COMP0130: ROBOT VISION AND NAVIGATION

Coursework 02：

Graph-based Optimisation and SLAM

Jian Zhou

21082500

ucabj42@ucl.ac.uk

Xianjian Bai

21135608

ucabxb1@ucl.ac.uk

Hanpeng Li

22072479

ucabhl9@ucl.ac.uk

March 16, 2023

Q1

a)

Factor graph is a visualization of the factorization of joint probability, which makes the whole system more manageable and intuitive. By factorizing the joint probability, the dependencies of the event or system will be clarified clearly.

In this question, the factor graph is designed to predict the poses of the vehicle and then to update by GPS and compass. As is shown in figure 1, $x_0, x_1, x_2 \dots . x_n$ represent the poses of the vehicle, and the edges connected to them correspond to the kinematics function, which is the prediction part of system. Vertices named z_{GPS} and $z\_Compass$ are the latest GPS and compass event in each timestep. And H_{GPS} and $H_{compass}$ are the observation matrix of each measurement.

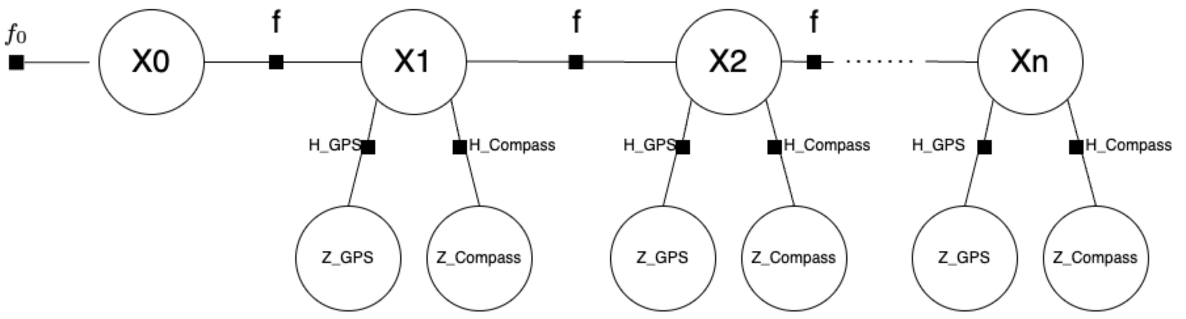


Figure 1: The factor graph that can just predict the robot pose and update states by GPS and compass.

In factor graph, the joint prediction of first step can be given by following Eq., where the first item is state transition density and the second one is prior.

$$f(x_1|u_1, x_0) = f(x_1|u_1, x_0)\, f(x_0) \tag{1}$$

Following the similar decomposition strategy, when add the second vehicle state vertex, the probability distribution can be expressed by next Eq., where U_2 is a set of input at timestamp 2,

$$f(x_{1:2}|U_2) = f(x_2|x_1, U_2)\, f(x_1|U_2) \tag{2}$$

According to the process model show below in Eq. [3], the current state only depends on the previous state and current control input.

$$x_{k+1} = x_k + \triangle T_k M(\psi_k)(u_k + v_k) \tag{3}$$

The Eq. can be simplified to Eq. [4], which is the function of prediction edge.

$$f(x_k|x_{k-1}U_k) = f(x_k|x_{k-1}, U_k) \tag{4}$$

And the whole distribution can be written as Eq. [5].

$$f(x_{1:k}|U_k) = f(x_0)\prod_{i=1}^{k} f(x_i|x_{i-1}, u_i) \tag{5}$$

The next stage is adding observation, which means we need to find the probability like Eq. [6].

$$f(x_{1:k}|\mathbb{I}_k) \ where\ \mathbb{I}_k = \{Z_{0:k}, U_{0:k}, x_0\} \tag{6}$$

Applying Bayes Rule, we can obtain the function of observation edge:

$$f(x_{1:k}|\mathbb{I}_k) = \frac{f(z_{1:k}|x_{1:k})f(x_{1:k}|U_k)}{f(z_{1:k}|U_k)} \tag{7}$$

The compass observation model can be expressed by:

$$z_k^C = [\psi_k] + [\triangle\,\psi_c] + w_k^C \tag{8}$$

The GPS observation can be expressed by:

$$z_k^G = \begin{bmatrix} x_k \\ y_k \end{bmatrix} + M(\psi_k)\begin{bmatrix} \triangle\,x_G \\ \triangle\,y_G \end{bmatrix} + w_k^G \tag{9}$$

In the observation model, the current observation only depends on the current state, thus we can get:

$$f(x_{0:k}|\mathbb{I}_k) \propto f(x_0)\prod_{i=1}^{k} f(x_i|x_{i-1}, u_i) \times \prod_{i=1}^{k} L(x_i; z_i) \tag{10}$$

At the end we find the expected state by maximum aposteriori estimation, so the effect of the unknown constants in Eq. [10] can be ignored.

b)

i)

The method $handlePredictionToTime$ in file $DriveBotSLAMSystem.m$ is firstly edited. And then the class $VehicleKinematicsEdge.m$ is implemented by completing the method $computeError$ and $linearizeOplus$. The logic of the codes are as follows:

1. Create the next vehicle vertex
2. Create a new prediction edge
3. Set original vertex, destination vertex, input, covariance of the input noise
4. Initialize the edge and add it to the graph
5. Put the edge in the chain of $processModelEdges$ of the system

As for the $computeError$ and $linearizeOplus$, the first one is to calculate the residual between the control input under prediction and the control input in real situation with the equation. The second one is to calculate the Jacobian matrices of its two vertices that are connected to it.

ii)

If we look at the error graph (figure2) alone, the x and y coordinates of the vehicle have similar errors throughout, hovering between plus and minus 0.6. The error in the theta angle is also very small.
If we look at the covariance graph (figure 3), although the covariance of theta angle is very small, we see that the x covariance rises steeply after about frame 350. Looking at this simulation schematic in conjunction with figure 4, we would be able to make a reasonable guess that the car is not able to correctly predict the position of the vehicle after it changes motion due to our prediction system's inability to cope with changes in velocity and acceleration after the turn. Therefore, this causes the difference between prediction and odometry to become larger, and thus causing the covariance to become larger. Also, the chi2 graph has a slope increase at around frame 350.

The residuals always equal to zero. It could be the simulator treat the measurement from odometry as the ground truth.

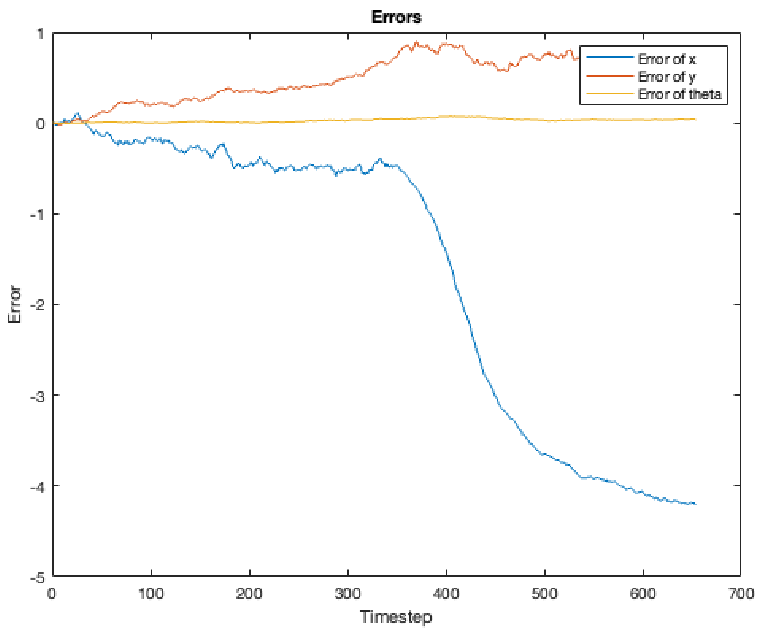


Figure 2: Errors of Q1

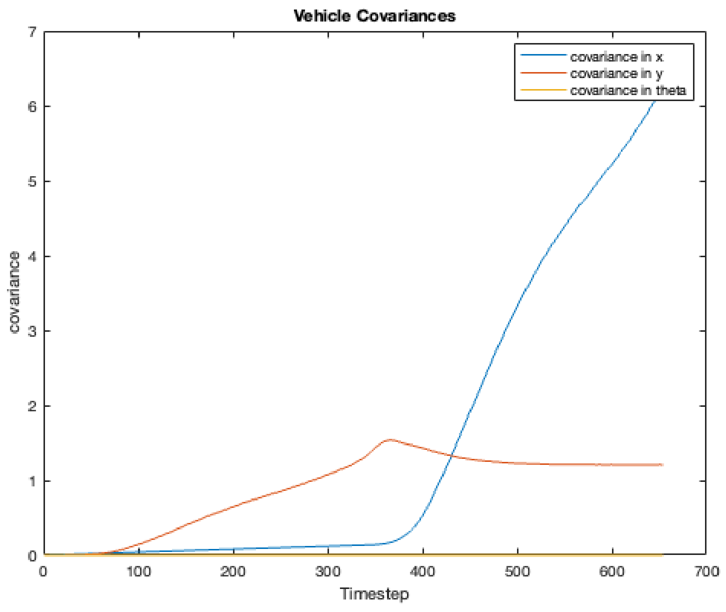


Figure 3: Vehicle covariances of Q1b

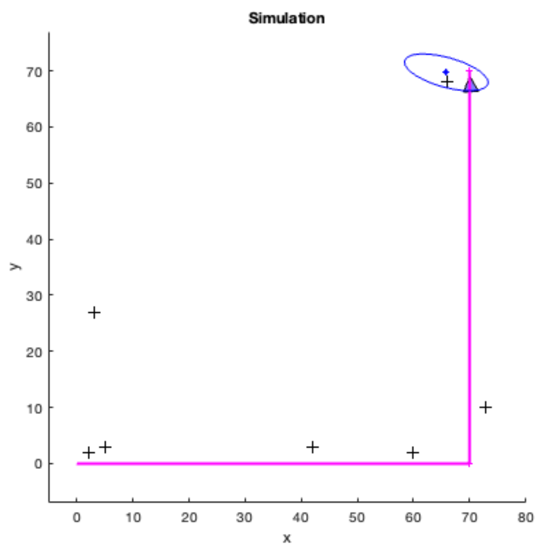


Figure 4: Simulation illustration of Q1b

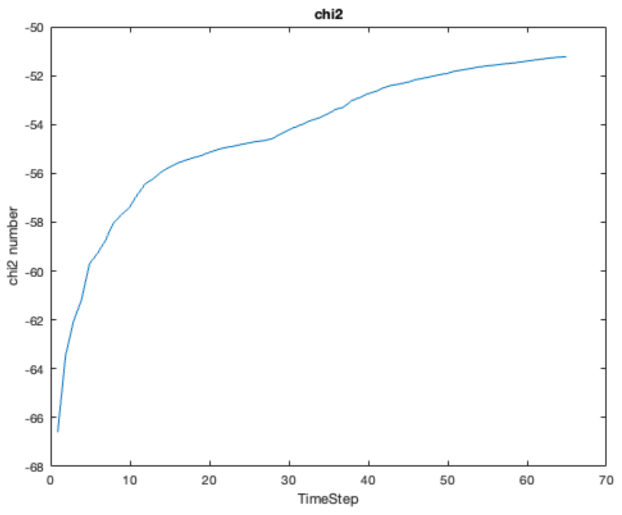


Figure 5: chi2 graph of Q1b

c)

i)

From Figure 8, we can see that in the last frame, the vehicle is very close to the ground truth, but in the ERROR, the difference between x, y and their ground truth can reach more than 80 (shown in figure 6), which means that the prediction value should be wrong. The histogram-like change of the angle suggests the possibility of breaking the threshold. And in the last frame, the residual reaches 6 digits (shown in figure 9), which further verifies the error of the predicted value. The sudden increase of Chi2 (figure 7) also verifies this point.

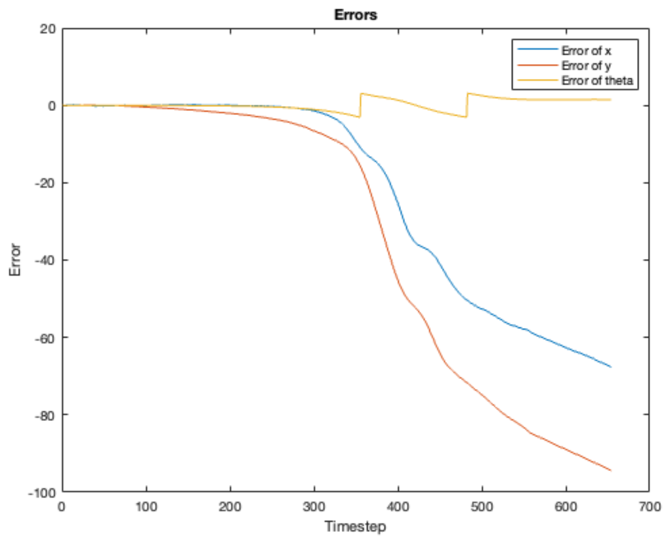


Figure 6: Error of Q1c-i

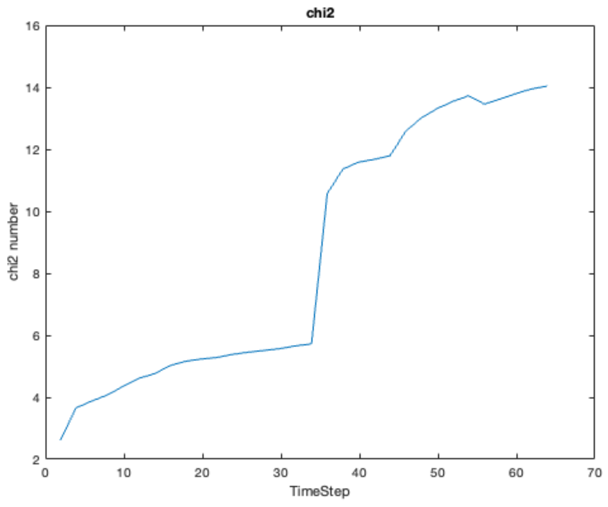


Figure 7: chi2 of Q1c-i

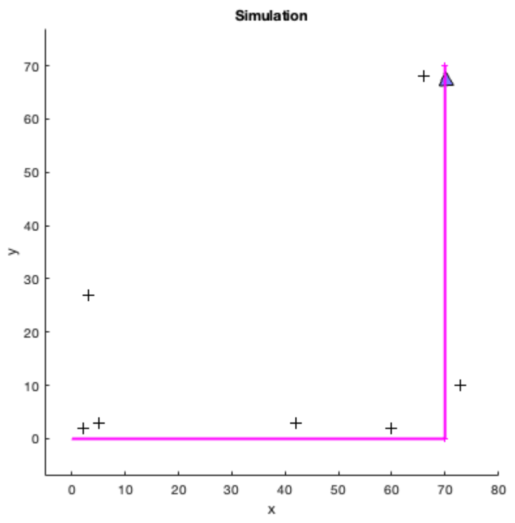
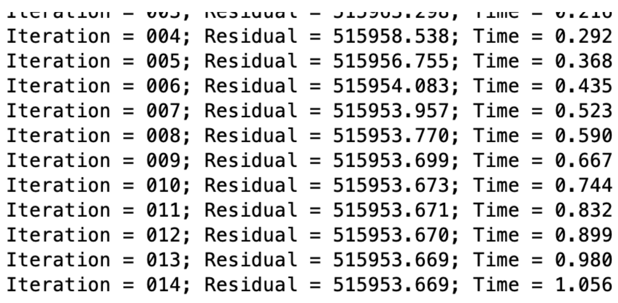


Figure 8: Simulation of Q1c-i



```
Iteration = 004; Residual = 515958.538; Time = 0.292
Iteration = 005; Residual = 515956.755; Time = 0.368
Iteration = 006; Residual = 515954.083; Time = 0.435
Iteration = 007; Residual = 515953.957; Time = 0.523
Iteration = 008; Residual = 515953.770; Time = 0.590
Iteration = 009; Residual = 515953.699; Time = 0.667
Iteration = 010; Residual = 515953.673; Time = 0.744
Iteration = 011; Residual = 515953.671; Time = 0.832
Iteration = 012; Residual = 515953.670; Time = 0.899
Iteration = 013; Residual = 515953.669; Time = 0.980
Iteration = 014; Residual = 515953.669; Time = 1.056
```

Figure 9: Large residual of Q1c-i

ii)

```
classdef CompassMeasurementEdge < g2o.core.BaseUnaryEdge

  % Q1c:
  % This implementation contains a bug. Identify the problem
  % and fix it as per the question.

  % Q1c answer:
  % the bug is failing to normalize the theta

  properties(Access = protected)

    compassAngularOffset;

  end

  methods(Access = public)

    function this = CompassMeasurementEdge(compassAngularOffset)
      this = this@g2o.core.BaseUnaryEdge(1);
      this.compassAngularOffset = compassAngularOffset;
    end

    function computeError(this)
      x = this.edgeVertices{1}.estimate(); % state
      this.errorZ = x(3) + this.compassAngularOffset - this.z; %calculate the error
      % Q1c start
      this.errorZ = g2o.stuff.normalize_theta(this.errorZ); % put it into the range of [-pi,pi]
      % Q1c end
    end

    function linearizeOplus(this)
      this.J{1} = [0 0 1]; % the compass only measure the angle of the vehicle
    end
  end
end
```

In the code above, we edited the function *computeError*, so that it will normalize the theta of the vehicle into the range of [-pi, pi]. As a result, the system can now compute the right error and the results are shown in figure 10-11. You can find that the error is in the range of [-1.2,0.2] and the chi2 graph is roughly log like, which indicate a good tracking.

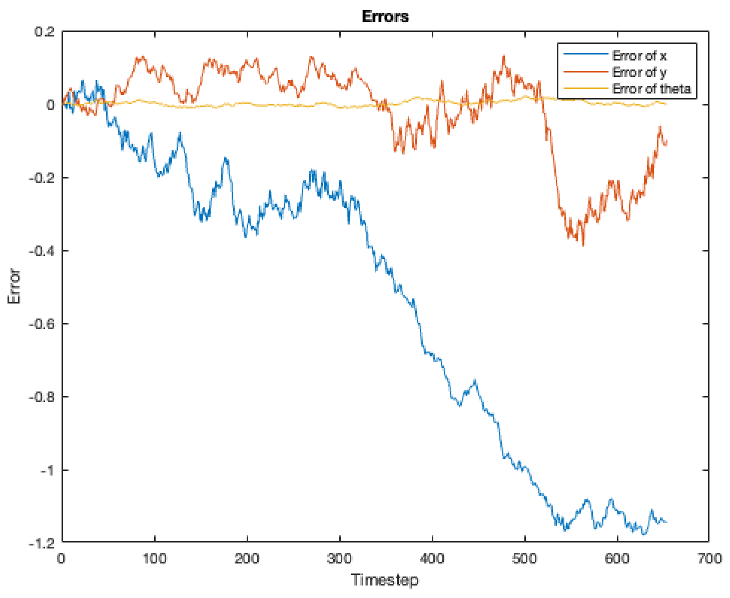


Figure 10: Error for Q1c-ii

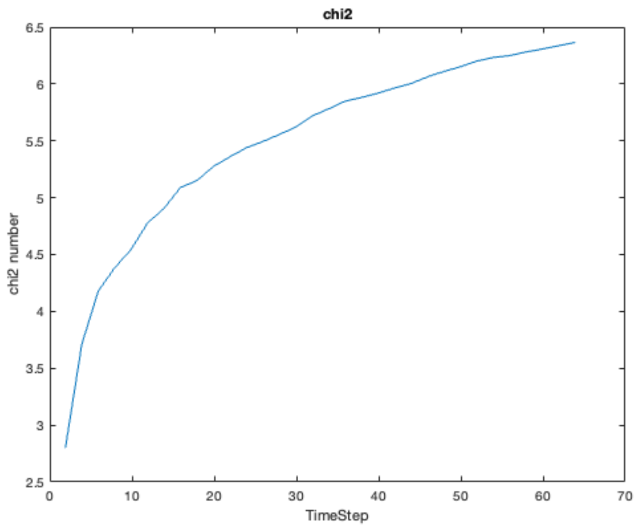


Figure 11: Chi2 for Q1c-ii

d)

i)

Method $handleGPSObservationEvent$ in file $DriveBotSLAMSystem.m$ has been edited. The logic of it is as follows:

1. Create a GPS edge
2. Set the measurement data and its corresponding covariance
3. Add it to the graph

Similar to prediction part, methods $computeError$ and $linearizeOplus$ are implemented. The error of GPS data is quite straightforward, which is just to subtract the measured x and y from the first two terms of system state, which share the same meaning.

ii)

In the graph of Error (figure 13), the error of x and y hovers between -12.5 and 12.5. the curve of covariance oscillates up and down at 0.03.

From the graph we can find two problems, the first one is that the wandering of error is too large for a total track of 80*80. And its periodic wandering makes one guess that it is the reason for the inaccuracy of the GPS.
The second problem is that the covariance of the GPS at the end is too large, meaning that it is inherently inaccurate. It could be the effect of outers.

From the changes in the simulation illustration graph, we can observe that the vehicle makes a total of 4 turns around a total of 15 turns. This corresponds to the 15 peaks of covariance. Therefore, we can speculate that the covariance of the vehicle becomes larger each time it goes through a turn. And the covariance of x and y become larger at different times with the direction of the bend, respectively. Since GPS has a correction effect on the vehicle's position, the covariance will reduce the uncertainty of the vehicle when moving in a straight line.

After investigating the code of the simulation, we found that it is the GPS offset that makes the vehicle has no idea of which of the observation and prediction to believe, as the observation is too inaccurate due to the offset. An error result with no GPS offset is shown in figure (14)

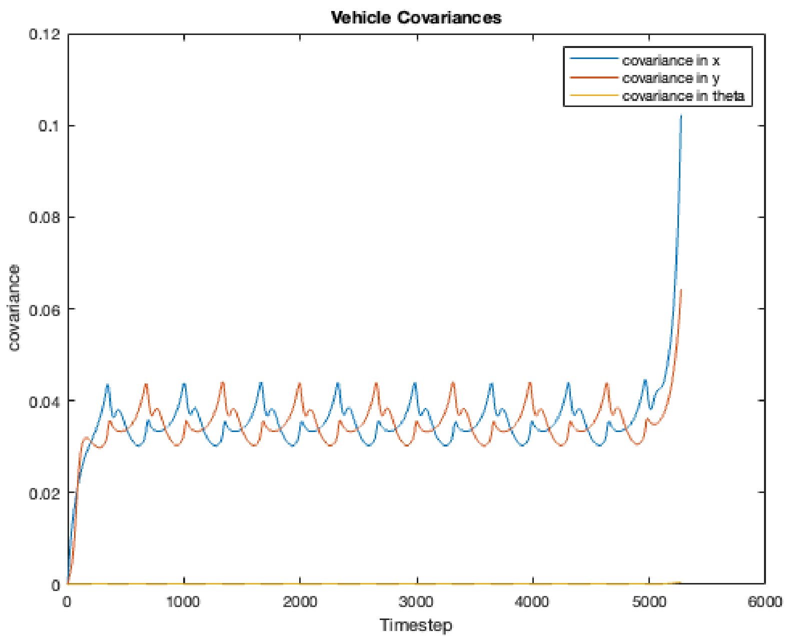


Figure 12: Covariance for Q1d

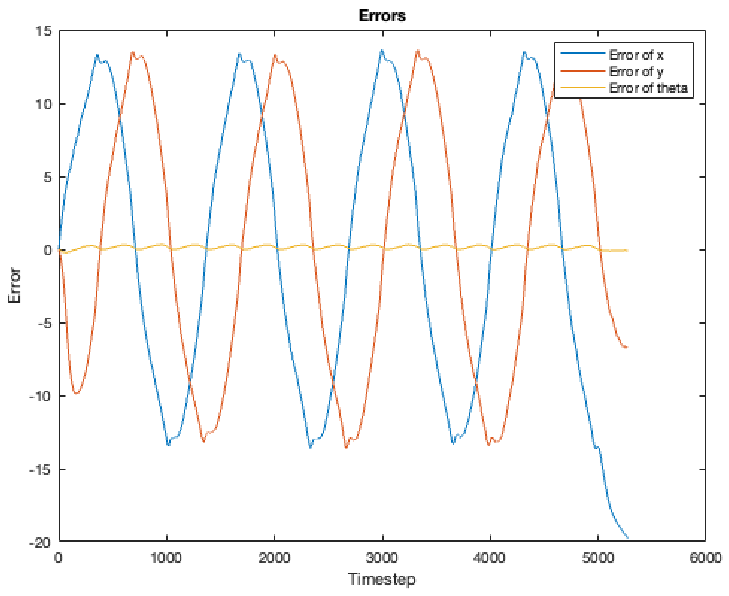


Figure 13: Error for Q1d

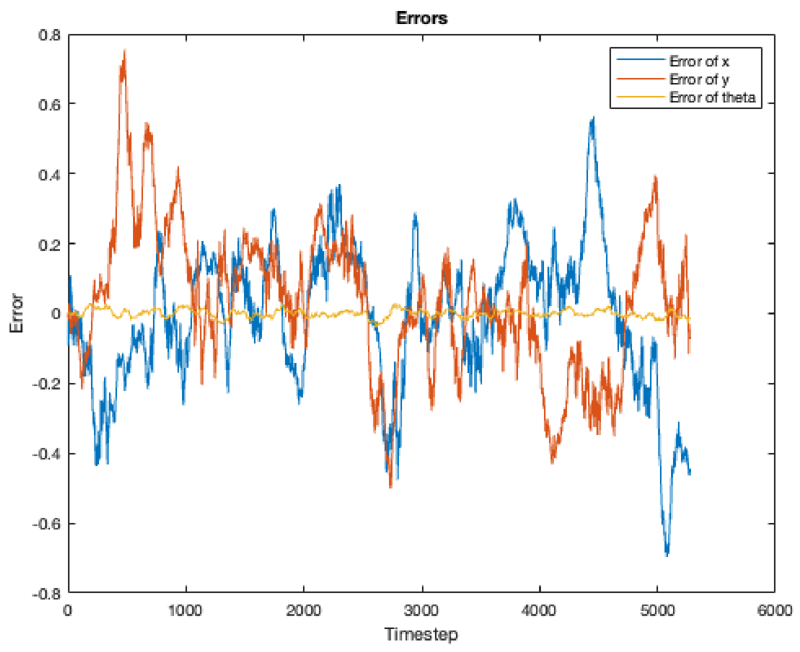


Figure 14: Error for Q1d with no GPS offset

e)

i)

As can be seen from Figure 17, the presence of an unreasonable GPS offset produces a huge bias in the system's position of the vehicle. Combined with Figure 15 and 16 we will find that this will be greatly distorted. Meanwhile, analyzing a extremely inaccurate result is meaningless. So, for better researching, the offset of the GPS will be removed in the analysis. And the result is shown in figure 18 and figure 19.

In most cases, the program does not take much time to optimize to the appropriate state, but from the step time (Figure 18) plot we find that there are about 30 out of 300 points that take a relatively large amount of time to compute. And as we can see from chi2 (Figure 19), those points correspond to the points where the chi2 image leaps significantly. We know that chi2 is equal to $\sum \frac{(observed-expected)^2}{expected}$. As the expected value is constant, the steep increase of chi2 should be a result of the sharp jump of observed data. We can speculate that the wired increase of chi2 and long time in optimization are caused by GPS outers. There is also possible that the jump is caused by missing correction of theta.

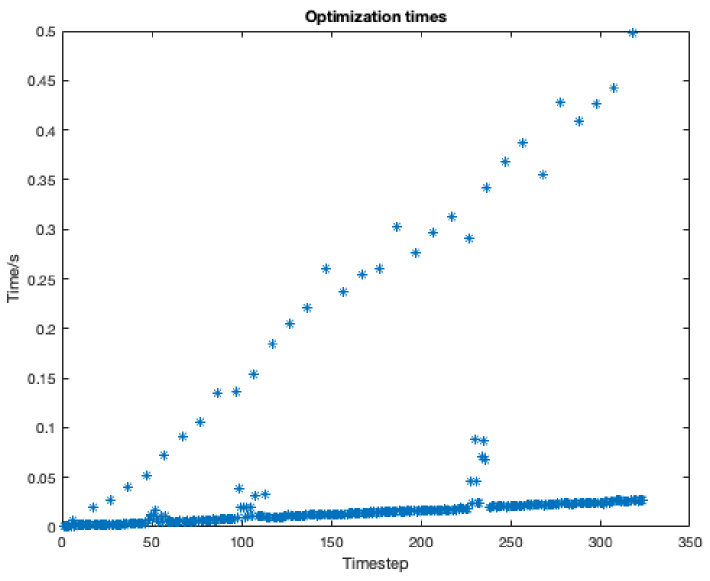


Figure 15: Optimization time for Q1e with GPS offset

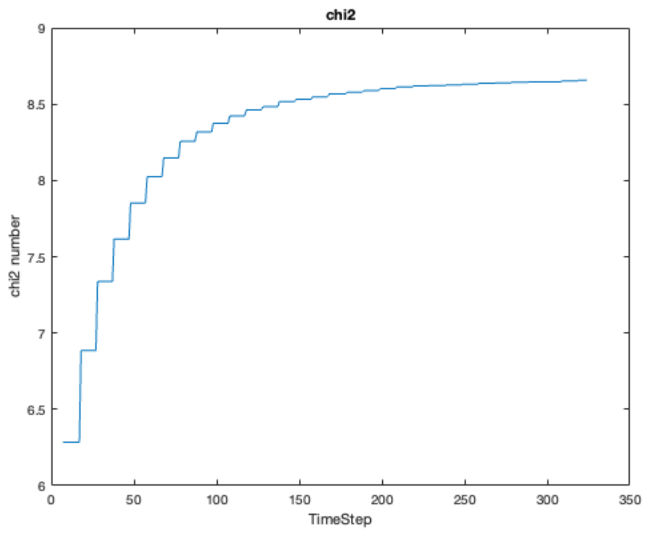


Figure 16: Chi2 for Q1e with GPS offset

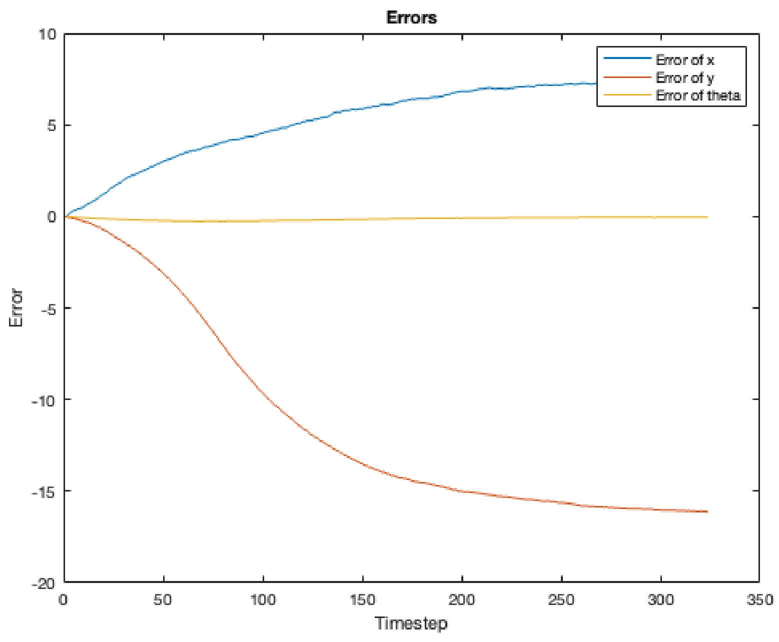


Figure 17: Error for Q1e with GPS offset

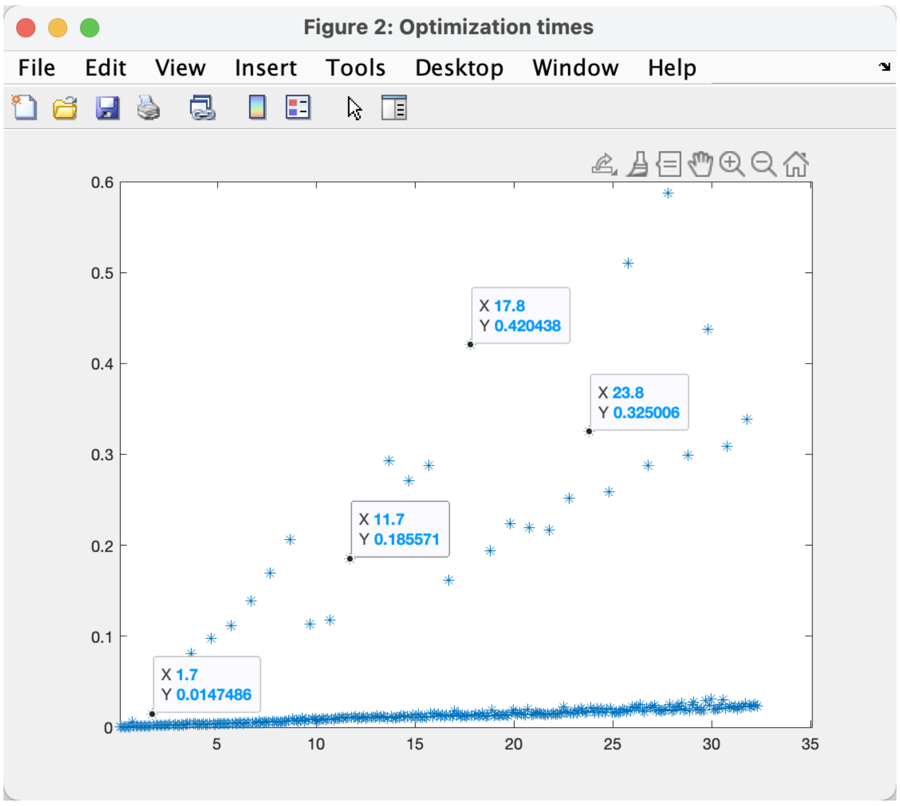


Figure 18: Optimization time without GPS offset

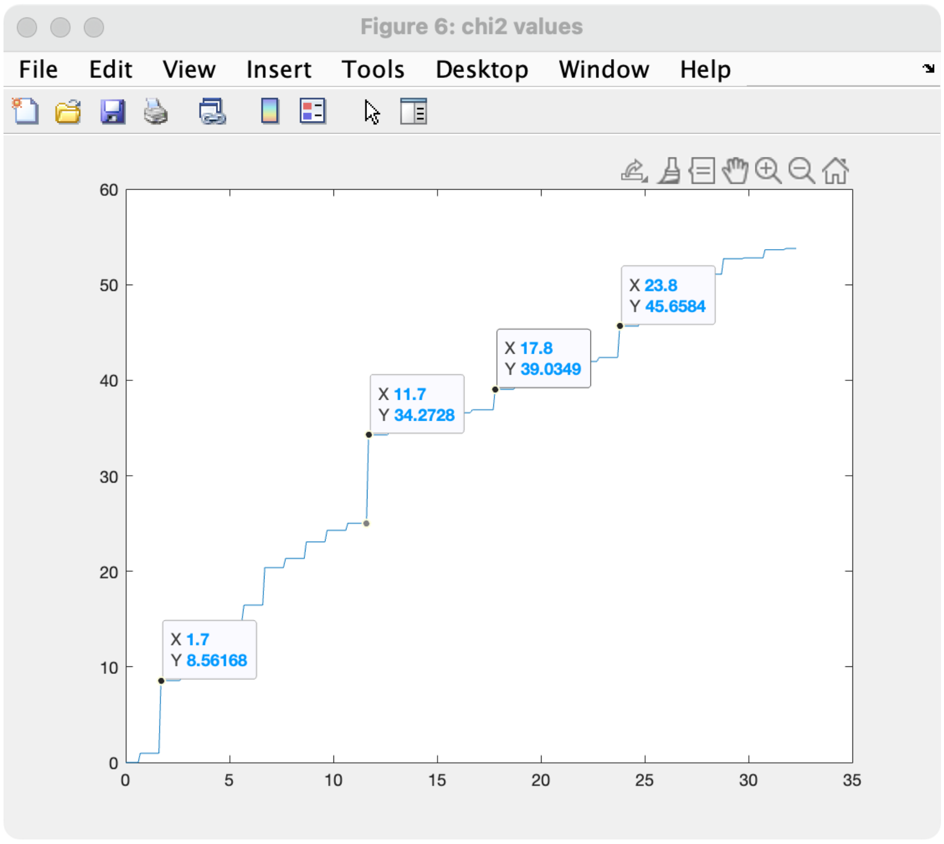


Figure 19: Chi2 figure without GPS offset

ii)

Chi2 images became smoother than before, indicating that the observed data and expected data are more similar. This may be due to the fact that the updated state theta makes the prediction more accurate and therefore reduces the difference between prediction and observed data. It is also possible that the previous outer was simply because the theta was not corrected.

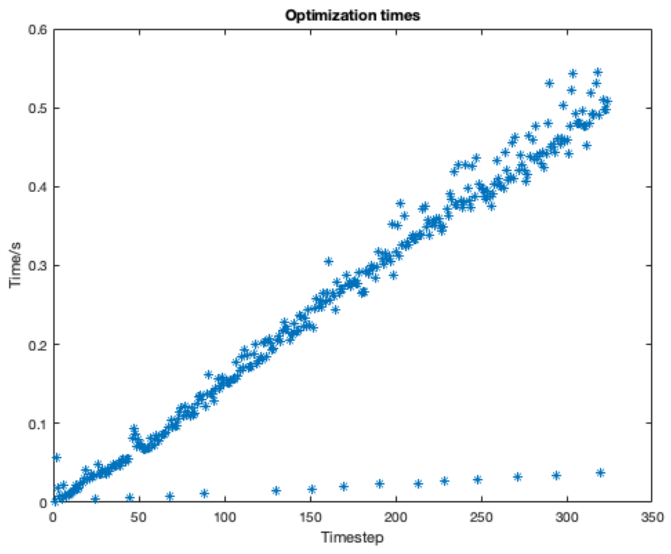


Figure 20: Optimization time with compass

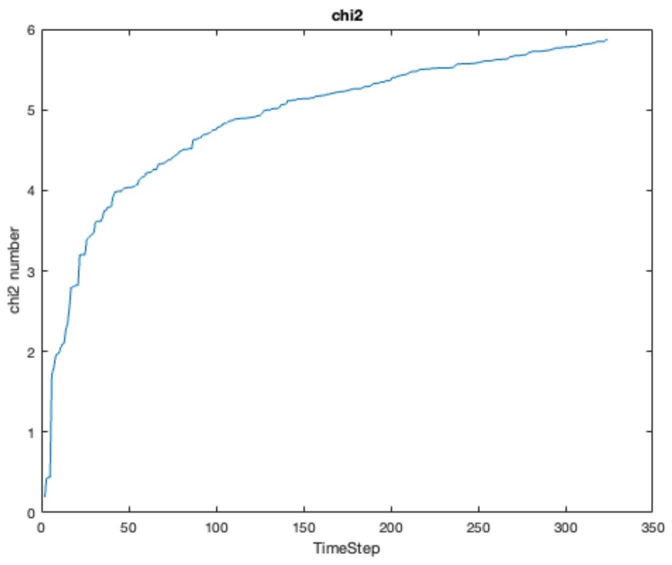


Figure 21: Chi2 figure with compass

Q2

a)

At this stage, the observation model of landmarks should be added. When introduce the landmarks, since the SLAM observation is a function of the landmark position and vehicle state, a new vertex should be used to store the landmark. When a landmark first observed, we need to initialize it according to the inverse observation model:

$$\begin{cases} x^i = x_k + r_k^i \cos(\beta_k^i) \\ y^i = y_k + r_k^i \sin(\beta_k^i) \end{cases} \tag{11}$$

At each timestamp, if the platform observes an existing landmark, an edge will connect the state at that moment to the corresponding landmark vertex, and the factor graph will be update. The landmark observation model shows the edge between vehicle vertex and landmark vertex.

$$z_k^L = \begin{bmatrix} r_k^i \\ \beta_k^i \end{bmatrix} + w_k^L \; where \begin{cases} r_k^i = \sqrt{(x^i - x_k)^{\,2} + (y^i - y_k)^{\,2}} \\ \beta_k^i = tan^{-1}\left(\dfrac{y^i - y_k}{x^i - x_k}\right) - \phi_k \end{cases} \tag{12}$$

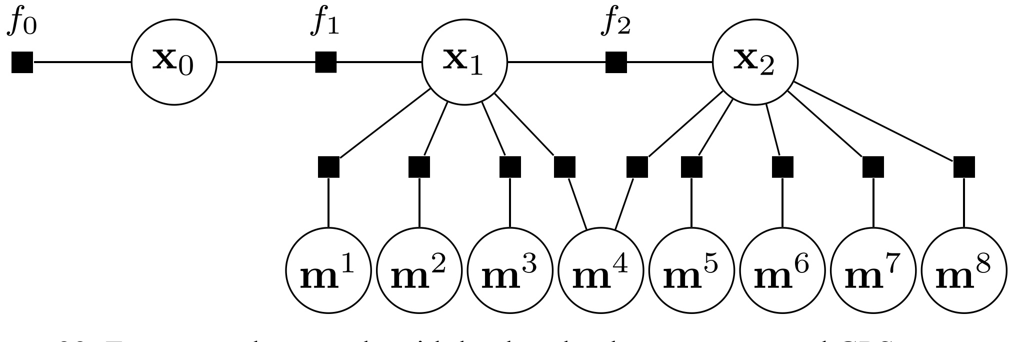


Figure 22: Factor graph example with landmark when compass and GPS are unused.

b)

i)

First of all, the method *handleLandmarkObservationEvent* are edited. In each timestep, the simulator will generate a landmark event where all landmarks are stored in *event. landmarks*. So, for each new landmark measurement, check it is recorded or not. If it is recorded before, then extend an edge from the landmark to the new state. If it is a new landmark, we should create a vertex to store it and connect it to the current vertex.

We also implemented $computeError$ and $linearizeOplus$ in class $LandmarkRangeBearingEdge.m$, which defines the behavior of the edge connected from landmark to system state. The implementation based on the analysis in Q2a.

ii)

As can be seen in Figure 23, since in this system we didn't turn on any sensor that can calibrate the position of the vehicle, the uncertainty of the vehicle accumulates over time. Therefore, we will find that the uncertainty of both x and y increases in the first half of the time. Later, after loop closure, the vehicle uncertainty converges rapidly. In addition to this, there is a delay in the increasing decreasing trend of the x and y uncertainty images. This comes from the fact that the vehicle travels first along the fixed y and then along the fixed x.

From Fig. 24 and Fig. 25, we can find that there are two areas where the optimization time is approximately equal to 0 and the chi2 images are flat. This is because in this interval, the vehicle does not encounter any landmark, so the time to be computed decreases abruptly and chi2 does not accumulate

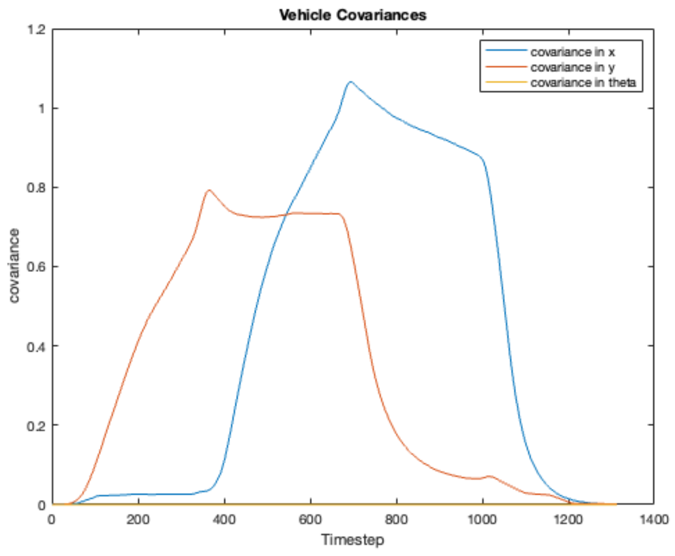


Figure 23: Vehicle covariances for Q2b

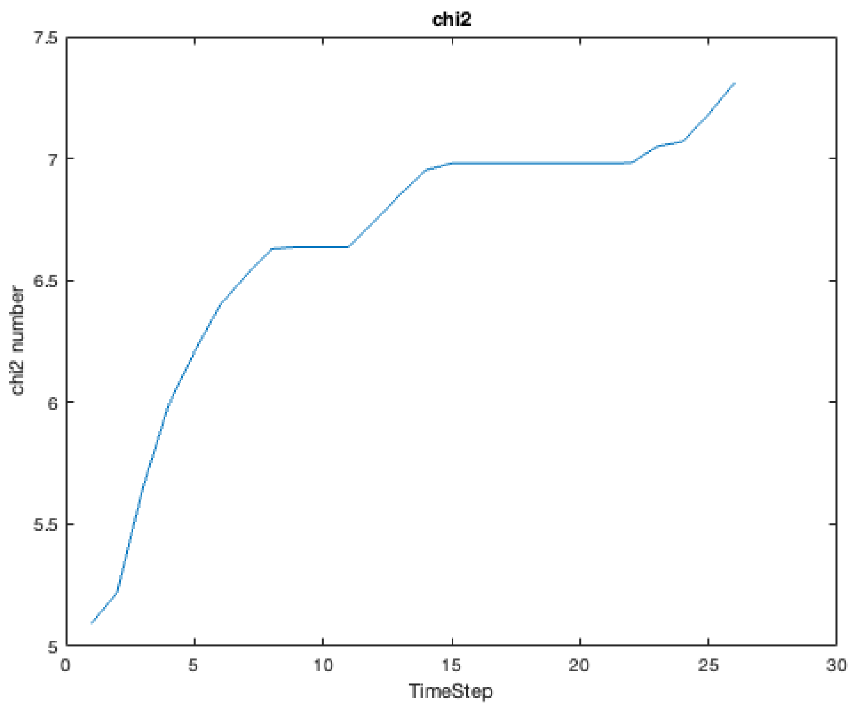


Figure 24: Chi2 figure for Q2b

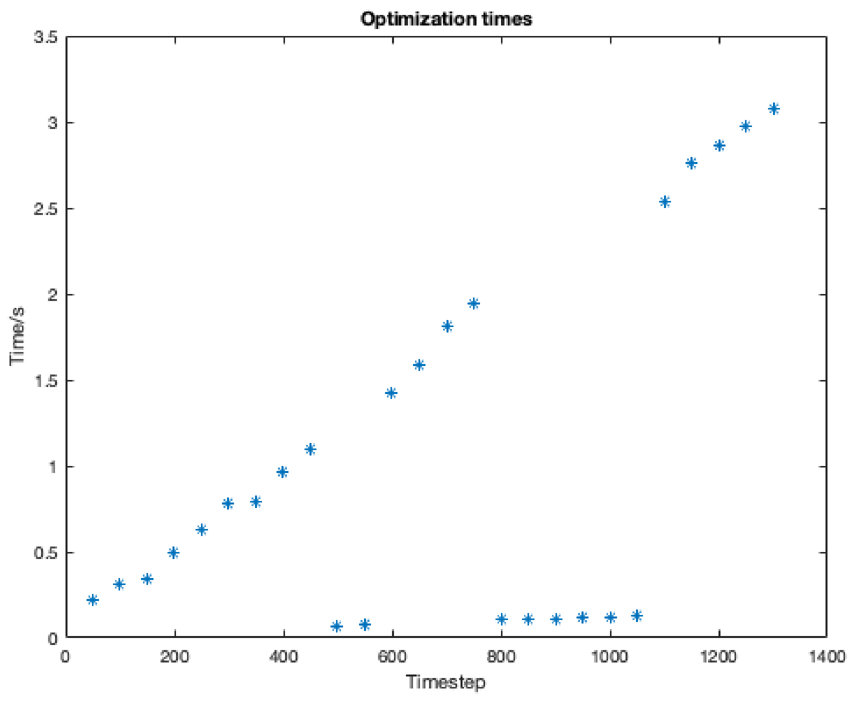


Figure 25: time consumption for Q2b

c)

i)

As we set the vehicle poses as the system state of the graphical model, the number of poses can be found by measure the length of $vehicleStateHistory$. The number of landmark can be calculated by $the\ number\ of\ vertices$ and $the\ number\ of\ poses$, because there are only two kind of vertices in the graph and we have already known the total number and the number of poses.

The average number of observations at each timestep can be obtained by $\frac{numEdges-numPoses}{numPoses}$. because each pose has a prediction edge before him, the number of poses is the same with the number of prediction edges. Then using $numEdges - numPoses$ we can get the number of total measurement edges. Then divide it by the number of poses, we can get the average number of observations at each timestep.

The average number of observations received by each landmark can be obtained by total number of observations divided by the number of total poses.

ii)

Table 1: Results for Q2c-ii			
Vehicle poses	5273	Observation per timestep	0.61
Landmarks	7	Observation per landmark	461.71

Vehicle poses and landmarks can only tell us the information themselves.
Observation per timestep implies the sparsity of landmarks
Observation per landmark implies the how long the vehicle stays around landmark in average if we know the interval between timestep.

d)

Figure 26 and Figure 27 show the simulations of the vehicle before and after the loop closure, respectively. The uncertainty of the vehicle and each landmark is high during the just before time, but the uncertainty of the vehicle and each landmark converges rapidly when the vehicle meets a landmark that has been encountered before. This can also be seen in Figure 28 and Figure 29.

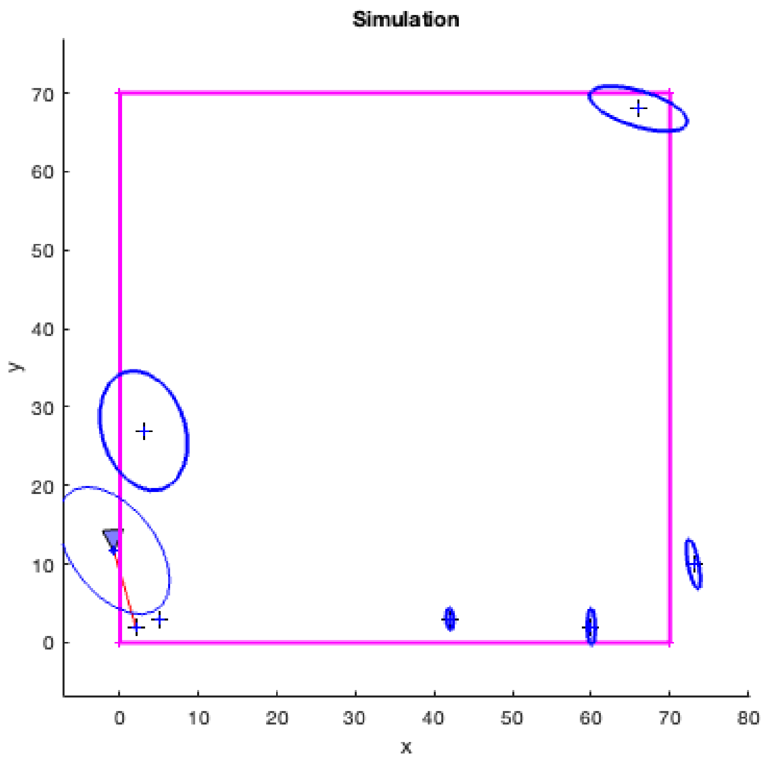


Figure 26: Simulation just before loop closure

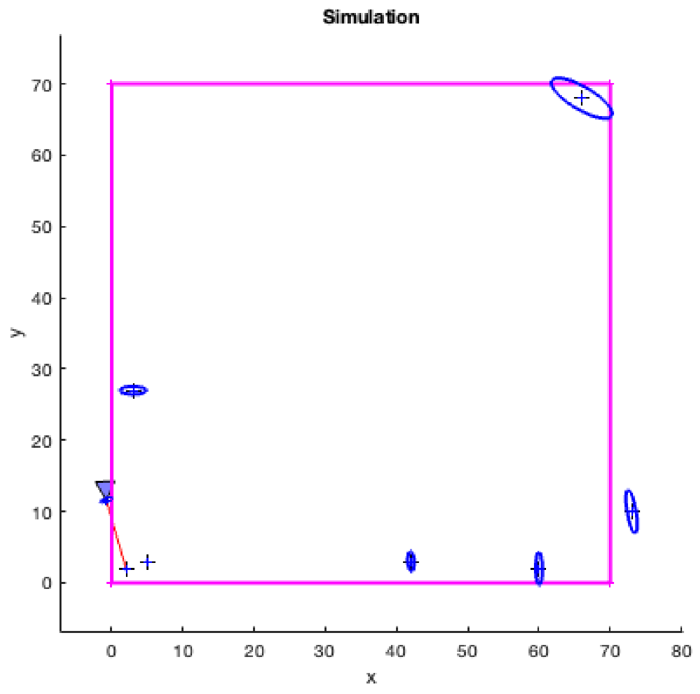


Figure 27: Simulation just after loop closure

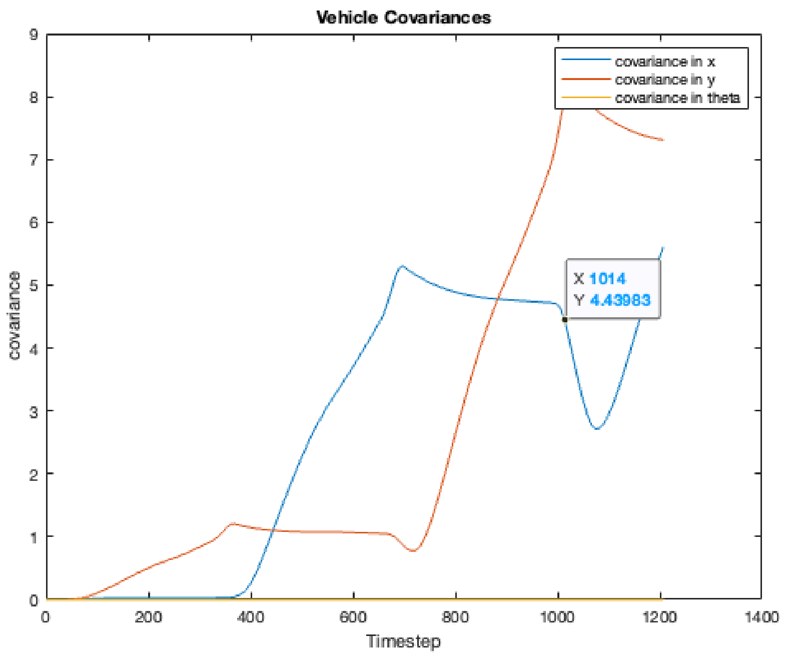


Figure 28: Covariance just before loop closure

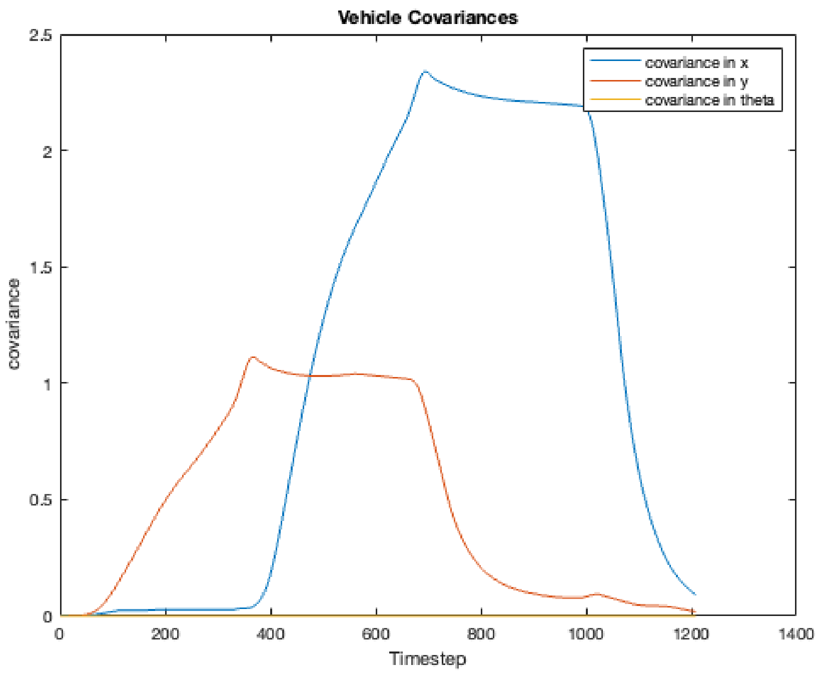


Figure 29: Covariance just after loop closure

e)

Origin at (0, 0) because the observation matrix of laser is (r, phi). When we calculate the global coordinates of landmark, we use the observation matrix (r, phi) which is the position of a landmark with respect to the cart, and then we use the coordinates of the cart itself (x, y) to get the coordinates of the landmark with respect to the global map. At this point we can get r_map very precisely by triangulation. phi is not necessarily so.
The problem is not caused by the covariance of the vehicle themselves in the first place. The accumulated phi of the vehicle is transmitted to them, but loop closure reduces the covariance of the vehicle to a negligible level.

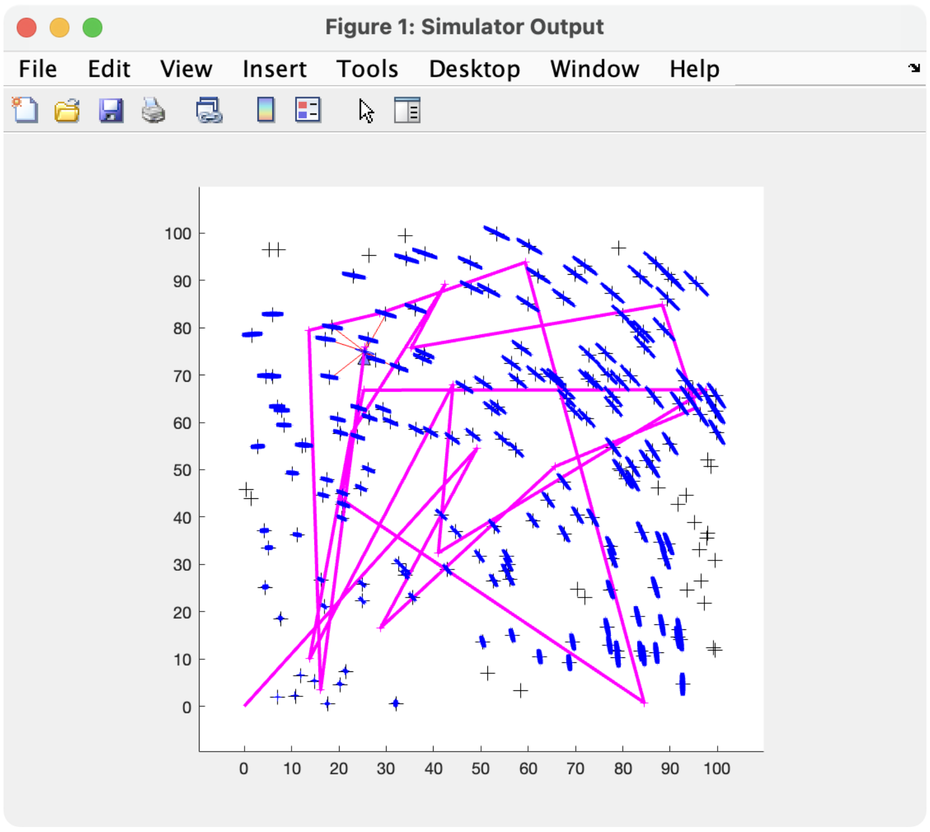


Figure 30： Simulation for q2e

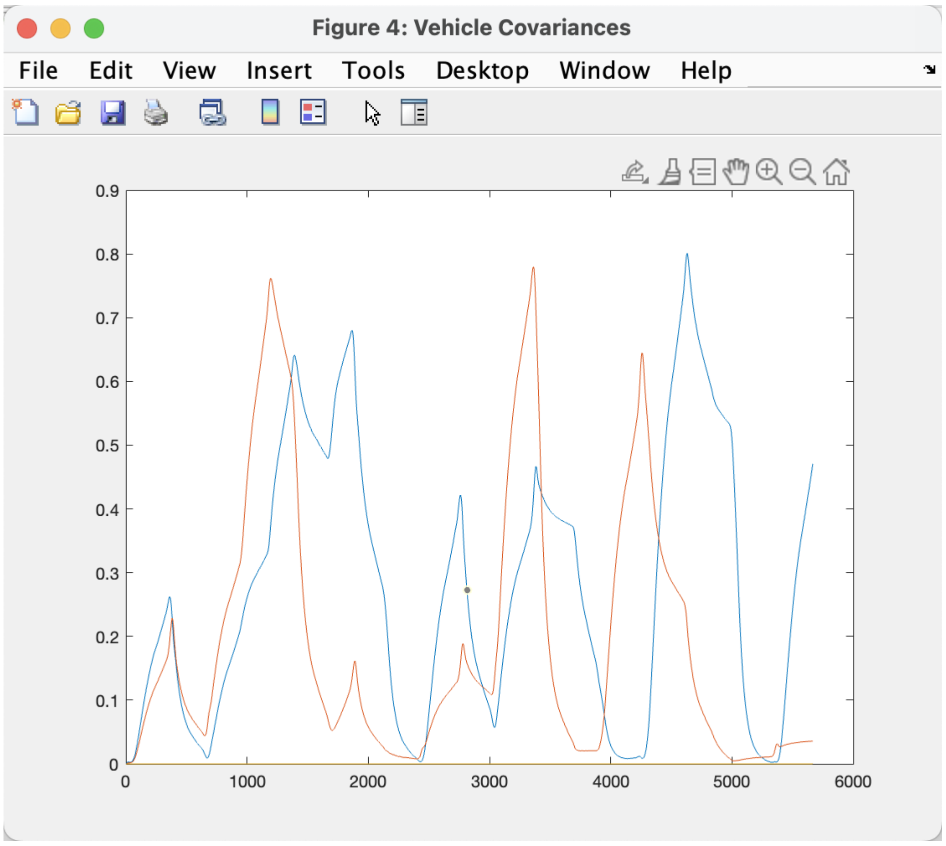


Figure 31: Vehicle Covariances for q2e

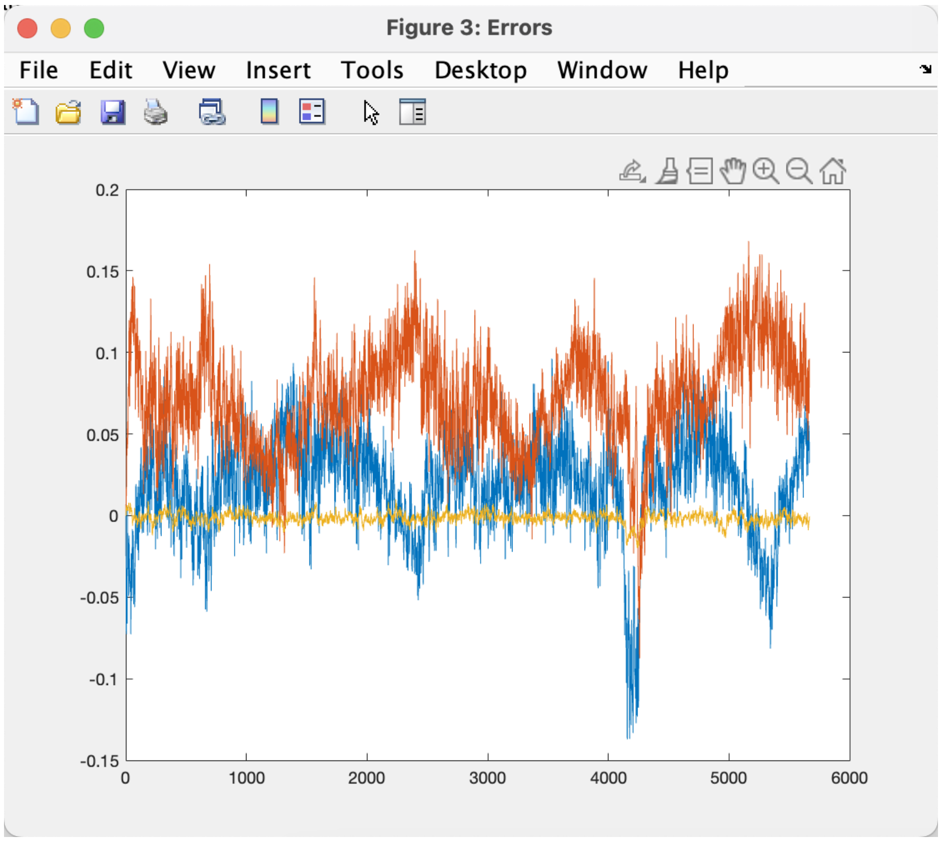


Figure 32: Error for q2e

Q3

a)

i)

Why do you think this might be a sensible strategy to take?

Discarding all predictions means giving up on correcting odometer data by prediction. The downside of this is making the states become more inaccurate as the system will fully believe the observations. However, we have a laser to look at the landmark as an addition to locating. As long as the loop is closed, then the state error can be corrected by the closure of the landmark without causing an accumulation of errors. This is the reason why it works. And it is beneficial after there is no connection between states, because the incorrectness of a single state will no longer affect other vertices.

When is it likely to be successful and unsuccessful?

When landmark is few and sparse, the most unsuccessful situation is when the loop closure cannot be formed. In that case, the cumulative uncertainty of the vehicle location increases with run time and eventually collapse the system's reliability.
However, if we are confident on that the vehicle will revisit the previous landmarks, this method deserve trying.

ii)

The only method that was edited is $deleteVehiclePredictionEdges$ in $DriveBotSLAMS.m$. the logic of it is as follows:

1. Iterate all edges of the graph
2. Judge if it is of the class "$drivebot.VehiclekinematicsEdge$"
3. If so judge if it is the first Edge
4. Basing on the configuration decide whether to delete each edge

iii)

The case when we remove all prediction edges was failed.

Commonly, if we want to delete all prediction edges of a graph, we will loss all the information of that a prediction will give us and relying only on measurement is risky, because it will be highly affected by measurement and eventually leads to an unreliable result.

However, in our coursework case, this program's failure is caused by unable to initialize the first state. In this situation, the initial state of the system can be any numbers. When we were calculating optimization, the unstable state has high probability to generate a complex number, which will break the simulator.

iv)

Both of them have a linear increasing optimization time and a similar increasing chi2 graph, but the removed one has better efficiency as it used less time.

However, the covariance of the removed one jumps up and down between the covariance curve of ordinary graph and zero. This is because not every time step has a measurement. If we abandon all prior information from previous system state, then for the timestep that don't have measurement event. The covariance of the system state cannot be updated, therefore the program set it to default covariance, which is zero. But actually, the uncertainty of the states that don't have measurement cannot be determined easily as we don't have any information in that specific timestep.

I don't think it is successful. First of all, it loses too much information, which make it completely believe the measurement data. If there exist a "bad data point", the mapping and navigation performance of the system will be affected seriously by that point. Secondly, the computation cost and memory storage of the method are still linearly accumulating during the processing. Furthermore, the steadily climbing computational cost will rapidly consume the memory and CPU ability that were saved by this method.

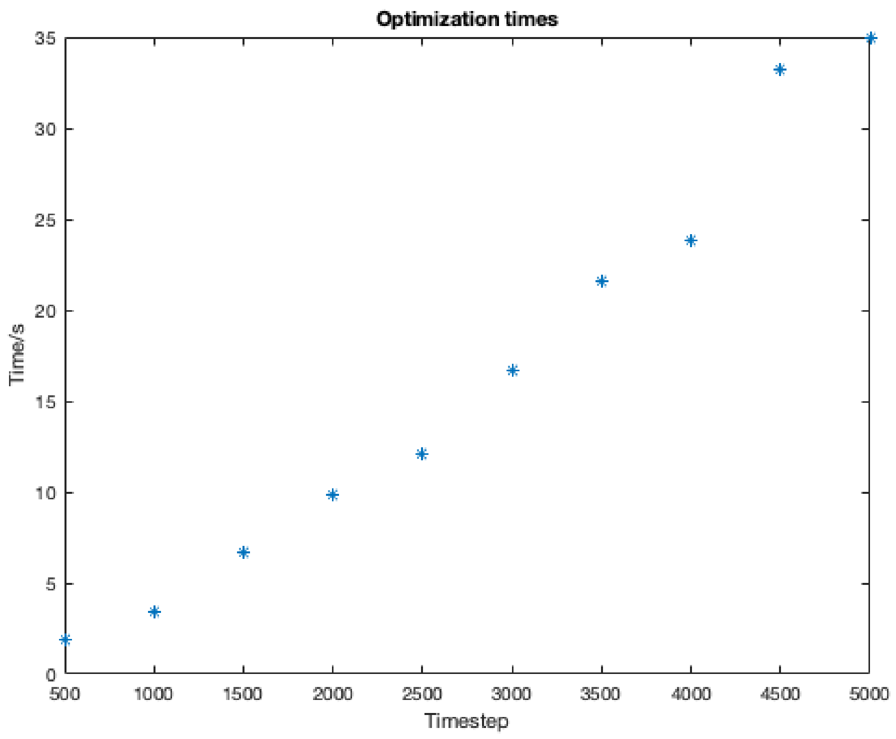


Figure33: Optimization for q3a

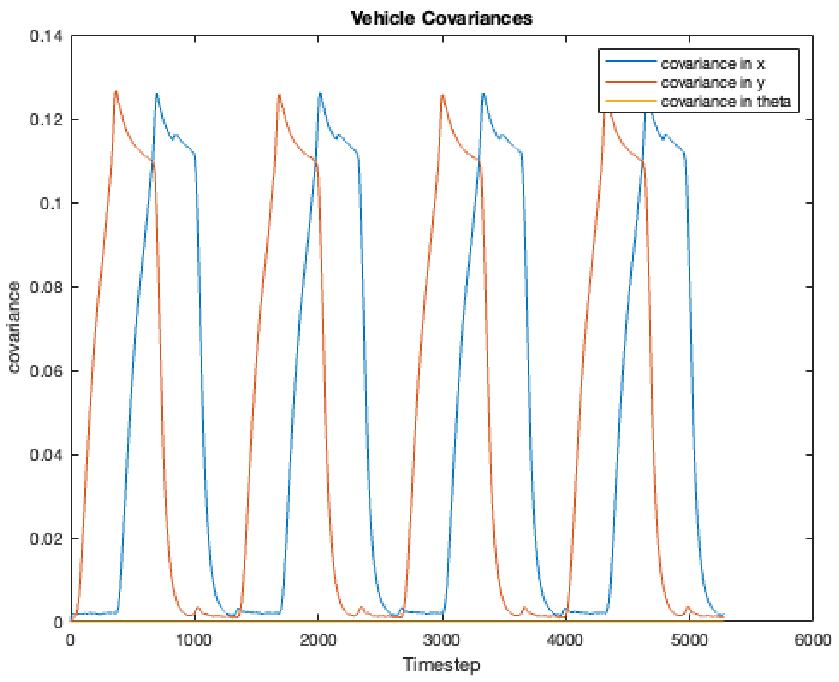


Figure34: Vehicle covariances for q3a

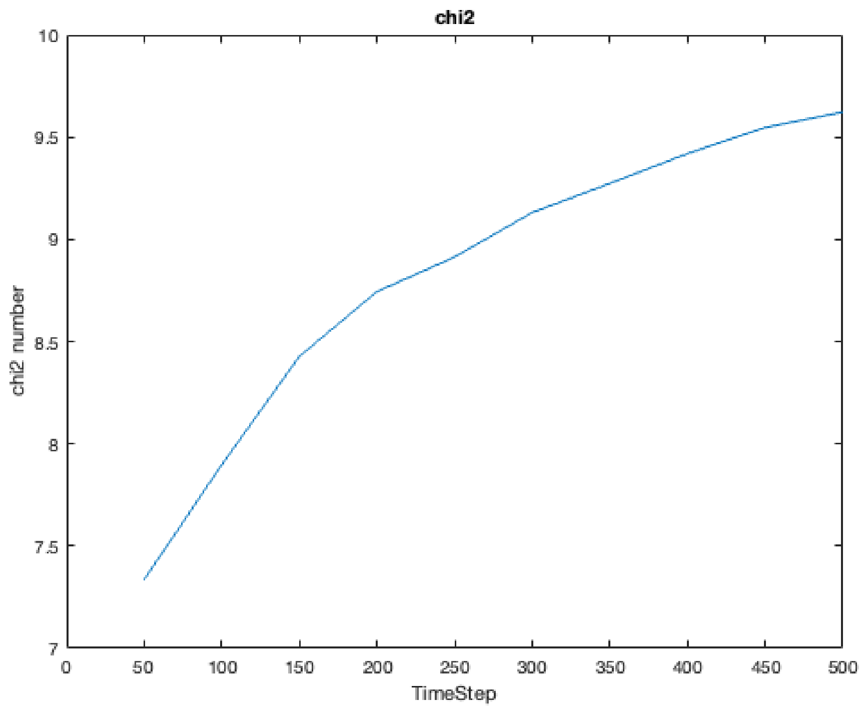


Figure 35:chi2 for q3a

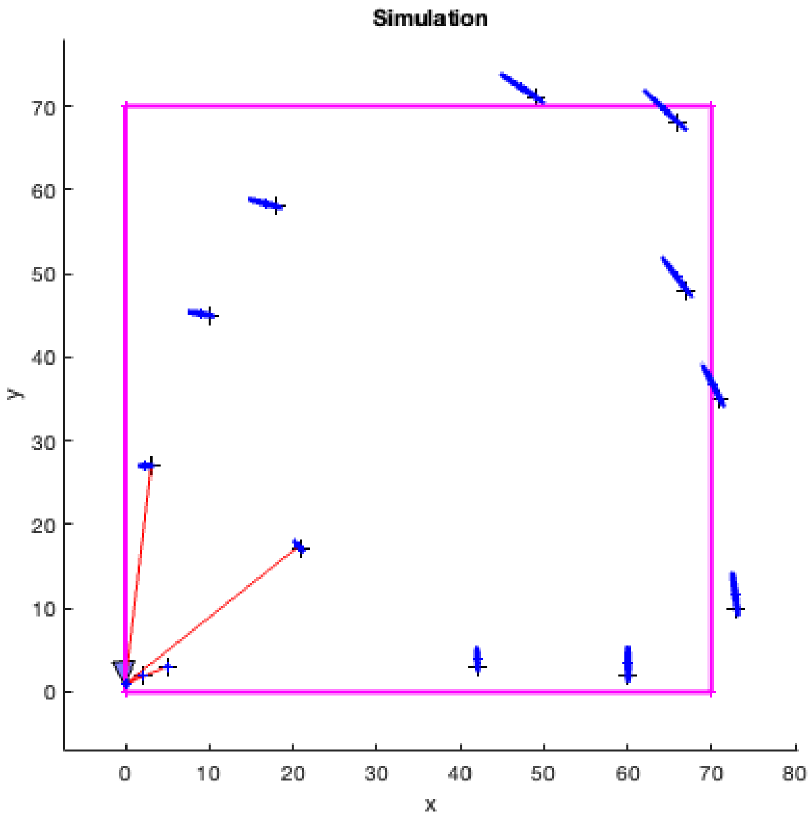


Figure36: Simulation output for q3a

Q3b-i

Graph pruning approach is a kind of method that can delete some trivial vertices and edges in the graph model and hence decrease the computational cost and storage cost.

In our plan, we are going to prune graph by keeping key frames. The central idea is to eliminate the trivial intermediate frames and keep the most important key frames, which are heads and tails in this case.

Q3b-ii

We will modify $deletevehiclePredictionEdges$ to implement data collecting. The main trimming method is placed in the newly created $PruningGraph$

1. As each landmark is attached to vertives, the program records the relationship of the pair of vertexes, and the ID of the vertex is recorded in a variable unique to each landmark.
2. Each variable is processed to record the head and tail of the consecutive IDs.
3. Once recorded, system state vertices other than heads and tails are deleted.
4. Emptying landmark containers

We also defined flag as a public property called $PruneGraph$.

Q3b-iii