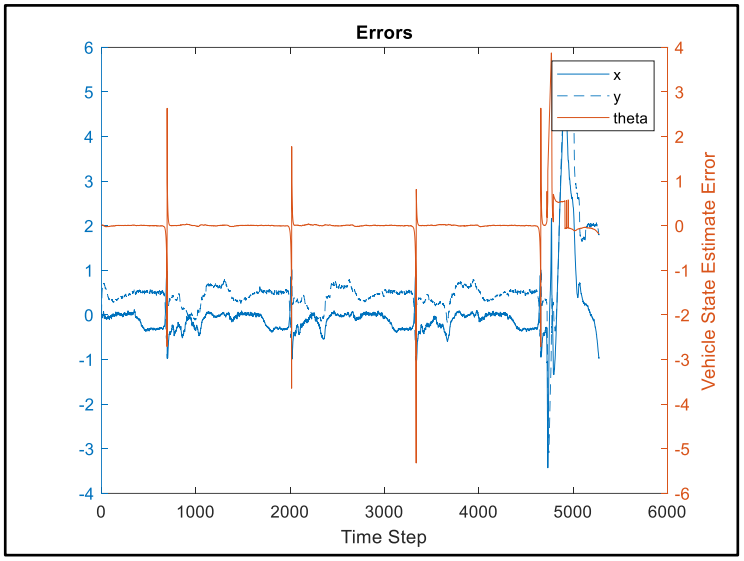

Figure 23 - showing Error between true and predicted state after implementing graph pruning algorithm

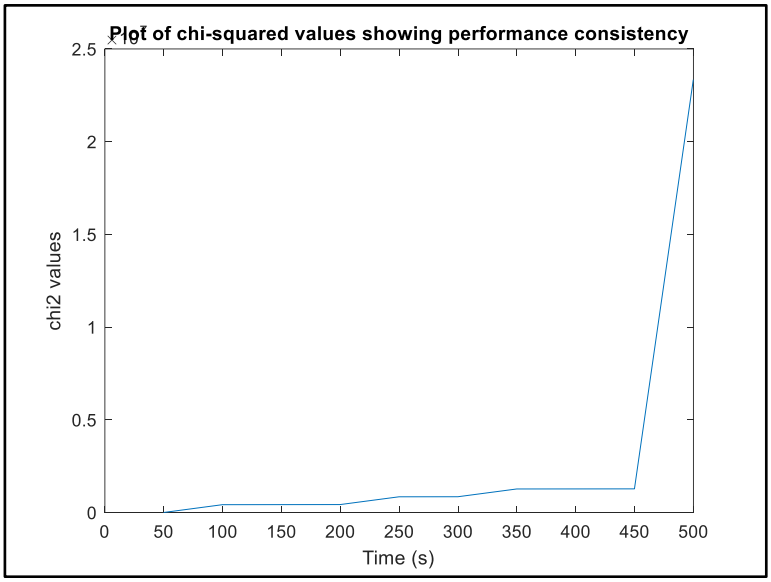

Figure 24 - showing the plot of the chi2 value over time after implementing the graph pruning algorithm

The results presented show that the changes have improved the performance speed as the optimization times have reduced greatly. The optimization was finished within 92 seconds compared to 125 seconds when no edges were deleted.

As can be observed from the graph, At the 450th time step, The chi2 value jumps up, this may be because of loop closure as it must adjust its value of estimates based on seeing a feature it has already seen before, while the estimates are consistent before this event. The chi2 values also increase constantly over the timesteps, specially within 200-250 and 300-350, signifying these are instances when the landmark observation edges may be deleted as there is an increase in difference between the expected and the predicted states.

The error performance of the vehicle state estimate has however, significantly depreciated, as seen in the graph. This makes sense as now we have multiple vehicle states which have no landmark edges attached to them, increasing their covariance estimates.

The results presented do reflect the changes made in the algorithm with respect to pruning, as seen in the simulator output, the landmark estimates are deviated from their true states slightly, signifying the absence of data required to improve these estimates.

The resulting error performance is very similar to the SLAM algorithm that retained all the information in the graph, although it had fewer edges, with very small landmark and vehicle covariances. With this approach, it is computationally less expensive to generate a map, with minimal difference between the shape of the map built and therefore this approach to simplifying graphs works as expected.

List of References

1. *Kretzschmar, H. and Stachniss, C., 2012. Information-theoretic compression of pose graphs for laser-based SLAM. The International Journal of Robotics Research, 31(11), pp.1219-1230.*
2. *S.J. Julier, COMP0130: Robot Vision and Navigation Lecture 7B slides 11 to 17*
3. *S.J. Julier, COMP0130: Robot Vision and Navigation Lecture 7A slide 29*
4. *S.J. Julier, COMP0130: Robot Vision and Navigation Lecture 7B slides 18 to 22*
5. *Khattak, S., Papachristos, C. and Alexis, K., 2020. Keyframe-based thermal–inertial odometry. Journal of Field Robotics, 37(4), pp.552-579.*

Appendix A –