1 Question 1

1.a Factor Graph

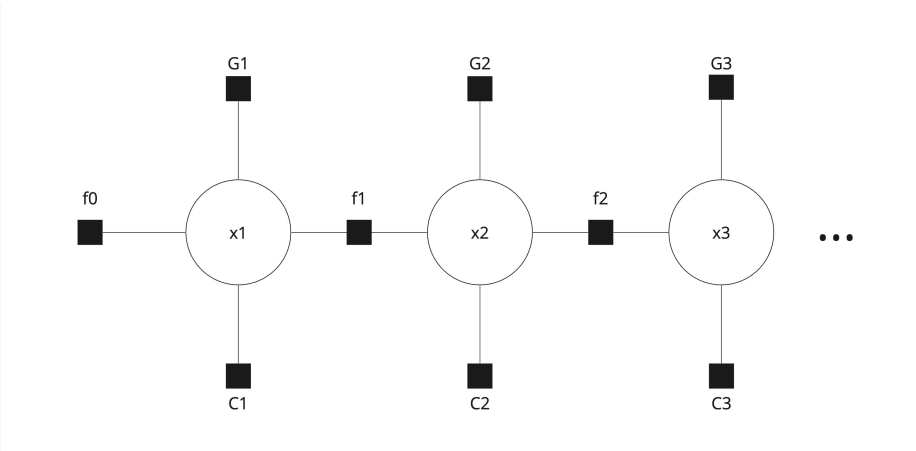

Figure 1: Factor Graph

Vertices:
X_i: Represents the state of the vehicle at time i, including position, heading.
Edges:
$f_i(X_{i-1}, X_i)$: Represents the vehicle kinematics constraint between state X_{i-1} and X_i. The relationship can be defined as a function of motion model, for instance, $f_i(X_{i-1}, X_i) = \phi(X_{i-1}, u_i)$, where u_i is the control input at time i, and ϕ is the motion model.
$G_i(X_i)$: Represents the GPS measurement for state X_i. This can be expressed as $G_i(X_i) = X_i^{gps}$, where X_i^{gps} is the GPS measurement.
$C_i(X_i)$: Represents the compass measurement for state X_i. This can be modeled as $C_i(X_i) = X_i^{compass}$, where $X_i^{compass}$ is the compass measurement.
Model Description:
At each time step i, the state X_i is connected to the previous state X_{i-1} through a motion model encapsulated by the function f_i. This function takes into account the vehicle's dynamics and the control inputs to provide a prediction of the next state. Additionally, sensory measurements are incorporated to correct the state estimates and mitigate the accumulation of errors. The GPS measurements G_i provide absolute positioning information, while the compass measurements C_i supply orientation data. Finally overall model can be defined as $f_0 f_1 ... f_n = f_0(x_0) f_1(x_1, x_2) ... f_n(x_n, x_{n+1})$ ...

1.b DriveBotSLAMSystem

1.b.i

In *DriveBotSLAMSystem.m*, the missing function is *handlePredictToTime*. It calculated the next state of the vertex by using odometer sensor. The code is applied according to the equation:

$$x_{k+1} = x_k + \triangle T_k M(\psi_k)(u_k + v_k)$$

where x is the state of the vertex at the time k, M is a transformation matrix during time interval $\triangle T_k$

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

ψ_k is a heading angle collected from odometer, and odometry is defined as u_k with process noise v_k. Later, it will create vehicle kinematic edge that is attached to the vertex, and search for the next vertex.

In *VehicleKinematicsEdge.m*, it is needed to define functions to compute error and Jacobian matrix. The error is calculated by reversing the current vertex and referencing with the previous one to compare with the measurement z_k regarding to the equation

$$E_k = \frac{1}{\triangle T_k} M(\psi_k)^{-1}(x_{k+1} - x_k) - z_k$$

1.b.ii

The figure of errors (Figure 3.) shows that there are no offset between ground truth and result. The reason of this error pattern is that the current model only use process model with odometry measurements. So when the errors are computed, it is only the odometry measurements and the ground-truth data being compared, which are the same data in this case because other sensor data is not involved and odometry measurements are treated as the ground-truth in the model configuration. Same for the residuals, because there is no difference between the prediction and the ground-truth, all residuals appears to be zero in the result.

Figure 2 presents the covariance of a vehicle, which indicates the estimated uncertainty in its state predictions. Initially, the uncertainty in the vehicle's y-axis position increases with each time step until it plateaus around time step 35. In contrast, the x-axis position uncertainty remains stable until it begins to increase sharply after time step 35. This pattern is attributed to the vehicle making a 90-degree turn at time step 35, leading to shifts in covariance. Before the turn, movement along the x-axis decreases x-position uncertainty while increasing y-position uncertainty. After the turn, as the vehicle moves along the y-axis, the certainty in the y-position improves, and the certainty in the x-position decreases. The vehicle's heading angle (theta) covariance stays at 0 throughout, indicating no estimated uncertainty in this parameter.

Based on the chi-square (chi2) analysis, plotting the chi2 directly after a single optimization step at the loop's end results in a singular data point. To investigate system behavior, we adjusted the optimization period to 100 using "drivebotSLAMSystem.setRecommendOptimizationPeriod(100)." This modification revealed that the chi2 value escalates as the vehicle distances itself from the initial point, correlating with an increase in uncertainty and, consequently, a higher chi2 value.

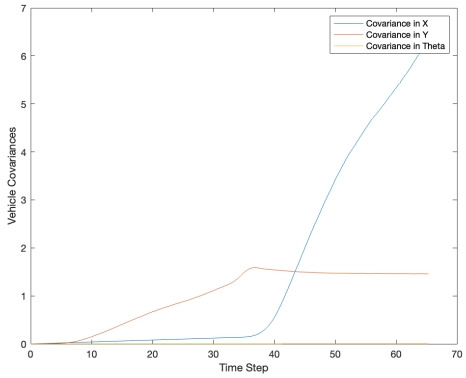

Figure 2: Vehicle Covariance

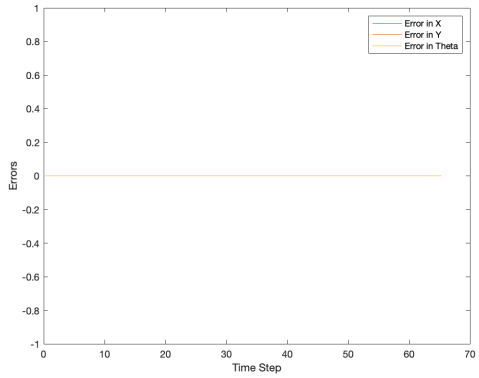

Figure 3: Errors

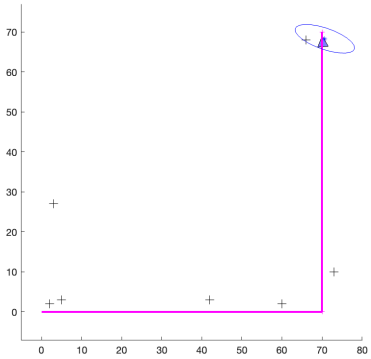

Figure 4: Simulation Output

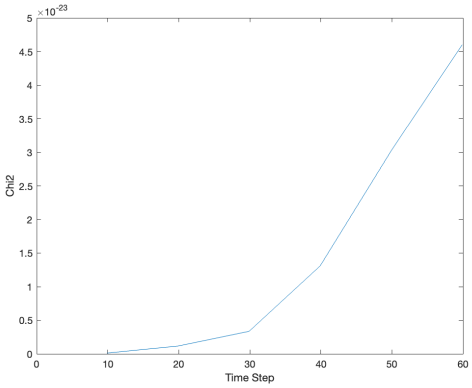

Figure 5: Chi2 with Optimization Period 100 (without log)

1.c Fixed Compass Measurement Bug

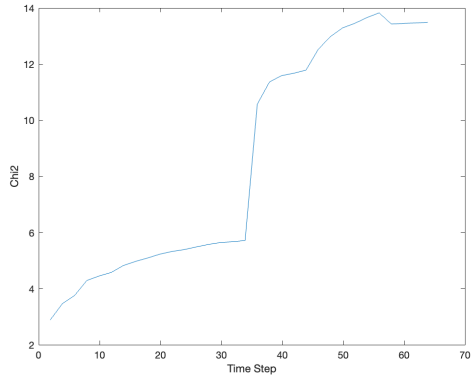

Figure 6: Chi2 before debugging

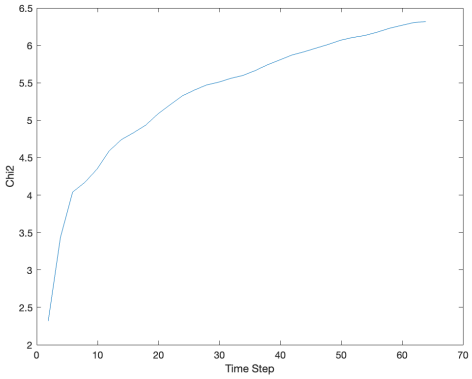

Figure 7: Chi2 after debugging

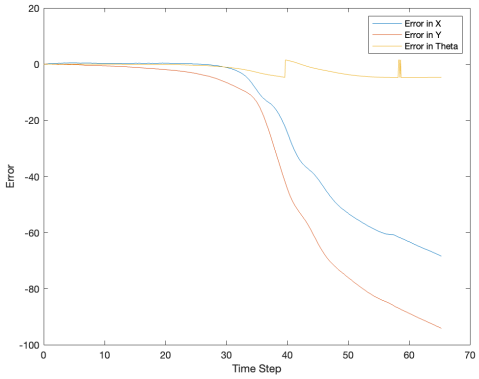

Figure 8: Errors before debugging

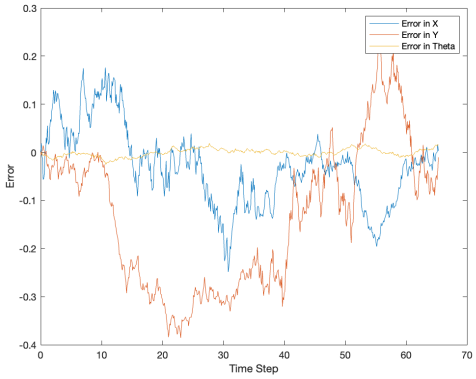

Figure 9: Errors after debugging

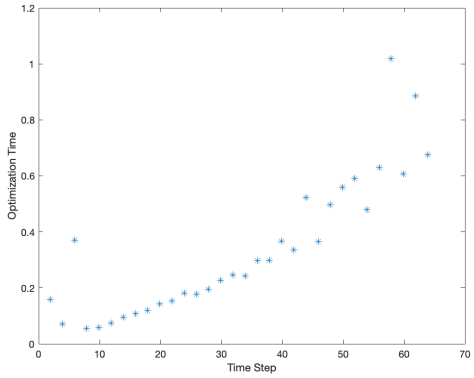

Figure 10: Optimization time before debugging

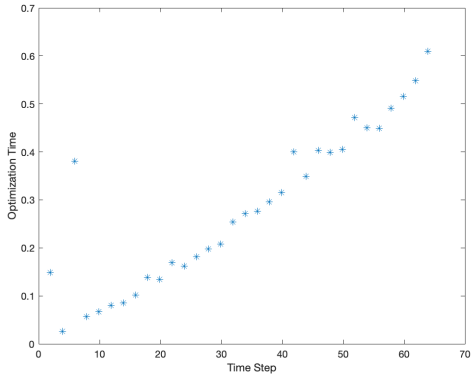

Figure 11: Optimization time after debugging

1.c.i

As observed in the error graph prior to debugging, the heading error exhibits some outliers, and the errors in x and y consistently decrease, trending towards negative values. This provides evidence that the 'errorZ' variable, calculated from 'CompassMeasurement', is contributing to the bug.

1.c.ii

```
this.errorZ = g2o.stuff.normalize_theta(x(3) + this.compassAngularOffset -
    this.z);
```

Specifically, the issue arises because the angle is not mapped within the interval $[-\pi, \pi]$. By addressing this bug as code mentioned above, we observe a gradual increase in the χ^2 value and optimization time without irregularities in its shape. Furthermore, the error graph demonstrates an improvement; however, it remains suboptimal as it does not account for GPS data.

1.d Handle GPS Measurement

1.d.i

In *DriveBotSLAMSystem.m*, the implementation is applied in *handleGPSObservationEvent* Function. It is used to create an observation edge relying on GPS sensor by calling *GPSMeasurementEdge* class.

To compute the error of GPS measurement E_k^G, it uses the position of the edge $\begin{bmatrix} x_k \\ y_k \end{bmatrix}$ comparing with the measurement or ground truth of the position z_k^G with added position offset $\begin{bmatrix} \triangle x_G \\ \triangle y_G \end{bmatrix}$, as mentioned below

$$E_k^G = \begin{bmatrix} x_k \\ y_k \end{bmatrix} + M(\psi_k) \begin{bmatrix} \triangle x_G \\ \triangle y_G \end{bmatrix} - z_k^G$$

where matrix $M(\psi_k)$ is an transformation matrix, which is multiplied with the offset to convert from fixed-platform coordinate to global coordinate,

$$M(\psi_k) = \begin{bmatrix} cos(\psi_k) & -sin(\psi_k) \\ sin(\psi_k) & cos(\psi_k) \end{bmatrix}$$

and also defines Jacobians $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ since it needs position of x-axis and y-axis.

1.d.ii

According to Figure 13, it can be an evidence that the system using GPS sensor is accurate since the value of errors are between -1 and 1, for x and y position. However, the covariance graph in Figure 12 shows that the uncertainty of x and y positions depends on each other since the values rise and fall in loop until near the end of the graph where they increases abruptly. This happens because the system has no detection where the place it has visited. In other words, this system has no loop closure event to reduce the number of predicted landmark observation when there are 2 similar vertex position. It is resulted in too many observation and state vertices at the end of loop and cause occlusion of data.

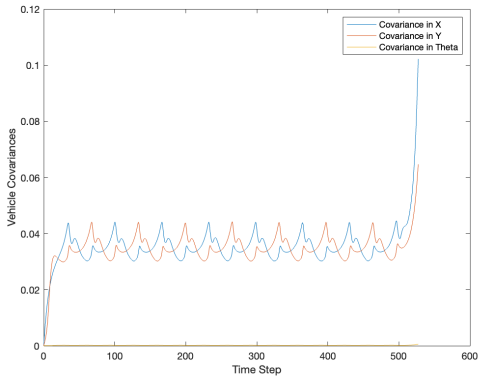

Figure 12: Vehicle Covariance

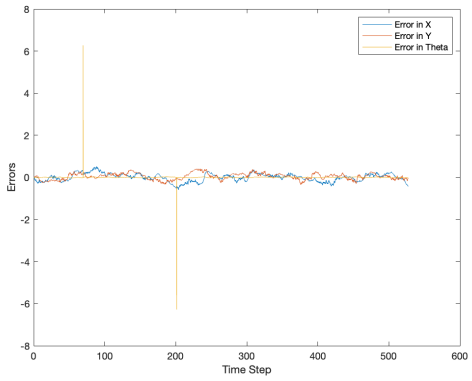

Figure 13: Errors

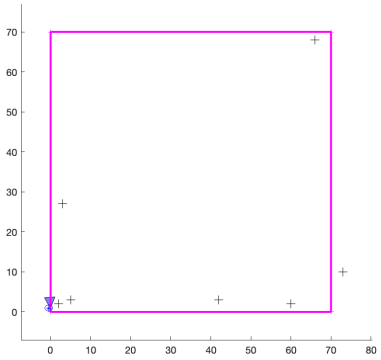

Figure 14: Simulation Output

1.e Comparison Between With and Without Compass Measurement

1.e.i

In this section, we only used GPS sensor to gather data. Referring to Figure 15, a significant χ^2 value is observed, primarily attributed to the sole use of the GPS sensor without integrating compass data. This high χ^2 value indicates a substantial discrepancy between the expected and observed data points, suggesting increased uncertainty in position estimates derived solely from GPS measurements. Moreover, the optimization process, which is executed at every timestep as depicted in Figure 16, requires a higher computational effort. This increased optimization time can be attributed to the frequent adjustments needed to align with GPS data, which, lacking compass guidance, introduces higher variability and complexity in achieving convergence.

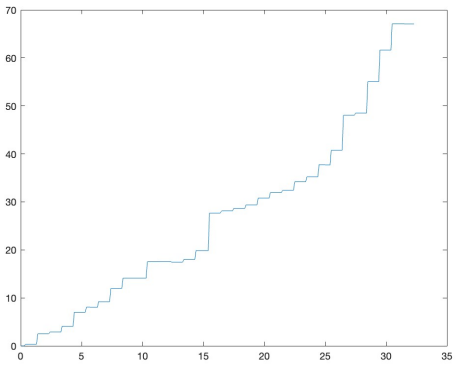

Figure 15: Chi2 without compass measurement

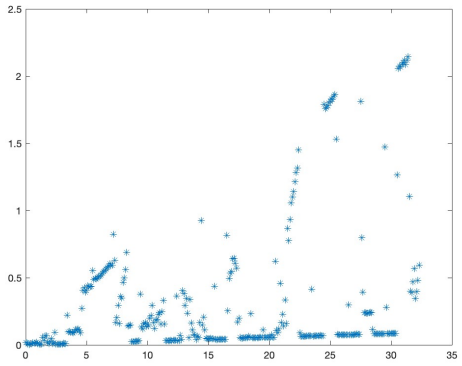

Figure 16: Optimization time without compass measurement

1.e.ii

In contrast, when compass data is incorporated into the analysis, as shown in Figure 17, there is a notable reduction in the χ^2 value. This decrease represents an improvement in the alignment between expected and observed data points, emphasizing the compass's role in reducing positional uncertainty. The integration of compass data helps anchor the directional information, which complements the GPS data by providing a more accurate and reliable estimate of orientation and position. Consequently, the optimization process, as illustrated in Figure 18, exhibits a reduction in computation time compared to the scenario where only GPS data is used. This efficiency increases from the additional information provided by the compass data, which improve the optimization by offering a more sure direction, so simplifying the path to convergence.

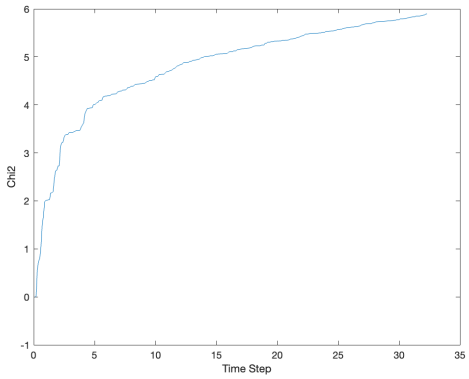

Figure 17: Chi2 with compass measurement

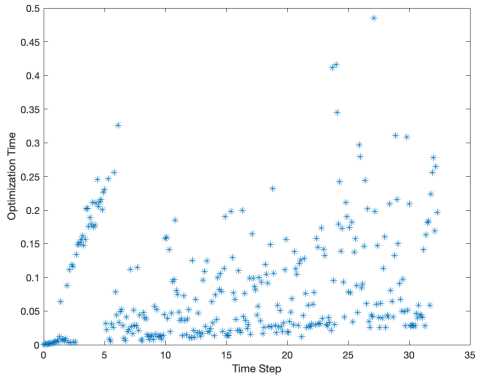

Figure 18: Optimization time with compass measurement

The observed differences in χ^2 values and optimization times between using only GPS data and integrating compass data underscore the importance of multi-sensor fusion in enhancing the accuracy and efficiency of navigational and positioning systems as mentioned in [3].

2 Question 2

2.a Factor Graph

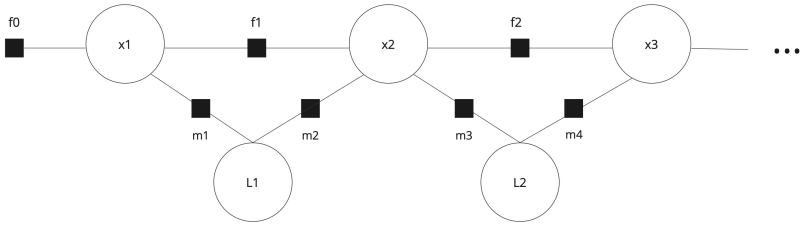

Figure 19: Factor Graph

Vertices:
X_i: Represents the state of the vehicle at time i.
L_j: Represents the j^{th} landmark or feature in the environment.
Edges:
$f_i(X_{i-1}, X_i)$: Represents the vehicle kinematics constraint, as described previously.
$m_k(X_i, L_j)$: Measurement from state X_i to landmark L_j. This can be represented by an equation such as $m_k(X_i, L_j) = h(X_i, L_j)$, where h is a function that maps the predicted observation of landmark L_j given the vehicle's state X_i.
Model Description:
Different as question 1, this task use landmark to get better result. The state of the vehicle at time i is represented by vertex X_i, while landmarks or features in the environment are denoted by L_j. The kinematic constraint of the vehicle moving from state X_{i-1} to X_i is modeled by the edge function $f_i(X_{i-1}, X_i)$. The edge $m_k(X_i, L_j)$ represents the measurement from the vehicle's state X_i to a landmark L_j, which is described by a function h mapping the predicted observation of landmark L_j given the vehicle's state X_i. This is a general factor graph, in real task, number of landmark vertices connected to each vehicle state vertex can be different. Finally overall model can be defined as $f_0 f_1 ... f_n = f_0(x_0) f_1(x_1, x_2) ... f_n(x_n, x_{n+1})$ ...

2.b Handle Landmark Observation

2.b.i

In section Q2b, the implementation focuses on processing landmark observations to enhance the graph-based SLAM system. The method `handleLandarkObservationEvent` iterates over each landmark observation, performing the following key steps as [4]:

1. For each landmark measurement, the system either retrieves an existing landmark vertex from the graph or creates a new one if the landmark has not been observed before.
2. A new `LandmarkRangeBearingEdge` object is instantiated. This object represents an edge in the graph that links the current vehicle state vertex with the landmark vertex, encoding the spatial relationship between the vehicle and the landmark based on the latest observation.
3. The vertices associated with the current vehicle state and the observed landmark are set as the endpoints of the edge, establishing a direct link within the graph that reflects the current observation.
4. The measurement data, consisting of the range and bearing to the landmark, is attached to the edge as its observation.
5. The information matrix, which is the inverse of the covariance matrix associated with the landmark observation, is computed and set for the edge.
6. If the landmark vertex was newly created during this process, the edge is initialized.
7. Finally, the newly created and configured edge is added to the graph. This addition expands the graph with new information about the environment, incrementally improving the SLAM system's map and vehicle trajectory estimates.

2.b.ii

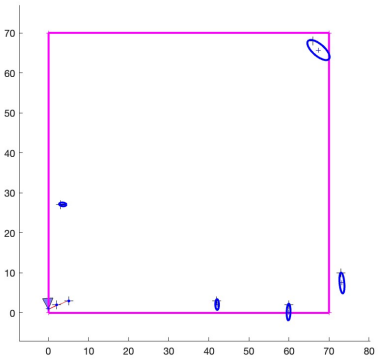

Figure 20: Simulation Output

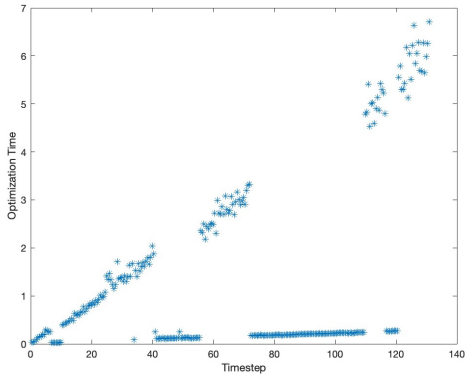

Figure 21: Optimization time

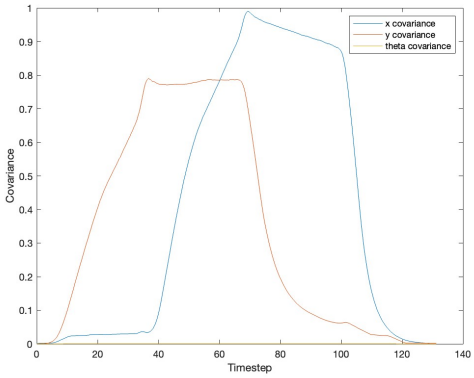

Figure 22: Vehicle Covariance

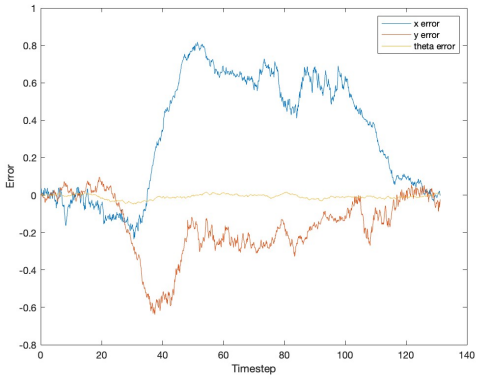

Figure 23: Vehicle Error

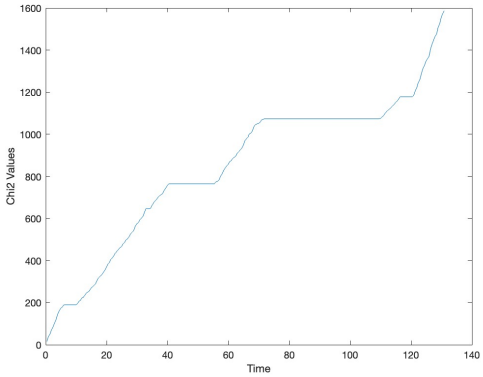

Figure 24: Chi2 Value

Upon examining Figure 21, we observe that the optimization time significantly decreases at the timestep when loop closure occurs. This reduction in optimization time can be attributed to the fact that the loop closure provides additional information that helps in correcting the trajectory estimation more efficiently. This efficiency arises because the system can now leverage the loop closure data to reconcile the accumulated drift, allowing for a quicker convergence of the optimization algorithm.

Similarly, Figure 22 reveals that the uncertainty, and consequently the covariance, reaches its peak when the vehicle is farthest from the starting point. This is indicative of increased uncertainty in the vehicle's position due to the accumulation of errors over time. Interestingly, the heading error and covariance are notably lower compared to those for the x and y positions, suggesting that the vehicle's orientation is determined with greater accuracy. This is because of the vehicle's motion model and sensor fusion algorithms that provide more reliable estimates of orientation than position.

The χ^2 values, calculated as the absolute deviation of the measured value from its expected value, exhibit a gradual increase with each timestep, as illustrated in Figure 24. However, an

exception occurs at the point of loop closure, mirroring the trend observed in the optimization time graph. This pattern is because of the impact of loop closure in correcting the trajectory estimation, thereby reducing the overall system error.

2.c Properties of the graph

2.c.i

In *q2_c.m*, we implement the method to find the total number of vehicle state vertex and the total number of landmarks, including the calculation of the average number of landmark edges at each observation vertex, and the average number of edges attached to each landmark.

We access all the vertices of the graph, and then check the class each vertex if it is a landmark or vehicle vertex. At the end, we get the the number of landmarks and the number of vehicle vertex. Then, we will count the the number of edges attaching to each landmark, and divide by the total number of vehicle vertex to find the average number of landmark edges at each observation vertex. The average number of edges attached to each landmark is to divide the number of edges attaching to each landmark by the number of landmarks.

- The total number of vehicle state vertex: 5273
- The total number of landmarks: 7
- The average number of landmark edges at each observation vertex: 0.61142
- The average number of edges attached to each landmark: 460.571

2.c.ii

From the graph in Figure 25, it shows that there are 4 loops of rise and fall of covariance values, meaning the system travels around this 4 times and, in Figure 26, the graph moves as similar as cosine function, but the amplitude increases as the time passes. This happens because the graph detects same landmark but with different position due to sensor errors. In other words, the position of the system is error from the wrong estimated distance from the landmarks.

The quantity numbers in **2.c.i** mean that not every observation vertex has an edge attaching to the landmark as the average number of landmark edges at each observation vertex is less than 1, and another evidence would be that the number of edges attached to each landmark is less than the total number of vehicle state vertex.

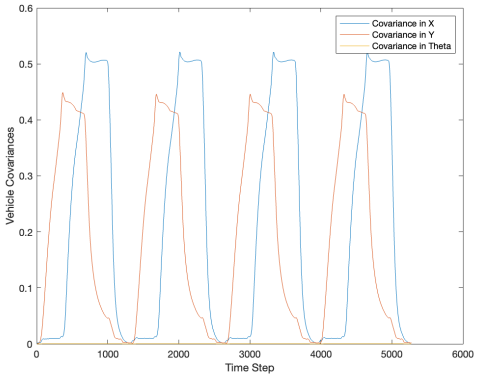

Figure 25: Vehicle Covariance

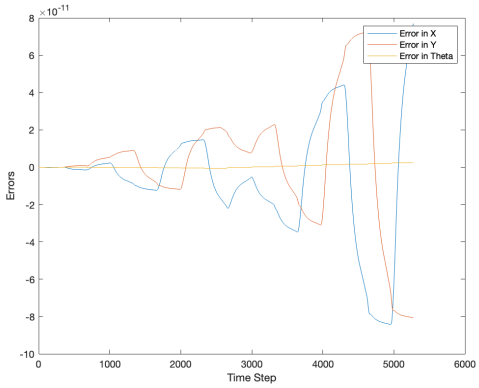

Figure 26: Errors

2.d Loop closure

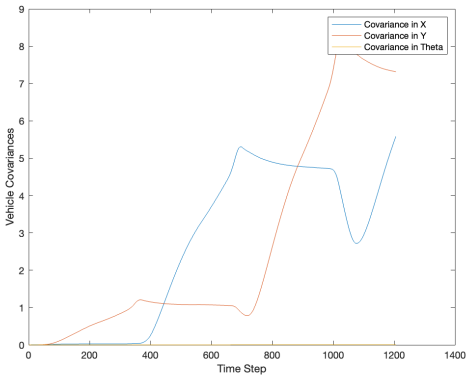

Figure 27: Vehicle Covariance before loop closure

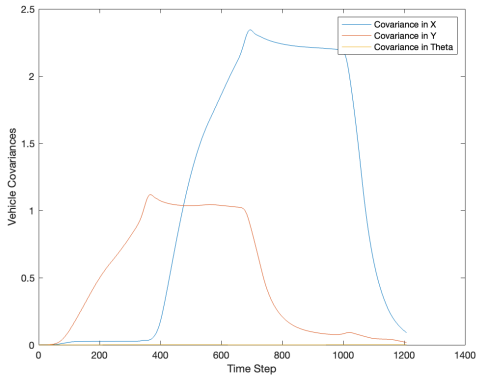

Figure 28: Vehicle Covariance after loop closure

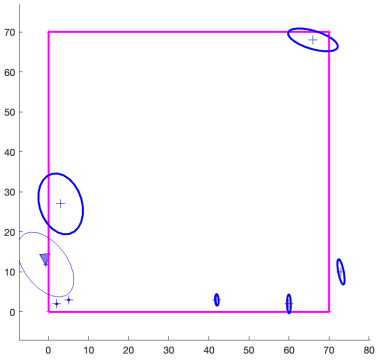

Figure 29: Simulation before loop closure (time step 1206)

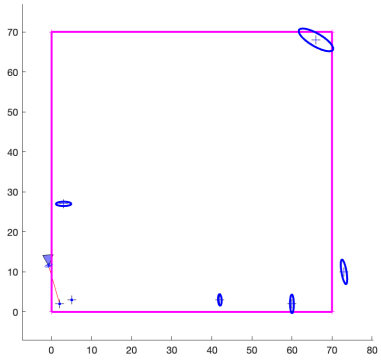

Figure 30: Simulation after loop closure (time step 1208)

As discussed in Section 2b(ii), loop closure is expected to reduce the covariance as the robot approaches its starting point. However, as shown in Figure 36 prior to the final loop closure event, the covariance in the x and y positions of the vehicle is higher than at the time step when the vehicle is furthest from the starting point. This phenomenon mirrors the behavior observed in the model discussed in Question 1, which lacks loop closure, leading to a continuous increase in uncertainty as the time step progresses. Conversely, after loop closure occurs, a significant reduction in covariance is evident, as demonstrated in the referenced figure. This reduction is attributed to the re-observation of landmarks that were observed at earlier time steps, thereby enhancing the accuracy of the robot's estimated position and reducing uncertainty.

3 Question 3

3.a Modify the factor graph

3.a.i

To improve the performance of a SLAM system, one may consider removing the vehicle prediction edges, denoted as $f_1, f_2, \ldots$ in the factor graph, as Figure 31. This modification would cause the SLAM system to operate similarly to a structure-from-motion system, relying primarily on sensor observations. The absence of motion prediction is advantageous when the predictions are unreliable or when the sensor data alone is sufficient to construct a robust map of the environment. Such a strategy is apt to succeed in feature-rich and structured settings where sensor accuracy is high and vehicle motion is comparatively predictable. Conversely, it might fail in feature-poor, dynamic, or highly noisy environments, where motion prediction significantly contributes to the accuracy of the SLAM system.

However, the removal of the vehicle prediction edges alongside the initial state edge f_0 can lead to a decline in performance. The initial state edge f_0 provides a necessary prior for the vehicle's starting position and orientation. Without this prior, the SLAM system lacks a global reference, rendering it unable to anchor and integrate the sensor measurements effectively. Consequently, the trajectory estimation and map construction may become inconsistent and unanchored in reality.

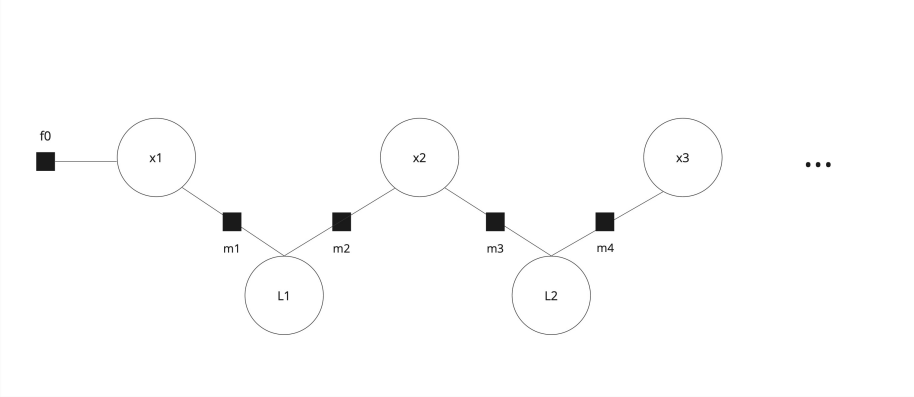

Figure 31: Factor Graph without vehicle state edges

3.a.ii

The following MATLAB function, `deleteVehiclePredictionEdges`, is designed to modify the factor graph of a SLAM system by removing specific edges related to vehicle kinematics. It checks visited edge is vehicle prediction edges, and remove it. In the case keepFirstPredictionEdge is true, first edge which is inital prior edge is kept.

Listing 1: MATLAB code for removing vehicle prediction edges from a SLAM factor graph.

```
function deleteVehiclePredictionEdges(this)
    % Iterate through all edges in the graph
```

```
    allEdges = this.graph.edges;
    numbersOfEdges = length(allEdges);

    currentEdgeIndex = 0;
    for e = 1:numbersOfEdges
        edge = allEdges{e};

        % Check if the edge is a vehicle kinematics prediction edge
        if class(edge) == "drivebot.graph.VehicleKinematicsEdge"
            currentEdgeIndex = currentEdgeIndex + 1;
            % Retain the first prediction edge if specified
            if (currentEdgeIndex == 1 && this.keepFirstPredictionEdge == true
                )
                disp("First Edge kept");
            else
                % Remove the vehicle prediction edge from the graph
                this.graph.removeEdge(edge);
            end
        end
    end
end
```

3.a.iii

We can find out when all the prediction edge including first one, failure occurs. The first prediction edge acts as a fundamental prior for the vehicle's initial state, represented as a unary edge with an embedded probability distribution. This initial edge is crucial as it provides a soft constraint indicating the likely starting position and orientation of the vehicle, essentially anchoring the entire estimation process. Without this anchor, the optimization and localization efforts would lack a clear direction, rendering them ineffective. This is because the initial constraint on the vehicle pose is indispensable for setting a reference point from which the system can begin its estimation and optimization processes.

Furthermore, prediction edges play a vital role in representing the expected motion of the vehicle from one pose to the next, with each edge embodying a probability distribution of this motion. The removal of these edges, particularly the initial one, would disrupt the system's ability to establish a coherent trajectory through time. These edges are not only pivotal for maintaining a flow of information but are also critical for the optimization process, where the minimization of the error function between pose vertices is essential. Without the foundational prior provided by the initial edge, applying Bayes' rule to solve for the posterior in the context of graphical models becomes unachievable, thus significantly compromising the system's ability to accurately predict and localize the vehicle's pose.

3.a.iv

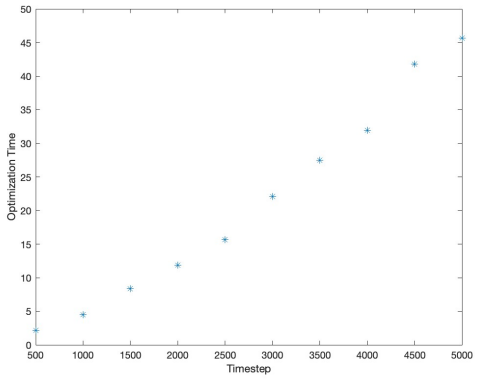

Figure 32: Optimization time removing edges except initial edge

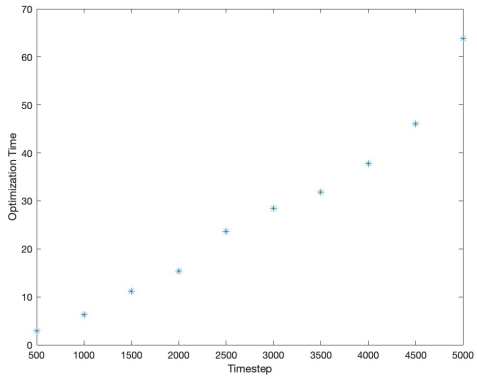

Figure 33: Optimization time without removing edge

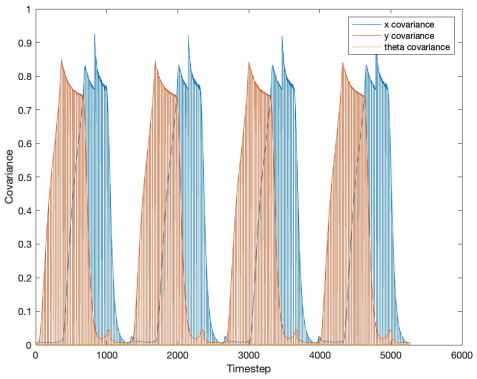

Figure 34: Vehicle Covariance removing edges except initial edge

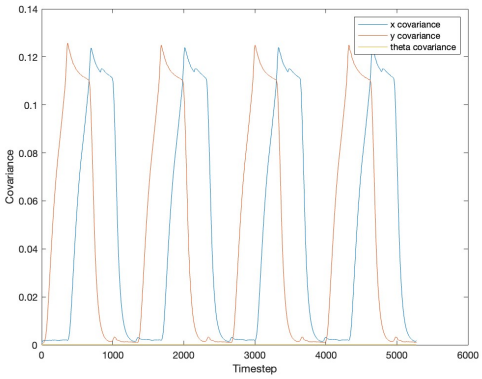

Figure 35: Vehicle Covariance without removing edge

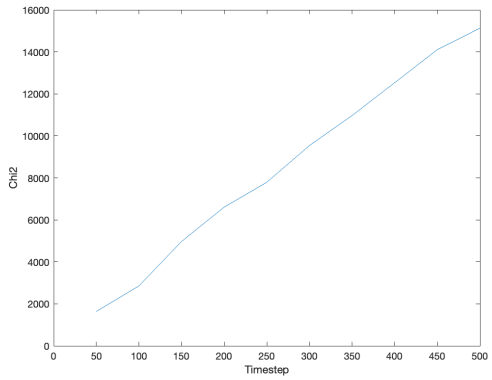

Figure 36: Chi2 Value removing edges without except edge

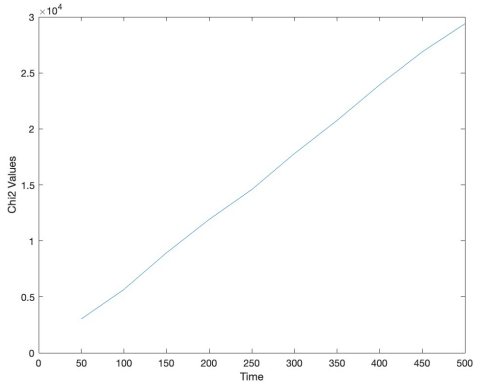

Figure 37: Chi2 Value without removing edge

Removing prediction edges except initial state, shows a significant decrease in optimization time, suggesting an improvement in computational efficiency. However, covariance and Chi2 values increased which gives result with less quality. The observed peaks in covariance correspond to increased uncertainty in the SLAM estimates, which can be attributed to the lack of kinematic constraints that typically aid in stabilizing the predictions between sensor measurements. The pronounced fluctuations in covariance indicate that the system is more sensitive to new sensor data and less smoothed by the vehicle's motion model, leading to abrupt adjustments in the estimated state.

The results of this comparison illuminate the impact of vehicle prediction edges on the balance between optimization speed and accuracy in graph-based SLAM systems. While removing these edges, apart from the initial one, can significantly speed up the optimization process, it does so by compromising the quality of the mapping and localization results. This compromise suggests that the standard approach, which retains all vehicle prediction edges, may be more suitable for applications where accuracy is paramount. Conversely, the modified approach could be preferred in scenarios where rapid processing is more critical than the ultimate precision of the SLAM solution.

3.b Graph Pruning

3.b.i

$$d(v_i) = \frac{1}{\pi} \sum_{\substack{k \in \{1,\ldots,n\} \\ \{i\}}} \|v_k - v_i\|^{-1}$$

Figure 38: Density equation [2]

To effectively apply graph pruning, it is critical to determine the importance of each vertex. For this purpose, we reference Figure 38 from [2], which illustrates the equation used to calculate the

scale-invariant density. It expresses the density $d(v_i)$ of a vertex v_i as the inverse of the average distance to all other vertices v_k in the graph, normalized by the factor $\frac{1}{\pi}$. Specifically, $\|v_k - v_i\|$ denotes the Euclidean distance between vertices v_k and v_i, and the summation is taken over all vertices k except for i itself. This formulation ensures that the density is scale-invariant, meaning it does not change if the entire graph is scaled up or down. The normalization by π is a conventional choice to simplify the integration over areas when working in continuous spaces.

This measure of density is independent of the data associated with any given vertex and facilitates an even distribution of vertices in the pruned graph. Utilizing the density values for each vertex, we can establish a threshold to decide which vertices should be removed, thereby optimizing the pruning process.

3.b.ii

Algorithm 1 Scale-Invariant Graph Pruning

Input: graph (V, E)
Output: pruned graph (V', E')
Initialize: $V' \leftarrow V, E' \leftarrow E$
Calculate scale-invariant density $d(v_i)$ for each $v_i \in V$ using:
$d(v_i) = \frac{1}{\pi} \sum_{\substack{k=1 \\ k \neq i}}^{N} \|v_k - v_i\|^{-1}$
/* $\|v_k - v_i\|$ *is the Euclidean distance between vertices* v_k *and* v_i */
Define density threshold $\hat{s}$
for $v_i \in V$ **do**
    if $d(v_i) > \hat{s}$ **and** v_i is not an initial state **then**
        Remove v_i's VehicleKinematicsEdges.
    end if
end for
return pruned graph (V', E');

The implementation of this scale-invariant density based graph pruning involves iterating all the vertex pair combinations, calculating the scale-invariant density, and deleting the edges based on the density threshold. A density of the vertex is calculated the accumulation of the inverse distance, and compared with a maximum threshold. The vehicle state edges on the vertex that has density higher than threshold is deleted. Specifically, key steps are as below, this implementation is done the the function "graphPruning()" of "DriveBotSLAMSystem".

1. Retrieve all vertices from the graph and define the density threshold;

2. For each state vertex, find each state vertex that is other than this state vertex, and keep the position values of both;

3. Use the position values to compute the Euclidean distance between each vertex pairs, and then apply the equation of figure 38 to calculate the density; This is done by accumulating density during the iteration and the accumulated density is divide by π once the loop iterated through all the state vertex that is other than this state vertex.

4. In the end, if the density is found above the predefined threshold, the corresponding edge of this vertex pair is decided to be removed, if it is a kinematics edge. The initial edge is protected for the SLAM model to be able to process.

3.b.iii

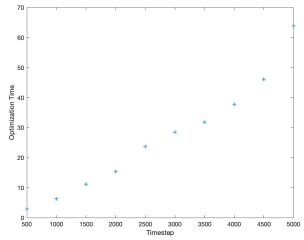

Figure 39: Optimization time without removing edge

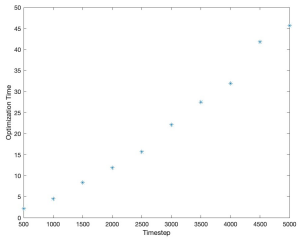

Figure 40: Optimization time removing edges except initial edge

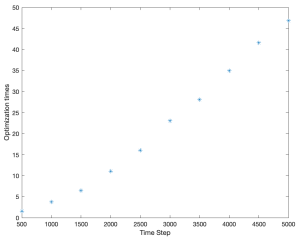

Figure 41: Optimization time with graph pruning

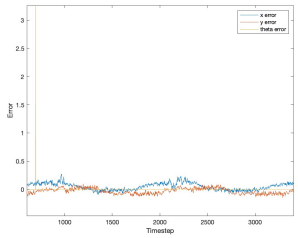

Figure 42: Error graph without removing edge

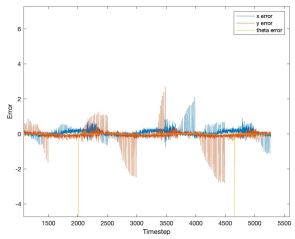

Figure 43: Error graph removing edges except initial edge

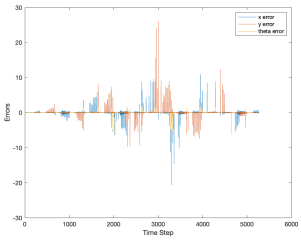

Figure 44: Error graph with graph pruning

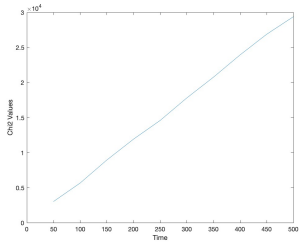

Figure 45: Chi2 Value without removing edge

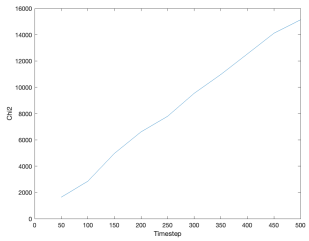

Figure 46: Chi2 Value removing edges except initial edge

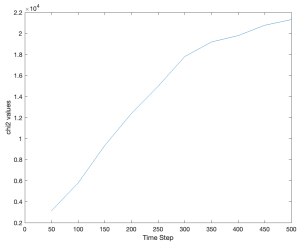

Figure 47: Chi2 Value with graph pruning

We implement the method as we mentioned in **3.b.ii**. After the experiment, we can see that the value of threshold has an effect on the performance. The lesser value of threshold will reduce optimization time and chi-square value, and increase errors. The results we show above (Figure 41, Figure 44, Figure 47) is created by using the threshold value of 100.

At the top row (Figure 39, Figure 40, Figure 41), it visualize the value of optimization time of the method without removing edge, the method removing all vehicle state edge except first edge and the graph pruning method respectively. It can be seen that the method we implemented uses time less than the original method and approximately close as removing all edges.

The errors of the graph pruning (Figure 44) are more fluctuated than the rest (Figure 42, Figure 43) due to removing some vehicle state edges and the remaining vehicle edges cause the noise.

The chi-square value of the graph pruning (Figure 47) is higher than the method with removing all edges (Figure 46) and lower than the method without removing edge (Figure 45). It can be concluded that our method can maintain well.

References

[1] Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: Part I. *IEEE Robotics & Automation Magazine*, 13(2), 99–110. https://doi.org/10.1109/mra.2006.1638022

[2] Kurz, G., Holoch, M., & Biber, P. (2021). Geometry-based Graph Pruning for Lifelong SLAM. *CoRR*. https://doi.org/10.48550/arXiv.2110.01286

[3] Magnabosco, M., & Breckon, T. P. (2013). Cross-spectral visual simultaneous localization and mapping (SLAM) with sensor handover. *Robotics and Autonomous Systems*, 61(2), 195–208. https://doi.org/10.1016/j.robot.2012.09.023

[4] Perera, S., Barnes, D., & Zelinsky, D. (2014). Exploration: Simultaneous Localization and Mapping (SLAM). In Ikeuchi, K. (Ed.), *Computer Vision*. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-31439-6_280