Graph-Based Optimization and SLAM

Stefan Swandel
22043958
stefan.swandel.22@ucl.ac.uk

March 16, 2023

1 Question 1

1.a

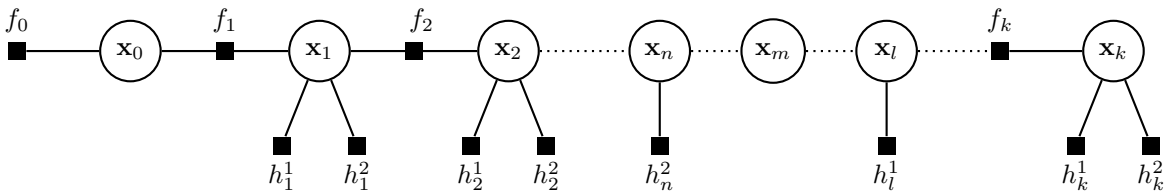


Figure 1: Factor graph with GPS and compass sensor measurements

A factor graph which can be used to predict a robot's pose over time and update it using GPS and compass sensors is presented in figure 1. In this factor graph, we have a single vertex type and four edge types:

1. **vehicle state vertices**
2. **initial prior edge**
3. **prediction edges**
4. **GPS observation edges**
5. **compass observation edges**

This factor graph represents a function proportional to the joint probability distribution of vehicle states. That is, it is a graphical representation of

$$f(\mathbf{x}_{0:k}|\mathbb{I}_k) \propto f(\mathbf{x}_0)\prod_{i=1}^{k} f(\mathbf{x}_i|\mathbf{x}_{i-1},\mathbf{u}_i)\prod_{i\in\mathcal{Z}^G}\mathrm{L}(\mathbf{x}_i|\mathbf{z}_i^G)\prod_{i\in\mathcal{Z}^C}\mathrm{L}(\mathbf{x}_i|\mathbf{z}_i^C) \tag{1}$$

where $f(\mathbf{x}_0)$ is the prior distribution of vehicle state, $f(\mathbf{x}_i|\mathbf{x}_{i-1},\mathbf{u}_i)$ is the probability distribution the next state given the current state and some control inputs, $\mathrm{L}(\mathbf{x}_i|\mathbf{z}_i^G)$ is the GPS measurement likelihood function, and $\mathrm{L}(\mathbf{x}_i|\mathbf{z}_i^C)$ is the compass measurement likelihood function.

The *initial edge* is the unary edge connected to the first vertex. The factor associated with this edge, $f_0(\mathbf{x}_0)$, is the probability distribution of $\mathbf{x}_0$ that incorporates all of our prior knowledge about the initial vehicle state. For the coursework, we assume that we know perfectly the vehicle begins at the origin. Thus, $f(\mathbf{x}_0)$ is a distribution concentrated entirely at $\mathbf{0}$. In equation form this is:

$$f_0(\mathbf{x}_0) = \begin{cases} 1 & \mathbf{x}_0 = \mathbf{0} \\ 0 & \mathbf{x}_0 \neq \mathbf{0} \end{cases} \tag{2}$$

Prediction edges are the edges that connect two vehicle state vertices. The factors on these edges are the transition probability distributions, $f(\mathbf{x}_i|\mathbf{x}_{i-1},\mathbf{u}_i)$, from (1). As mentioned, this is the distribution of the

vehicle state at time step i given the current vehicle state and some control inputs, $\mathbf{u}$. The equation for this factor is derivable from the process model:

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \triangle \mathrm{T}_{i-1}\mathbf{M}(\psi_{i-1})(\mathbf{u}_{i-1} - \mathbf{v}_{i-1}) \quad , \quad \mathbf{v}_{i-1} \sim N(\mathbf{0}, \mathbf{Q}_i)$$

$$\iff \mathbf{v}_{i-1} = \mathbf{M}^{-1}(\psi_{i-1})\Big(\frac{\mathbf{x}_i - \mathbf{x}_{i-1}}{\triangle \mathrm{T}_{i-1}}\Big) - \mathbf{u}_{i-1}$$

$$\implies f(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{u}_i) = f_{\mathbf{v}_{i-1}}\Big(\mathbf{v}_{i-1} = \mathbf{M}^{-1}(\psi_{i-1})\Big(\frac{\mathbf{x}_i - \mathbf{x}_{i-1}}{\triangle \mathrm{T}_{i-1}}\Big) - \mathbf{u}_{i-1}\Big) \tag{3}$$

Thus, this factor is the likelihood that the process noise at time step $i-1$ is the same as the prediction edge error, which is the difference between the predicted controls and the measured controls.

**Note that we must pay special attention when computing (3). Whenever we add angles together we must use the special operator $\oplus$. This special operator wraps the angle in $[-\pi, \pi]$ in order to avoid angular discontinuities. This special operator must be used in equation (3) when adding or subtracting headings.

GPS and compass observation edges are the edges attached to a single vertex. Note that, in the diagram, some vertices have a GPS and compass edge, some have one of the two, and some have none. This inconsistency is because compass and GPS data arrive at different rates, which means that at some timesteps, we will receive sensor data from both, one, or neither source.

The final two factors, h_i^1 and h_i^2, are GPS and compass measurement likelihood functions. The factor h_i^1 is the likelihood of the observed GPS measurement at time step i given the vehicle state. The equation for this is derivable from the GPS observation model as follows:

$$\mathbf{z}_i^G = \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \mathbf{M}(\psi_i) \begin{bmatrix} \triangle x_G \\ \triangle y_G \end{bmatrix} + \mathbf{w}_i^G \quad , \quad \mathbf{w}_i^G \sim N(\mathbf{0}, \mathbf{R}^G)$$

$$\iff \mathbf{w}_i^G = \mathbf{z}_i^G - \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \mathbf{M}(\psi_i) \begin{bmatrix} \triangle x_G \\ \triangle y_G \end{bmatrix}$$

$$\implies f(\mathbf{z}_i^G|\mathbf{x}_i) = f_{\mathbf{w}_i^G}\Big(\mathbf{w}_i^G = (\mathbf{z}_i^G - \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \mathbf{M}(\psi_i) \begin{bmatrix} \triangle x_G \\ \triangle y_G \end{bmatrix})\Big) \tag{4}$$

Thus, this factor is computed as the likelihood that the GPS observation noise at time step i is the same as the GPS observation edge error.

The compass measurement likelihood function can be derived in a similar manner using the compass observation model:

$$\mathbf{z}_i^C = \psi_i \oplus \psi_c \oplus \mathbf{w}_i^C \quad , \quad \mathbf{w}_i^C \sim N(\mathbf{0}, \mathbf{R}^C)$$

$$\iff \mathbf{w}_i^C = \mathbf{z}_i^C - (\psi_i \oplus \psi_c)$$

$$\implies f(\mathbf{z}_i^C|\mathbf{x}_i) = f_{\mathbf{w}_i^C}\Big(\mathbf{w}_i^C = (\mathbf{z}_i^C - (\psi_i \oplus \psi_c))\Big) \tag{5}$$

Thus, the h_i^2 factor is the likelihood that the compass observation noise at time step i is the same as the as the compass observation edge error.

With equations (2), (3), (4), and (5), we can use numeric techniques to optimize (1) to get the maximum likelihood estimate of the vehicles states.

1.b

i. To complete functionality of the `q1_b.m` script, we edit the following methods:

1. `VehicleKinematicsEdge.computeErrors`
2. `VehicleKinematicsEdge.linearizeOplus`
3. `drivebot.handlePredictToTime`

Method 1: `VehicleKinematicsEdge.computeErrors`
The `VehicleKinematicsEdge.computeErrors` method computes and stores the error between the observed vehicle kinematics measurements and the true vehicle kinematics measurements. That is, in this method we wish to compute

$$\mathbf{e} = \hat{\mathbf{u}}_k - \mathbf{z}_k$$

where $\mathbf{z}_k$ is the true measurement of control inputs associated with this edge and $\hat{\mathbf{u}}_k$ is the predicted control inputs. We compute $\hat{\mathbf{u}}_k$ using the process model assuming no noise:

$$\mathbf{x}_{k+1} = \mathbf{x}_{k+1} + \triangle \mathrm{T}_k \mathrm{M}(\psi_k)\mathbf{u}_k \implies \hat{\mathbf{u}}_k = \frac{1}{\triangle T_k}\mathrm{M}^{-1}(\psi_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) \tag{6}$$

Thus, using (6) we can compute the error as

$$\mathbf{e} = \frac{1}{\triangle T_k}\mathbf{M}^{-1}(\psi_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) - \mathbf{z}_k \tag{7}$$

So, we edited the `VehicleKinematicsEdge.computeErrors` to execute equation (7), making sure to wrap any angles, and store the result.

Method 2: `VehicleKinematicsEdge.linearizeOplus`
The `VehicleKinematicsEdge.linearizeOplus` method computes and stores the Jacobian of the vehicle kinematic edge error with respect to the connected vehicle states, $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$. So, we edit this method to use equation (7) to compute and store:

$$\frac{\partial \mathbf{e}}{\partial \mathbf{x}_k} = \frac{\mathbf{x}_{k+1}}{\triangle T_k}\frac{\partial \mathbf{M^{-1}}(\psi_\mathbf{k})}{\partial \mathbf{x}_k} - \frac{\mathbf{M^{-1}}(\psi_k)}{\triangle T_k} \tag{8}$$

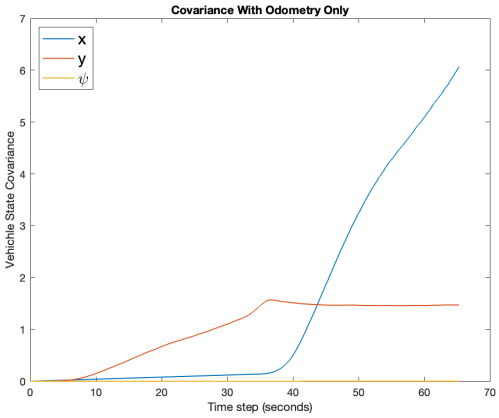
$$\frac{\partial \mathbf{e}}{\partial \mathbf{x}_{k+1}} = \frac{1}{\triangle T_k}\mathrm{M}^{-1}(\psi_k) \tag{9}$$

Method 3: `drivebot.handlePredictToTime`
The `drivebot.handlePredictToTime` updates the factor graph to represent the vehicle reaching a new time step. We do this by first creating a new vehicle state vertex and adding it to the graph. We then create a new vehicle kinematics edge and link it to the newly created and previous vertex. After this, we add the observed vehicle kinematics measurement and the information matrix (inverse of the event covariance matrix) of this measurement to the edge and add the edge to the graph. Finally, we use the observed vehicle kinematics edge measurement to predict and set the estimated vehicle state at the new vertex.

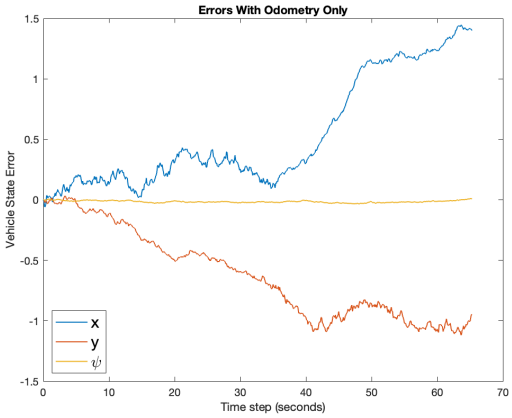
ii. In figure 2(a), we visualize the covariance of state estimates at each time step using only odometry data. We can see from this plot that initially, the covariance of the vehicle's x position is relatively low and is slowly increasing until the midpoint of the simulation. At the midpoint, the vehicle turns, and the rate at which the covariances increases changes. After the vehicle turns, the covariance in x changes from growing steadily to growing rapidly. This figure also shows that for the first half of the simulation, the y covariance steadily increases. Then, when the vehicle turns, the covariance in y stabilizes and remains stable for the remainder of the simulation. Finally, figure 2(a) shows that the covariance of the vehicle's heading (ψ) remains close to zero for the entire simulation.

The covariances of x and y are related to the squared errors in x and y. The squared errors in y increase more than the squared errors in x for the first half of the simulation, which is why we see the covariance of y increase more in the first half of the simulation. After the vehicle turns, the squared errors of y increase while the squared error of y remains similar from timestep to timestep. As a result, the covariance of x increases a lot after the vehicle turns, while the covariance of y does not change much.

Figure 2(b) shows the vehicle state errors at each time step. It shows that errors increase with time, which makes sense since, in the covariance plot, we observed our vehicle state estimate becoming



(a) Vehicle covariance using odometry sensor



(b) Vehicle state errors using odometry sensor

Figure 2: Q1b: Covariance and State Error Plots

increasingly uncertain. We will make more significant mistakes if we are more uncertain about our estimates. Furthermore, 2(b) shows that errors in y increase faster than errors in x in the first half of the simulation and then stabilize. Whereas errors in x increase slowly relative to y in the first half and then rapidly increase. This trend again aligns with what we observed in the covariance plot. The covariance shows that uncertainty in x is low, and uncertainty in y increases for the first half of the simulation. Thus, errors in x should also be low, and errors in y should be increasing, which is what we observe. The covariance also shows that uncertainty in x is quickly increasing, and uncertainty in y is relatively stable in the second half of the simulation. Thus, errors in x should also be rising, and errors in y should not increase much, which is what we observe.

We can observe how the residuals and chi2 values behave by changing the graph to optimize after ten timesteps rather than optimize once at the end. Doing this, we see that all residuals are 0. The residuals being zero also imply that the chi2 value of the graph is 0. The residuals and chi2 values behave this way because when we optimize our graph, we find state estimates that perfectly satisfy the constraint that predicted control inputs should equal observed control inputs. This claim follows from the definition of chi2 values in this scenario and the odometry noise:

$$\chi^2 = \sum_{\mathbf{e}} \mathbf{e}\Omega\mathbf{e}^T = \sum_{e_1,e_2,e_3} 25e_1^2 + 100e_2^2 + 3282.81e_3^2$$

where we are summing over all prediction edge errors. Thus,

$$\chi^2 \approx 0 \iff e_1^2 \approx e_2^2 \approx e_3^2 \approx 0 \quad \forall \quad \text{prediction edges}$$

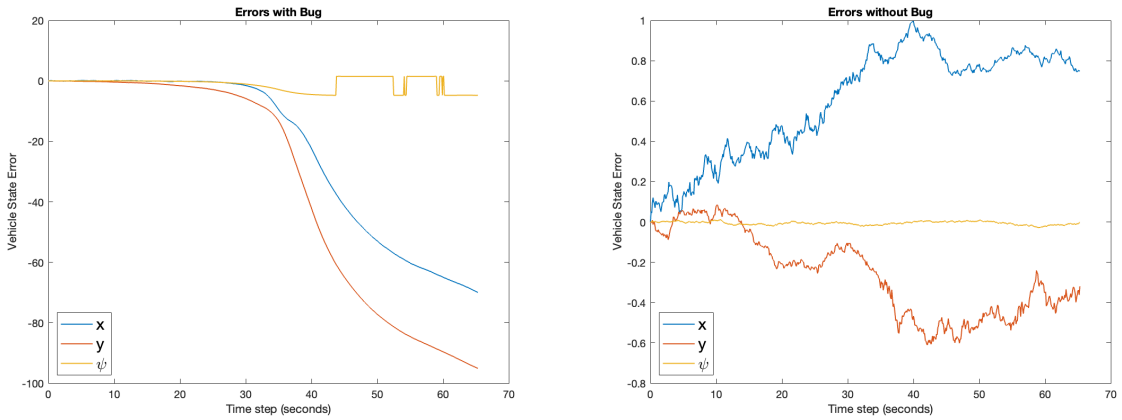
$$\implies \frac{1}{\triangle T_k}\mathbf{M}^{-1}(\psi_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) - \mathbf{z}_k \approx 0$$

1.c

i. We provide the plot generated with the bug in figure 3(a). This plot indicates that something in the code is incorrect because of the erratic behaviour of the heading error and the massive x and y errors.

We expect the compass sensor to improve our heading and position estimates. However, the exact opposite has happened. When using an odometry sensor, the heading error was consistently near zero. However, when using an odometry and compass sensor, the heading error worsens considerably and displays erratic behaviour. In addition to the deteriorating heading error, the x and y errors exhibit strange behaviour, growing to be massive by the end of the simulation. Given that the compass sensor only provides heading measurements, we can conclude that something in the compass measurement edge is incorrect.

We believe the error is that the `CompassMeasurementEdge.computeError` method does not wrap the heading angle before storing the error. Not wrapping the heading is problematic because the rest of the code assumes all angles are in $[-\pi, \pi]$. When optimizing the graph, we need the compass measurement error to decide which direction to step in. If the error drifts outside of $[-\pi, \pi]$ without



(a) Errors with bug

(b) Errors without bug

Figure 3: Buggy errors vs Fixed errors

being wrapped, it could cause the algorithm to step in the wrong direction and converge to a sub-optimal solution. Figure 3(a) is likely not the actual maximum likelihood estimate.

ii. To fix the bug in the code, we edit the `CompassMeasurementEdge.computeError` to wrap the heading error in the range of $[-\pi, \pi]$. We show the corrected plot in figure 3(b). Like the error plot with an odometry sensor, the heading error is now near zero for the entire simulation. Also, the new plot no longer has exploding x and y errors.

1.d

i. To incorporate the GPS sensor and make the code functional, we modify the following methods:

1. `GPSMeasurementEdge.computeError`
2. `GPSMeasurementEdge.linearizeOplus`
3. `drivebot.handleGPSObservationEvent`

Method 1: `GPSMeasurementEdge.computeError`
The `GPSMeasruementEdge.computeErrors` method computes and stores the error between the observed GPS measurement and the predicted GPS measurement. That is, in this method we wish to compute:

$$\mathbf{e} = \hat{\mathbf{z}}_k^G - \mathbf{z}_k^G$$

where $\hat{\mathbf{z}}_k^G$ is the predicted GPS measurement and $\mathbf{z}_k^G$ is the observed GPS measurement. We use the expected value of the GPS observation model to predict the GPS observation. The error then becomes:

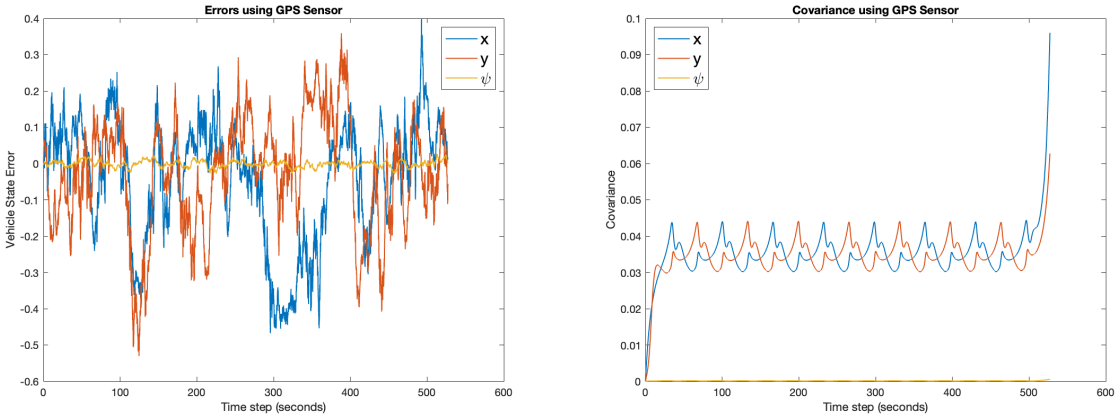
$$\mathbf{e} = \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \mathbf{M}(\psi_k) \begin{bmatrix} \triangle x_G \\ \triangle y_G \end{bmatrix} - \begin{bmatrix} z_k^1 \\ z_k^2 \end{bmatrix} \tag{10}$$

So, in `GPSMeasurementEdge.computeError` we compute and store the result of equation (10).

Method 2: `GPSMeasurementEdge.linearizeOplus`
The `GPSMeasurementEdge.linearizeOplus` method computes and stores the Jacobian of the GPS measurement edge error with respect to vehicle position. So, we edit this method to use equation (10) to compute and store:

$$\frac{\partial \mathbf{e}}{\partial [x_k, y_k]} = \frac{\partial}{\partial [x_k, y_k]} \begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Method 3: `drivebot.handleGPSObservationEvent`
In the `drivebot.handleGPSObservationEvent` method, we must execute the necessary steps to receive a new GPS observation measurement. First, we create a new GPS edge and link it to the vehicle state vertex at that timestep. Next, we assign that edge its measurement value (x and y position) and information matrix (inverse covariance matrix of the measurement). Finally, we add this newly created edge to the graph.



(a) Vehicle State Errors

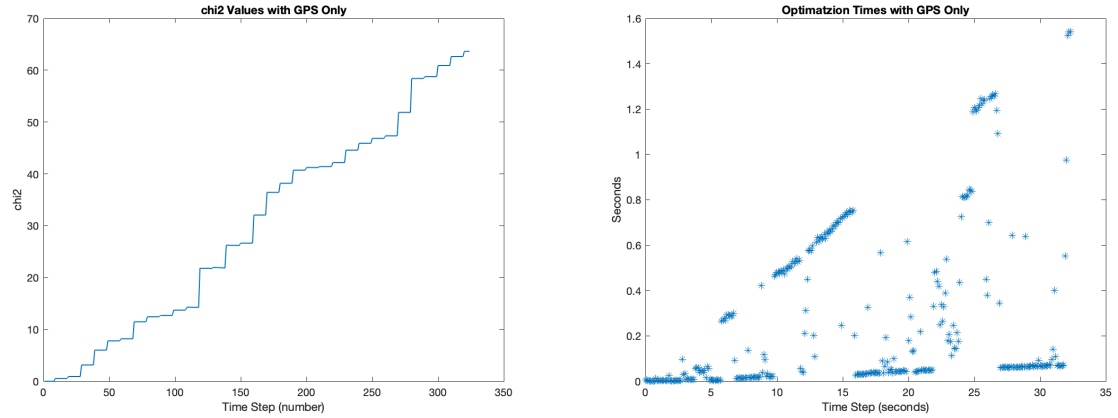
(b) Vehicle State Covariance

Figure 4: Q1d ii: Covariance and Errors using GPS Sensor

ii. Figure 4 shows the plots of vehicle state error (4a) and covariance (4b). In general, we observe that both errors and covariance of vehicle states are much lower when using a GPS sensor than using odometry only. Unlike the odometry-only case, the errors do not increase with time but rather oscillate around zero. The errors do not increase with time because the periodic GPS measurements allow us to correct any error we have accumulated from the odometry sensor between the current and previous GPS measurements.

Figure 4b shows that the covariance follows a cyclical pattern, where the covariance of x and y rise and fall. This cyclical pattern arises because the vehicle drives the same pattern multiple times. If we investigate further, we notice that the peaks of the covariance are approximately 33 seconds apart, which is the same as the amount of time it takes to drive a straight line. Thus, the uncertainty in the vehicle's x position is highest when the vehicle turns from the horizontal axis to the vertical axis, and the uncertainty in the vehicle's y position is highest when the vehicle turns from the vertical axis to the horizontal axis.

1.e



(a) chi2 values with GPS only

(b) Optimization times with GPS only

Figure 5: Q1ei: chi2 and optimisation times with GPS sensor only

i. In figure 5, we illustrate the simulation's chi2 values and optimization times when using a GPS sensor only. Focusing on figure 5a, we see that chi2 values increase with the size of the graph. They increase like a step function; they remain constant for several time steps and then jump. Looking more closely, we observe that the number of time steps between jumps is 10. The steps occur every ten timesteps because the period between GPS measurements is set to 1 (ten timesteps) in the simulator configuration. Sometimes the jump is significant, while it is almost unnoticeable other

times. The reason for the varying jump size follows from the fact that

$$\chi^2 = \sum_{\mathbf{e}\in\mathbf{E}} \mathbf{e}^{\mathrm{T}}\Omega_{\mathbf{e}}\mathbf{e}$$

where $\mathbf{E}$ is the set of all edge measurement errors in the graph and $\Omega_{\mathbf{e}}$ is the inverse covariance matrix of that measurement. After adding new elements to a factor graph, the new chi2 value of the graph can be written mathematically as:

$$\chi^2_{\text{new}} = \chi^2_{\text{old}} + \sum_{\mathbf{e}_{new}} \mathbf{e}^{\mathrm{T}}\Omega_{\mathbf{e}}\mathbf{e}$$

where $\mathbf{e}_{new}$ are all the new edge errors introduced by the changes to the graph. Note that since $\mathbf{e}^{\mathrm{T}}\Omega_{\mathbf{e}}\mathbf{e} \geq 0$, chi2 increases with the size of the graph. In our case, at each time step, we can either add a new state vertex or a new state vertex and a measurement edge. Thus, there are two scenarios:

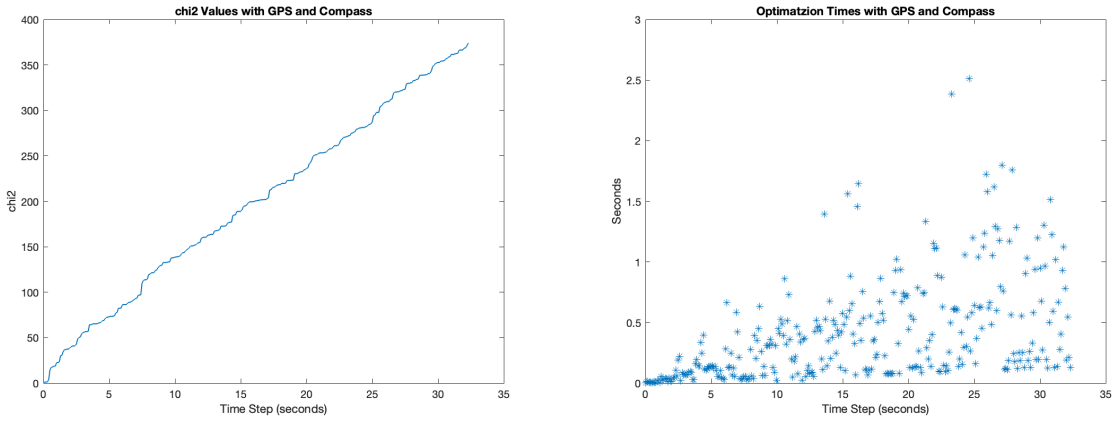
$$\chi^2_{\text{new}} = \begin{cases} \chi^2_{\text{old}} + \mathbf{e}_V^{\mathrm{T}}\Omega_{\mathbf{e}}\mathbf{e}_V & \text{, no GPS observation} \\ \chi^2_{\text{old}} + \mathbf{e}_V^{\mathrm{T}}\Omega_{\mathbf{e}}\mathbf{e}_V + \mathbf{e}_G^{\mathrm{T}}\Omega\mathbf{e}_G & \text{, GPS observation} \end{cases}$$

where $\mathbf{e}_V$ is the prediction edge error and $\mathbf{e}_G$ is the GPS observation edge error. It can be observed that prediction edge errors are extremely close to zero. Thus,

$$\chi^2_{\text{new}} \approx \begin{cases} \chi^2_{\text{old}} & \text{, no GPS observation} \\ \chi^2_{\text{old}} + \mathbf{e}_G^{\mathrm{T}}\Omega_{\mathbf{e}}\mathbf{e}_G & \text{, GPS observation} \end{cases} \tag{11}$$

Then, since Ω is diagonal, from equation (11), we can see that what determines if and how much chi2 increases is related to the GPS measurement error. The size of the measurement error will decide how much the chi2 value increases at a time step.

In order to optimize the factor graph, we must build the Hessian matrix. After each time step, the Hessian matrix increases in size. The amount the Hessian increases depends on what we add to the graph at that timestep. If we only add a prediction edge and a new vehicle state vertex, the number of diagonal and off-diagonal elements increases by 18. When we also add a GPS measurement edge to the graph, the number of diagonal elements of the Hessian increases by 27, which means we must compute an additional nine second-order derivatives at timesteps when we observe a GPS measurement. These increased computations cause the optimization times to follow two linear trends. The optimization time is less when we are not adding a GPS edge.



(a) chi2 values with GPS and compass

(b) Optimization times with GPS and compass

Figure 6: Q1eii: chi2 and optimisation times with GPS anb Compass Senors

ii. Figure 6 illustrates the behaviour of the chi2 values and optimization times when we use a compass and GPS sensor. From 6a, we see that the chi2 values are increasing. The rate of increase appears more like a straight line than a step function, as observed in i. The smoother increase is because the rate between compass measurements is 0.1, so we will observe a compass measurement at every

timestep, which implies adding an edge to the graph, which introduces a measurement error. For reasons explained in part i, we have

$$\chi^2_{k+1} \approx \begin{cases} \chi^2_k & \text{, odometry only} \\ \chi^2_k + \mathbf{e}_G^{\mathrm{T}} \Omega_{\mathbf{e}} \mathbf{e}_G & \text{, odeometry + GPS} \\ \chi^2_k + \mathbf{e}_G^{\mathrm{T}} \Omega_{\mathbf{e}} \mathbf{e}_G + \mathbf{e}_C^{\mathrm{T}} \Omega_{\mathbf{e}} \mathbf{e}_C & \text{, odometry + GPS + compass} \end{cases}$$

and since $\mathbf{e}_C^{\mathrm{T}} \Omega_{\mathbf{e}} \mathbf{e}_C \geq 0$ the chi2 value will increase by at least that amount at each time step. Occasionally we see a big jump like in ii. These big jumps occur when we observe a GPS event and have a large measurement error.

2 Question 2

2.a

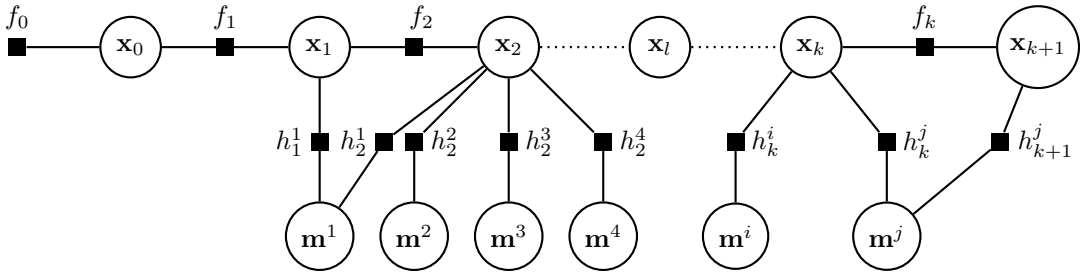


Figure 7: Example of Factor Graph for SLAM

An example of a graphical model which can be used to perform SLAM is presented in figure 7. This factor graph has two types of vertices and two types of edges:

1. **vehicle state vertex**
2. **landmark state vertex**
3. **prediction edge**
4. **landmark range bearing edge**

Vehicle state vertices are represented by the $\mathbf{x}$ nodes, while landmark state vertices are represented in the graph by the $\mathbf{m}$ nodes. Prediction edges connect two vehicle state vertices. Landmark range-bearing edges connect a vehicle state vertex to landmark state vertices.

Figure 7 is a graphical representation of a function that is proportional to the joint probability distribution of the vehicle's states given information about landmarks:

$$f(\mathbf{x}_{0:k}|\mathbb{I}_k) \propto f(\mathbf{x}_0)\prod_{i=1}^{k} f(\mathbf{x}_i|\mathbf{x}_{i-1},\mathbf{u}_i)\prod_{i\in\mathcal{Z}^L}\mathrm{L}(\mathbf{x}|\mathbf{z}^L)$$

where $f(\mathbf{x}_0)$ is the prior distribution of vehicle state, $f(\mathbf{x}_i|\mathbf{x}_{i-1},\mathbf{u}_i)$ is the transition probability distribution, and $\mathrm{L}(\mathbf{x}|\mathbf{z}^L)$ is the landmark measurement likelihood function. The prior distribution and state transition probability distributions behave the same as explained in 1a. The only difference from 1a is that we now need to compute the likelihood function of $\mathbf{x}$ given some landmark observations $\mathbf{z}^L$. That is we need the equation for $\mathrm{L}(\mathbf{x}|\mathbf{z})$, which is derivable from the landmark observation model as follows:

$$\mathbf{z}_k^L = \begin{bmatrix} \sqrt{(x^i-x_k)^2+(y^i-y_k)} \\ \tan^{-1}\left(\frac{y_i-y_k}{x^i-x_k}\right)-\psi_k \end{bmatrix} + \mathbf{w}_k^L \quad , \quad \mathbf{w}_k^L \sim N(\mathbf{0},\mathbf{R}_k^L)$$

$$\iff \mathbf{w}_k^L = \mathbf{z}_k^L - \begin{bmatrix} \sqrt{(x^i-x_k)^2+(y^i-y_k)} \\ \tan^{-1}\left(\frac{y_i-y_k}{x^i-x_k}\right)-\psi_k \end{bmatrix}$$

$$\implies \mathrm{L}(\mathbf{x}|\mathbf{z}^L) = f(\mathbf{x}_k|\mathbf{z}_k^L) = f_{\mathbf{w}}(\mathbf{w}_k^L = \mathbf{e}_k^L)$$

where $\mathbf{e}_k^L$ is the error in between predicted and observed landmark measurement. Thus, the likelihood of the vehicle's state at time step k is the same as the likelihood that the random noise of the landmark measurements is equal to the error between the predicted landmark measurement and the observed landmark measurement.

2.b

i. To complete the functionality of the code for question 2b, we modify the following methods:

(a) `LandmarkRangeBearingEdge.initialize`

(b) `LandmarkRangeBearingEdge.computeError`

(c) `LandmarkRangeBearingEdge.linearizeOplus`

(d) `drivebotSLAMSystem.handleLandmarkObservationEvent`

Method 1: `LandmarkRangeBearingEdge.initialize`
The `LandmarkRangeBearingEdge.initialize` method sets an initial value of our estimate of the landmark position. This value is needed to perform gradient descent. We set this initial value using our landmark observation model, assuming no noise. That is, our estimated landmark position is the solution to the following system of equations:

$$z^L_{x,k} = \sqrt{(x^i - x_k)^2 + (y^i - y_k)^2}$$

$$z^L_{y,k} = \tan^{-1}\left(\frac{y^i - y_k}{x^i - x_k}\right) - \psi_k$$

and, with some tedious algebra and use of the identity $\tan^2\theta + 1 = \sec^2\theta$, can be shown to reduce to

$$x^i = z^L_{x,k}\cos(z^L_{y,k} + \psi_k) + x_k \tag{12}$$

$$y^i = z^L_{x,k}\sin(z^L_{y,k} + \psi_k) + y_k \tag{13}$$

So, we edit `LandmarkRangeBearingEdge.initialize` to initialize our estimated landmark state as the solution to (12) and (13).

Method 2: `LandmarkRangeBearingEdge.computeError`
The `LandmarkRangeBearingEdge.computeError` method computes the error between the predicted landmark position measurement and the observed landmark measurement. So, we edit the method to compute and store the result of equation (14)

$$\begin{aligned}\mathbf{e} &= \hat{\mathbf{z}}^L_k - \mathbf{z}^L_k \\ &= \begin{bmatrix} \sqrt{(x^i - x_k)^2 + (y^i - y_k)^2} - z^L_{x,k} \\ \\ \tan^{-1}\left(\frac{y^i - y_k}{x^i - x_k}\right) - \psi_k - z^L_{y,k} \end{bmatrix}\end{aligned} \tag{14}$$

Method 3: `LandmarkRangeBearingEdge.linearizeOplus`
The `LandmarkRangeBearingEdge.linearizeOplus` method computes the Jacobian of the error with respect to the landmark state and vehicle state. So, we edit this method to compute and store:

$$\frac{\partial \mathbf{e}^i}{\partial \mathbf{x}_k} = \begin{bmatrix} \frac{(x_k - x^i)}{r^i_k} & \frac{(y_k - y^i)}{r^i_k} & 0 \\ \frac{(y^i - y_k)}{(r^i_k)^2} & \frac{(x^i - x_k)}{(r^i_k)^2} & -1 \end{bmatrix} \qquad \frac{\partial \mathbf{e}^i}{\partial \mathbf{m}^i} = -\begin{bmatrix} \frac{(x_k - x^i)}{r^i_k} & \frac{(y_k - y^i)}{r^i_k} & 0 \\ \frac{(y^i - y_k)}{(r^i_k)^2} & \frac{(x^i - x_k)}{(r^i_k)^2} & -1 \end{bmatrix} \tag{15}$$

Method 4: `drivebotSLAMSystem.handleLandmarkObservationEvent`
Finally, to achieve functionality, we edit `drivebotSLAMSystem.handleLandmarkObservationEvent` so we can handle a landmark event. If we observe a landmark, this method must modify our graph correctly. To do this, we first create a new landmark vertex if we have not seen the landmark yet; otherwise, we retrieve the existing landmark vertex. Then, we create a land-bearing edge and link it to the vehicle state vertex at that time step and the landmark vertex. We then encode the edge with the observed measurement value and the measurement information matrix. Finally, if we have

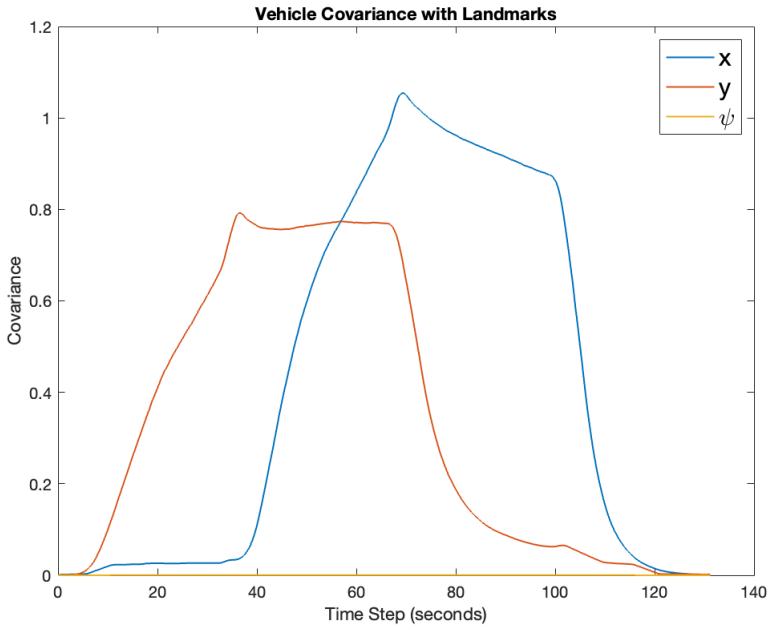
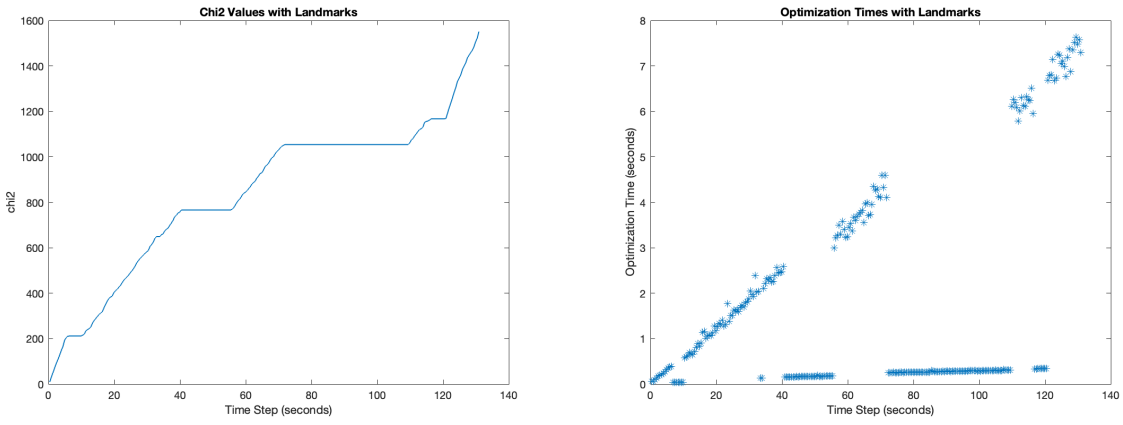


Figure 8: Q2b: vehicle covariance with landmarks

not seen this landmark yet, we initialize the vertex using the inverse landmark observation with no noise and add the landmark edge to the graph. If we have already seen this edge, we do not initialize the vertex but just add the edge to the graph.

ii. We illustrate the covariance of the states throughout the simulation in figure 8. During the first leg of the simulation, the covariance of x and y increase, but y increases much more rapidly than x. After the vehicle makes the first turn, the covariance in y stabilizes while the covariance in x begins a rapid increase. After the second turn, the covariance in x and y decrease, with y decreasing more rapidly. After the final turn, the covariance in x and y continues to decrease, but now x is decreasing more rapidly than y. Heading covariance remains close to zero the entire simulation. We notice that overall uncertainty in states is highest at the second turn, which is the state furthest from where loop closure occurs. As states get closer to where loop closure occurs, overall uncertainty decreases. Uncertainty is highest when we are far from the point of loop closure because when the loop closes, we can look back on the path we travelled and update it with new information. However, this information is more relevant to locations nearby. So our estimated location of landmarks far away from the loop closure will be affected the least, which implies our state estimates near those landmarks will not improve much, resulting in higher uncertainty.



(a) chi2 values with landmarks

(b) Optimization times with landmarks

Figure 9: Q2b: chi2 and optimzation times with landmarks

Figure 9a illustrates the chi2 value at each time step of the simulation. It increases with time but

does not increase at a constant rate. It cycles between phases of increasing linearly and phases of not increasing. To understand why the graph takes this shape, recall that the chi2 value for the factor graph is computed as

$$\chi^2 = \sum_{\mathbf{e}\in\mathbf{E}} \mathbf{e}^T \, \Omega\mathbf{e}$$

where $\mathbf{E}$ is the set of all errors associated with every edge in the graph. As we explained in question 2 part a, the factor graph for SLAM has two edge types, prediction and landmark edges. Thus, the chi2 value at a time step, $k+1$, can be written recursively as:

$$\chi^2_{k+1} = \begin{cases} \chi^2_k + \mathbf{e}_P^T \Omega \mathbf{e}_P & \text{, no landmark observed at time step } k \\ \chi^2_k + \mathbf{e}_P^T \Omega \mathbf{e}_P + \mathbf{e}_L^T \Omega \mathbf{e}_L & \text{, landmark observed at time step } k \end{cases}$$

which can be very closely approximated by:

$$\chi^2_{k+1} \approx \begin{cases} \chi^2_k & \text{, no landmark observed at time step } k \\ \chi^2_k + \mathbf{e}_L^T \Omega \mathbf{e}_L & \text{, landmark observed at time step } k \end{cases}$$

So, when we are not observing any landmarks, our chi2 value does not change significantly, but when we observe landmarks, we get measurement errors which cause the chi2 values to increase. If we cross-examine the chi2 values with the simulation, we notice that the sections where chi2 is flat are when the vehicle is not observing any landmarks. The sections where the chi2 rises are when the vehicle observes landmarks.

The optimization times in figure 9b follow two distinct linear trends; one trend with a much steeper slope than the other. The sections of the plot where the slope is steep correspond to timesteps we observe landmarks, while the sections with a lesser slope correspond to timesteps we observe no landmarks. These two distinct trends exist because we must build the Hessian matrix to optimize the factor graph. Building the Hessian takes longer at timesteps we observe landmarks. This extended time is because the Hessian is sparsely populated with elements on the diagonal and relatively few elements off the diagonal. The elements on the diagonal of the Hessian come from temporal prediction between two states. The off-diagonal elements align with landmarks and the timesteps from which we observe them.

The difference in optimization time occurs because when we are at a time step observing a landmark, we must expand the old Hessian with new diagonal and off-diagonal elements. These new elements are second-order partial derivatives that must be computed and placed in the matrix. In contrast, when we are at timesteps with no landmark observations, we only need to expand the Hessian with additional diagonal elements, which is less computationally expensive.

2.c

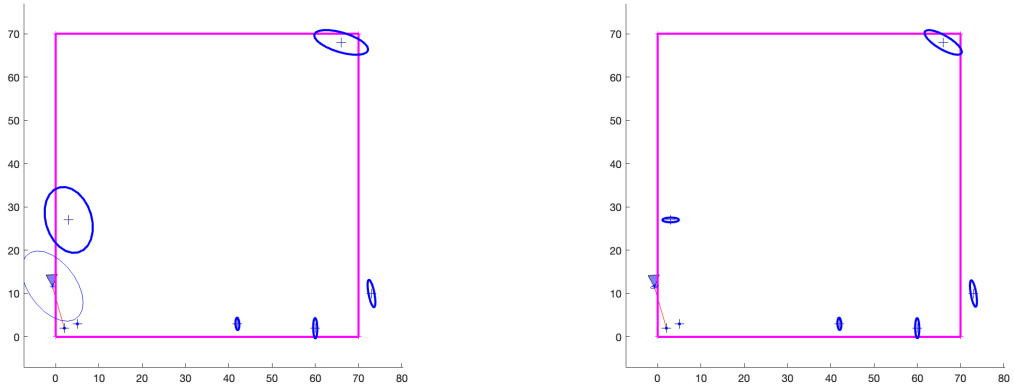
i. The number of vehicle poses stored is the same as the number of vehicle state vertices in a graph, and the number of landmarks initialized is the same as the number of landmark vertices in the graph. Thus, to count the number of vehicle poses and landmarks initialized, we edit the q2_c script to loop through all vertices and check using the isa method whether it is a landmark vertex or a vehicle state vertex, increment the total numbers depending on the result. Also, while looping through the vertices, we count each vehicle state vertex's total number of landmark observations. Since there is only one type of observation in this scenario, the average number of observations a robot makes at each time step is the same as the average number of landmark observations at each timestep. So, to compute this, we divide the total number of landmark observations by the total number of vehicle state vertices. Similarly, since every landmark observation edge connects to a landmark vertex, we compute the average number of observations each receives as the number of landmark observations divided by the number of landmark vertices.

Number of vehicle poses stored	Number of landmarks initialized	Average number of obvservations made by the robot at each time step	Average number of observations recived per landmark
5273	7	0.61123	460.4286

Table 1: Q2c results

ii. We present the results obtained from analyzing the graph in table 1. The results tell us that we store 5273 vehicle poses, meaning there were 5273 timesteps in the simulation and 5273 vehicle state vertices in the graph. Table 1 also reveals that we initialized 7 landmarks throughout the simulation. Since we only initialize a landmark once, this result implies that we observed seven unique landmarks. The robot's average number of observations at each time step is 0.61123. This result implies that if we choose a time step at random, then the probability that we observed at least one landmark is approximately 0.61123. Finally, we see in table 1 that the average number of observations received per landmark is 460.4286. This result implies that, on average, we observe every landmark at 460 timesteps.

2.d

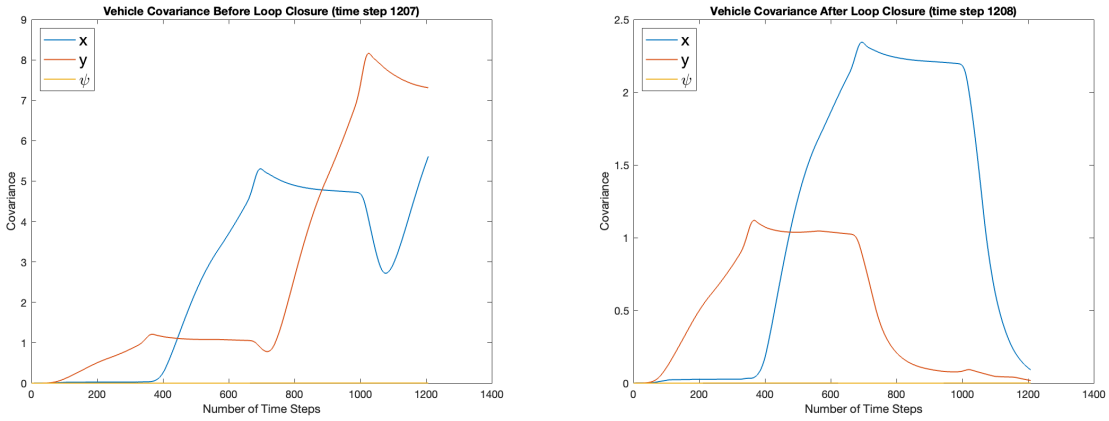


(a) Map immediately prior to loop closure

(b) map immediately after loop closure

Figure 10: Q2d: Map before and after loop closure

Figure 10 shows the map with landmark covariance ellipses before and after loop closure. The figure illustrates that the covariances surrounding the landmark positions decrease immediately after loop closure. The landmark covariances closest to the vehicle decreased the most, whereas those far away from the vehicle decreased less. Table 2 provides a summary of how much uncertainty in landmarks changed as a result of loop closure. Landmark 7 (the most recently seen landmark) decreased by 99.4%, the most out of any landmark, confirming our previous observation



(a) Covariance immediately before loop closure

(b) Covariance immediately after loop closure

Figure 11: Covariance before and after loop closure

Figure 11 shows the covariance in vehicle state one timestep before and after loop closure. We notice that the covariance before loop closure is relatively high, whereas, after loop closure, the covariance is much lower. This change in vehicle covariance is because when the robot recognizes it is in a location, it has been before, its confidence about where it is in its map goes up. Recognizing its location lets the

robot reflect on its travelled path and update the most likely states. Figure 11 also illustrates a difference in shape between covariance before and after loop closure. Before loop closure, the covariance increases similarly to how covariance increased using odometry only in question 1b. The covariance will continue to increase until the loop closes because without revisiting a location its seen before, the robot cannot use the map it is building to position itself. It relies solely on odometry data, which accumulates errors over time. Once the robot recognizes its location, it can understand where in the map it has created it is and use that information to infer state estimates better. In this case, the covariance looks like q2 b for the same reasons.

Landmark	Before LC	After LC	$\triangle$
1	0.0000	0.0000	0%
2	0.0000	0.0000	0%
3	0.0052	0.0051	-1.92%
4	0.0153	0.0149	-2.61%
5	0.0590	0.0470	-20.33%
6	2.6697	0.7705	-71.14%
7	21.2246	0.1305	-99.39%

Table 2: Determinants of landmark covariances before and after loop closure

3 Question 3

3.a

i. Without the prediction edges, we rely entirely on landmark range-bearing measurements to estimate the vehicle's state. If we have many landmarks and the loop closes, this may be suitable because the vehicle will only travel a little before seeing another landmark and thus will always have a good sense of where in the environment it has mapped it is. In other words, there will be lots of constraints the optimization will need to take into account, which will help narrow down the best solution. We must maximize the likelihood of vehicle state at many different timesteps across the map.

We saw in Q2c that, on average, we observe landmarks on approximately 60% of timesteps, so it is reasonable to assume that, in our case, removing all the prediction edges is suitable.

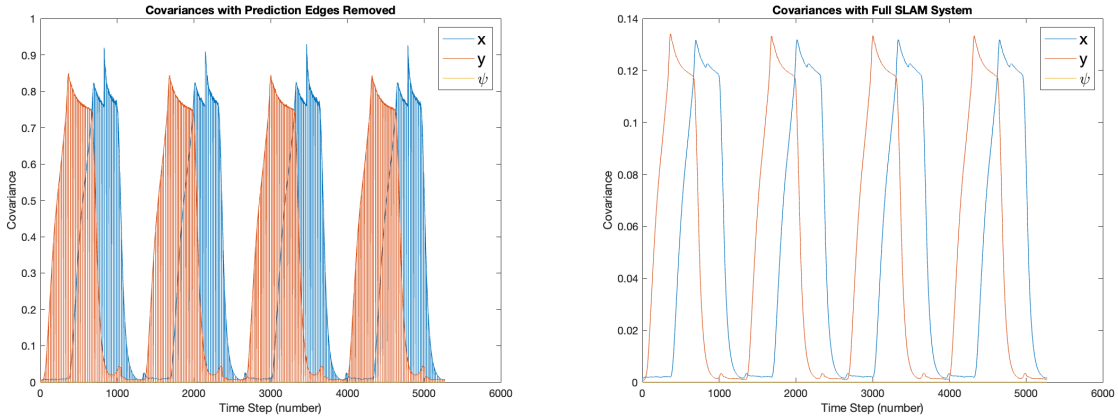
Removing prediction edges would be unsuccessful if there were long periods between landmark observations or if the loop never closes. Without the prediction edge guiding the vehicle through these sparsely populated areas, there are no constraints the algorithm must consider; thus, the range of possible estimates is extensive, and we may not find the best one. For example, suppose we only get one landmark measurement. In that case, the algorithm must find the vehicle state that maximizes the likelihood of the vehicle state at that timestep and is unconcerned about where the vehicle is at different timesteps. Additionally, if the loop never closes, the robot cannot use the map it has built to position itself.

ii. In `DrivebotSLAMSystem`, we edit the `deleteVehiclePredictionEdges` method. We edit the method to loop through all edges in the graph and, for each edge, check whether or not it is a prediction edge. If it is a prediction edge and the user has not specified to keep the first edge, then we delete it. If the user has specified to keep the first edge, we perform a check to see if this edge is the first prediction edge. If it is the first prediction edge, we keep it; otherwise, we delete it.

iii. Removing all prediction edges is the case where the algorithm fails. To understand why we cannot get a solution without the first prediction edge, consider the function the factor graph with no prediction edges represents. If the factor graph has no prediction edges, then to get vehicle state estimates, we seek to perform the following:

$$\text{argmax}_{\mathbf{X}} = \prod_{i \in \mathcal{Z}^{\mathcal{L}}} \text{L}(\mathbf{x}_k | \mathbf{z}_i^L) \tag{16}$$

We cannot get any information about the vehicle's position because we do not know where landmarks are situated in the world, only their distance and bearing from the vehicle. These measurements are useless without a general idea of the vehicle's global position, which the first prediction edge provides. By removing the first prediction edge, we are removing our prior information. In other words, we are removing our belief about the vehicle's position in the world, which makes landmarks impractical to us.

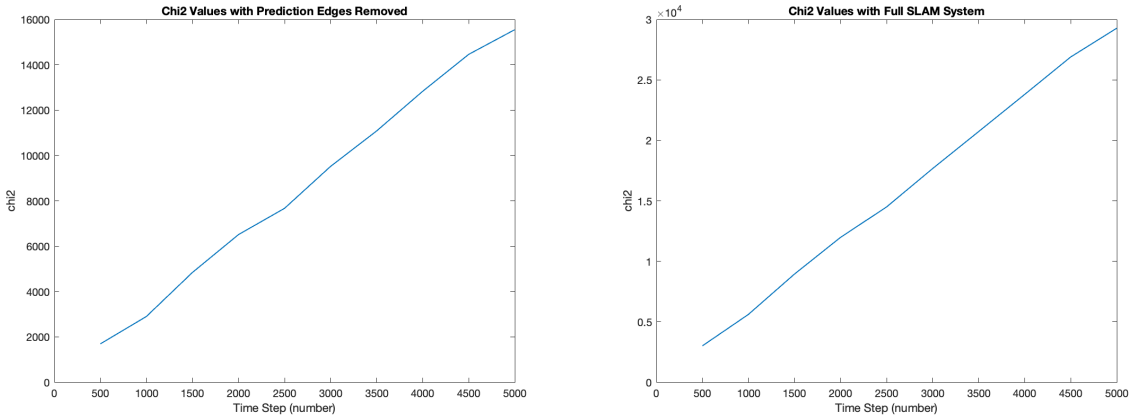


(a) Vehicle covariance with predictions edges removed

(b) Vehicle covariance with full SLAM

Figure 12: Q3a: Covariance with and without prediction edges

iv. In figure 12, we illustrate the difference in covariance between the full SLAM system and the system with all but the first prediction edge removed. Their general shapes are similar, which means the two graphs struggle with and excel at estimating the same states. However, the covariance of the full SLAM system is considerably lower, so it likely produces more accurate estimates. The covariance of the reduced system displays some strange behaviour. The covariance of x and y frequently suddenly drop to zero. From the covariance plot, the full SLAM system is superior.

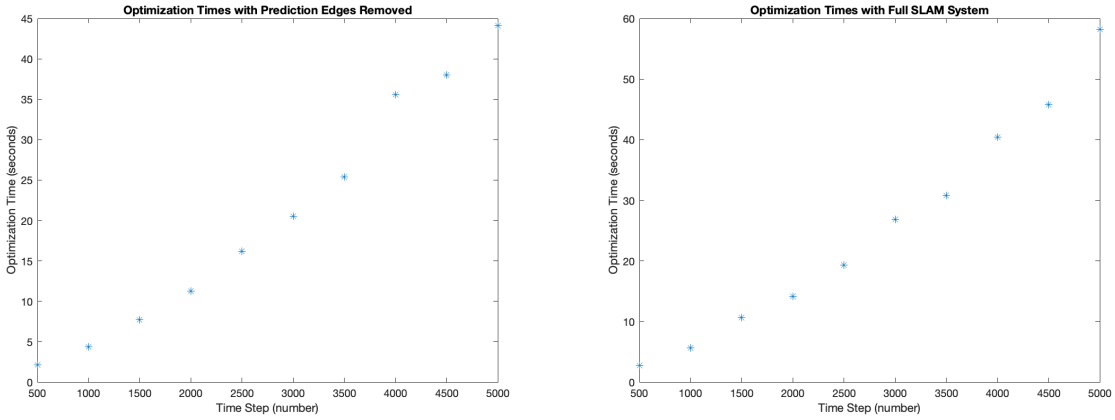


(a) chi2 with predictions edges removed

(b) chi2 times with full SLAM

Figure 13: Q3a: chi2 with and without prediction edges

In figure 13, we illustrate the chi2 plots of the full and reduced graphs. The plots show that the two systems follow a similar linear trend; however, the chi2 values of the full system are approximately twice as large as those of the reduced system. This difference is because we compute chi2 values as the weighted sum of errors of all edges. By removing all prediction edges, the reduced system has fewer edges to incorporate into its chi2 computation. Comparing these plots does not provide good insight into which graph is superior since the number of edges differs between the two graphs. A more appropriate measure would be the average error per edge.



(a) Optimization times without predictions edges

(b) Optimization times with full SLAM

Figure 14: Q3a: Optimization times with and without prediction edges

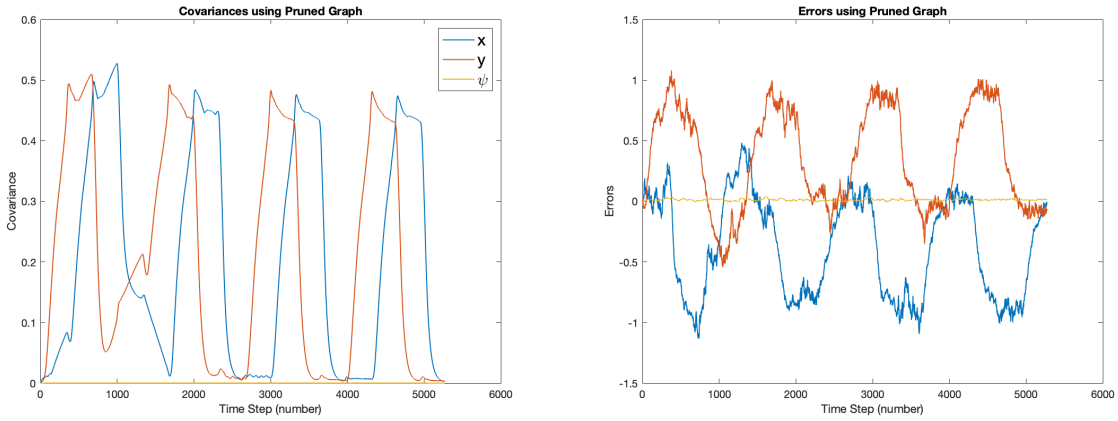
In figure 14, we illustrate the optimization times of the reduced and full systems. They follow similar trends, but the reduced system always has a lower optimization time. This difference exists because a graph with no prediction edges has a smaller Hessian matrix than the full graph.

Overall, this strategy does not seem to be successful. It has a marginally better optimization time but has high uncertainty in its estimates. Furthermore, the uncertainty follows strange behaviour, so we would not expect it to be superior to the full SLAM system.

3.b

i. At every timestep, we must add a vehicle state vertex and a prediction edge to the graph. Additionally, if we observe a landmark, we must add a landmark edge; if it is a new landmark, we must also add a landmark state vertex to the graph. Thus, the factor graph grows at each timestep. *Graph pruning* is a general term for methods that remove elements from the graph to prevent the graph from growing too large. One possible way to implement graph pruning is to remove some landmark measurement edges. In our solution, we remove every second landmark observation edge of a landmark vertex. Put another way, when the vehicle observes a particular landmark, we only keep the observation from every second timestep. The logic behind this is that subsequent edges contain similar information, so losing one will not cause a significant impact when optimizing for the vehicle states.

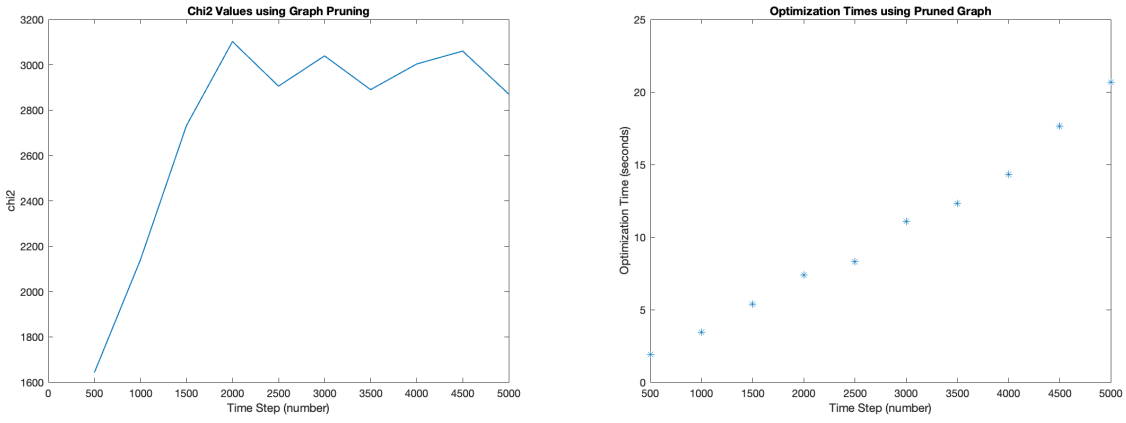
ii. We implement our graph pruning strategy by adding a `pruneGraph` method to the `DrivebotSLAMSystem` that works as follows. We loop through every vertex in the graph and check whether it is a landmark state vertex. If it is a landmark state vertex, we loop through all its edges, deleting every second edge. Additionally, we add a flag variable named `prune` in the `drivebotSLAMsystem` that indicates whether or not to prune the graph. If the `prune` variable is true, we call `pruneGraph` in the optimize method; otherwise, we do not. The prune variable is changeable via an `enalbePrune` method we created. So, when we run the `q3_b` script, we set the prune variable to true using the `enablePrune method`.



(a) Covariance with pruned graph

(b) Errors using pruned graph

Figure 15: Q3b: Covariances and Errors using purned graph



(a) Q3ba: chi2 with pruned graph

(b) Q3a: Optimization times with pruned graph

Figure 16: Q3b: chi2 and optimization times with pruned graph

iii. We evaluate the performance of this pruning method by visualizing the covariance and errors of states, as well as the chi2 values of the graph and optimization times. Figure 15 shows the covariance

and errors of states. Errors and covariance follow similar patterns, meaning both systems struggle and excel in the same areas. However, these results indicate that the pruned graph performs worse than the complete graph. On average, errors and covariance under the pruned graph are higher than in the full SLAM system. The covariance is around four times higher than the full SLAM system. This lesser performance is because we are using less information to estimate states. While the performance of the pruned graph is worse than the full graph, it outperforms the graph with all prediction edges removed. The pruned graph still gives reasonable results.

Figure 16 illustrates the chi2 values of the graph and optimization time. The optimization time is significantly faster with the pruned graph compared to the full SLAM system and the system with no prediction edges. Specifically, the pruned graph optimizes in a third of the time of the full graph and a half of the time of the prediction-edgeless graph. The chi2 values of the pruned graph are also significantly smaller than the full system. This difference is because chi2 is the weighted sum of all edge errors on the graph, and since pruning the graph removes half of the landmark edges, we sum together fewer errors. So, it is natural to expect the chi2 value to be lower.

Overall, the pruned graph has worse and more uncertain estimates than the full SLAM system but better and more certain estimates than the prediction-edgeless graph. Additionally, the pruned graph optimizes in substantially less time than the other graphs. Thus, if optimization time is a concern for someone using this pruning method may be suitable because the state estimates it gives are still reasonable. This pruning method allows people to trade off performance for less optimization time.