SEOUL NATIONAL UNIVERSITY
Department of Naval Architecture and Ocean Engineering

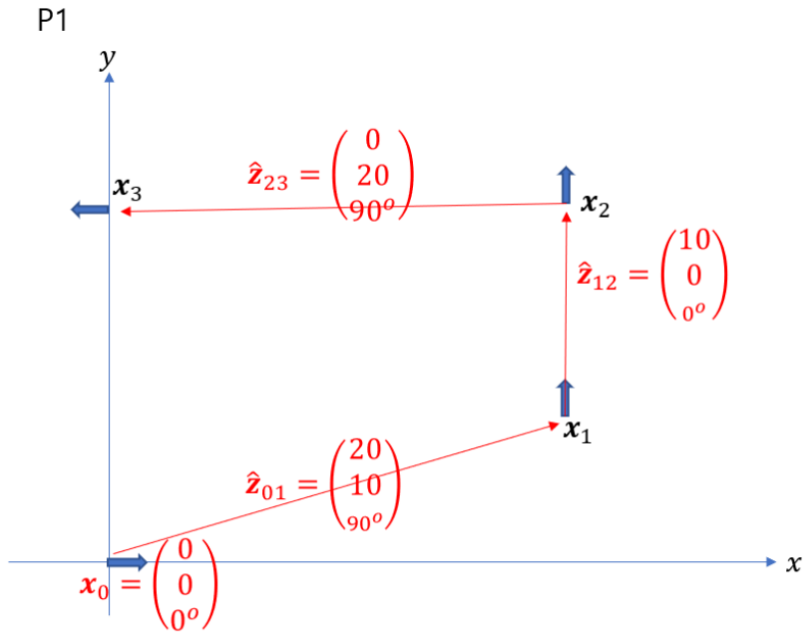
# AUTONOMOUS MOBILE ROBOTICS

# Programming assignment #1

KESSELER CAMILLE 14/10/2022

CONTENTS

## PROBLEM 1:




## THEORY

We want to express the problem as a graph optimization problem so we can solve it recursively using Levenberg-Marquard algorithm.

Then, we want to find the optimal position $\boldsymbol{x}$ given this data. So, we iteratively solve the equation: $H\Delta x = -b$ where $H, b$ are matrix filled with information from the graph. One edge between the node $i$ and $j$ will add a block of information in the matrix. They have the form:

$$\boldsymbol{b}_{ij}^T = [0 \dots e_{ij}^T \Omega_{\text{ij}} A_{ij} \dots e_{ij}^T \Omega_{ij} B_{ij} \dots 0]$$

$$\boldsymbol{H}_{ij} = \begin{bmatrix} A_{ij}^T \Omega_{ij} A_{ij} & A_{ij}^T \Omega_{ij} B_{ij} \\ B_{ij}^T \Omega_{ij} A_{ij} & B_{ij}^T \Omega_{ij} B_{ij} \end{bmatrix}$$

To compute them, we need to find the error $e_{ij}$ and the matrix $A_{ij}, B_{ij}$ which came from the Jacobian matrix. In our case, we can derive them for the 2D case:

$$e_{ij} = \begin{pmatrix} [x_j - x_i]\cos(\theta_{ij} + \theta_i) + [y_j - y_i]\sin(\theta_{ij} + \theta_i) - x_{ij}\cos\theta_{ij} - y_{ij} sin\theta_{ij} \\ -[x_j - x_i]\sin(\theta_{ij} + \theta_i) + [y_j - y_i] cos(\theta_{ij} + \theta_i) + x_{ij}\sin\theta_{ij} - y_{ij} cos\theta_{ij} \\ \theta_j - \theta_i - \theta_{ij} \end{pmatrix}$$

$$A_{ij} = \begin{bmatrix} -cos(\theta_{ij} + \theta_i) & -sin(\theta_{ij} + \theta_i) & -(x_j - x_i)sin(\theta_{ij} + \theta_i) + (y_j - y_i)cos(\theta_{ij} + \theta_i) \\ sin(\theta_{ij} + \theta_i) & -cos(\theta_{ij} + \theta_i) & -(x_j - x_i)cos(\theta_{ij} + \theta_i) - (y_j - y_i)sin(\theta_{ij} + \theta_i) \\ 0 & 0 & -1 \end{bmatrix}$$

$$B_{ij} = \begin{bmatrix} cos(\theta_{ij} + \theta_i) & sin(\theta_{ij} + \theta_i) & 0 \\ -sin(\theta_{ij} + \theta_i) & cos(\theta_{ij} + \theta_i) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where index $i, j$ represent the nodes information and $ij$ is the edge measurement.

The matrix $H$ has no full rank and so cannot be inverse. We need to add a fix global point, here the point of index 0. This means add the Identity matrix in the corresponding part of H. $H[0{:}2, 0{:}2] += I_3$

## CODE DESIGN

First, we need to create a graph instance which contains the position as nodes. The edges represent an odometry measurement between two nodes. We choose to implement the objects (Graph, Nodes, Edges) by structure in our code. We could have use class, but the members of a class object are private by default contrary to the member of a structure which are public by default. We will need to modify the member so in a first approach it will be easier to work with structure.

The Position node structure contains the following information/members:

- Index (int): index defining the node number
- $x$ (double): position along the x-axis in 2D
- $y$ (double): position along the y-axis in 2D
- $theta$ (double): angle between the x-axis and the heading direction of the robot (stored in radian)
- $type$ (int): if 0 then node is variable , if 1 then node is fixe point (useful if we want to marginalized by a part of the node)

The Edge structure contains the following information/members:

- Type_edge (int): if 0 means odometry measurement, if 1 landmark measurement
- Index_node_i (int): index of start edge
- Index_node_j (int): index of end edge
- $x\_measurement$ (double): measurement along x axis between the 2 nodes.
- $y\_measurement$ (double): measurement along y axis between the 2 nodes
- $theta\_measurement$ (double): angle measure between the 2 nodes
- $sigma$ (MatrixXd): Covariance matrix

The Graph structure contains the members and following function members:

- NodesPosition (list<NodePosition>): contains all the position node of the graph
- NodesLandmark (list<NodeLandmark>): contains all the landmark node of the graph
- AddNode(index,x,y,theta,type): function to add a node to the graph
- AddEdge(index_i,index_j, sigma, x,y,theta): function to add an edge to the graph
- Error_ij(edge): function to compute the error submatrix
- A_ij(edge): function to compute the $A_{ij}$ submatrix

- B_ij(edge): function to compute the $B_{ij}$ submatrix
- Optimize(convSeuil, nIteration): function optimizing the position in the graph with the information contained.
- Plot_nodes(): return the nodes information in the console.

Some part of the graph structure can be look up more precisely as they are the core part of the optimization. In this structure we use some of Eigen predefined matrix and vector type as we use Eigen for the linear problem resolution.

The function Optimize() is need to be called to launch a SLAM graph optimization. We define 2 conditions for terminating the procedure (the 2 input parameters): a number maximum of iteration and a minimum of change between two successive corrections $\Delta x$. We keep repeating the following steps until one of the conditions is meet.

We initialize first the matrix $H$ and vector $b$ as matrix of 0 with the size equal to number of nodes in the Nodes vector. Then for each edge in the Edges vector we compute $A_{ij}$, $B_{ij}$ and $e_{ij}$. These blocks matrix and vector are added to $H$, $b$ at the corresponding node index.

$A_{ij}$, $B_{ij}$ and $e_{ij}$ are compute in structure function with the formula derive in the theory part above. As we use an explicit formula for it instead of matrix and vector multiplication, it will not be possible to directly extend it to the 3D case (or we need to derive the formula again).

When all the edges have been process, we directly call the Eigen ldlt().solve Function for solving the linear system $H\Delta x = -b$. Eigen library has multiple function for solving a linear system in function of the matrix $H$ form and properties. We can also preprocess the matrix before solving for speed optimization.

After the optimization, we simply add to the node position and orientation information the results to change their position. This will lead to change the initial position for the next iteration. Also, for the node 0 as we choose to fix it as anchor point, we do not increment it.

Comment:

- Initialization of a structure: in our case we will not tackle this in a first approach but need to be though in case of missing data.

## RESULTS

We start from the initial Node position:

$$x_1 = \begin{pmatrix} 19 \\ 9.5 \\ 88° \end{pmatrix}, x_2 = \begin{pmatrix} 19 \\ 18 \\ 92° \end{pmatrix}, x_3 = \begin{pmatrix} 0 \\ 21 \\ 170° \end{pmatrix}$$

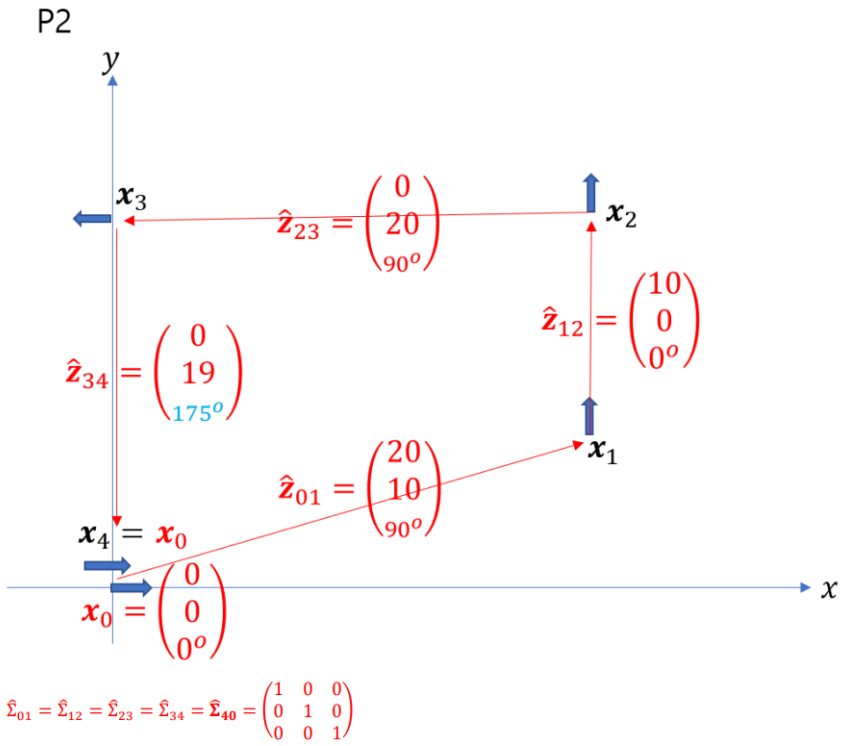
We iterate maximum 10 times or we wait for the change in the $\Delta x$ to be smaller than 0,001. The details of all the iteration step can be found in Annexes B. The result is only 3 iterations with a final error of $1.3 * e^{-14}$:

```
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 20  y: 10  theta: 90
 Position node 2  x: 20  y: 20  theta: 90
 Position node 3  x: -1.47105e-15  y: 20  theta: 180
the error_Tot is 1.33903e-14
```

Comment:

We can see in the solution some that some value are $-1{,}4 * 10^{-15}$ , this is equivalent to 0 with some numerical error.

## PROBLEM 2: LOOP CLOSURE




### THEORY

In this problem, we add an extra point $x_4$ with a loop closure. This is introducing 2 questions: how to detect loop closure and how to represent it in the graph.

- ➔ For detection, it depends highly on the data available. For instance if we have image, we could compare feature point and if the same one are found we say we have a loop closure. In the case where we only have odometry measurement, we can compare the position of the robot with the previous position. If it is close (define a threshold) we detect a loop closure. But it is hard to validate a loop closure as certain with only one data information.

- ➔ When the loop closure is detected, we add an edge in the graph between 2 nodes which represent their relative position. In case the loop is detected using feature points, we can detect that they look at the same environment from different point of view so the edge will represent the virtual motion between these 2 points of view.

## CODE

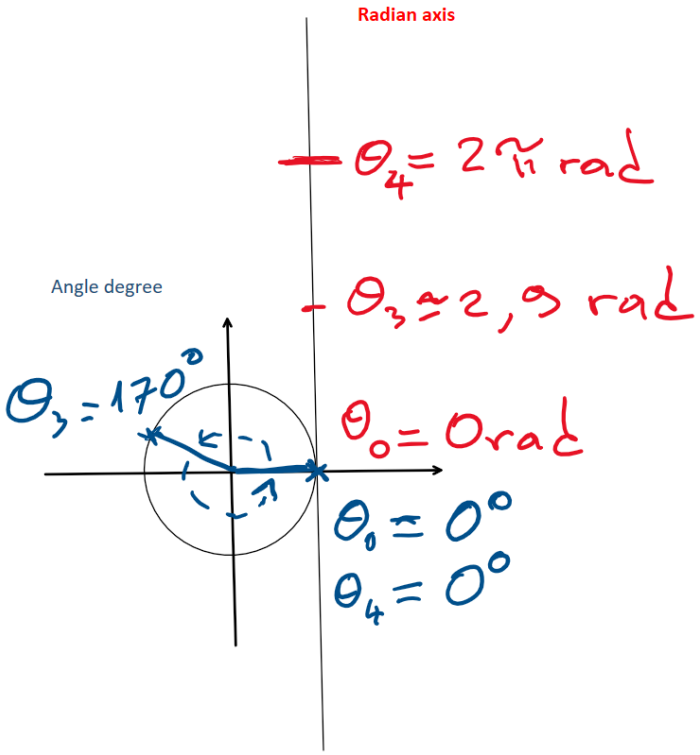
The problem 2 gives us already the position of the loop closure, so we don't need to implement the detection of the loop closure for the moment.

For the virtual measurement in our loop closure in this case we detect when the two position are the same. So I implement a function AddLoopClosure which create an edge with with measurement $(0,0,0)$ to fix at the same position the 2 points. This is working only for loop closure that detect when point are at the exact same position.

Comment:

One question/problem that arise from this problem is the representation of the orientation in the node structure.

For the first problem, I convert directly the degree angle in radian when adding the node or edge to the graph structure. But one error happened when adding the loop closure as I use the radian representation.




When I add the loop closure between node 0 and 4, I add a measurement of $0°$ which is convert to 0 radian between then.

This could be Ok if taken alone. But the edge measurement between the node 3 and 4 constraint the node 4 to be a little bit less than $2\pi$ as show in the figure on the left .

So when optimize like this the final orientation of the node 4 is found to be approximately $160°$ as it is in between 0 and $2\pi$ rad.

So here I add the loop closure to be a measurement of $(0,0,-360°)$.

These show a limit with this angular representation and the conversion from degree to radian. Some code improvements are needed for more robustness.

## RESULTS

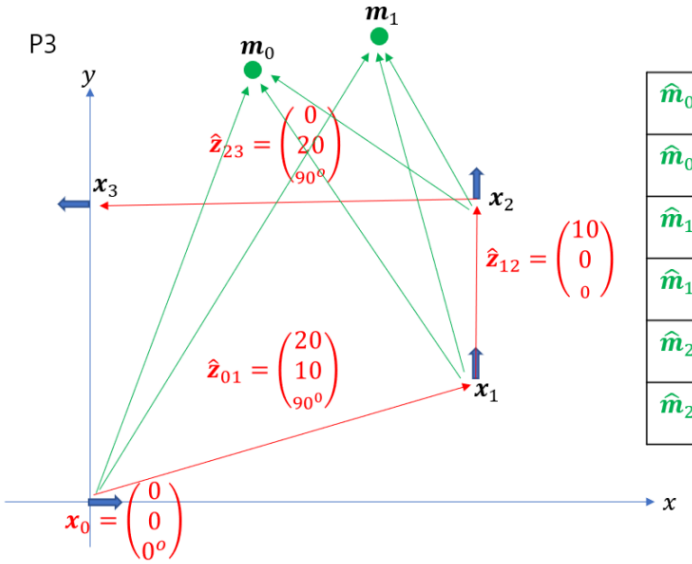
We start from the initial Node position:

$$x_1 = \begin{pmatrix} 19 \\ 9.5 \\ 88° \end{pmatrix}, x_2 = \begin{pmatrix} 19 \\ 18 \\ 92° \end{pmatrix}, x_3 = \begin{pmatrix} 0 \\ 21 \\ 170° \end{pmatrix}, x_3 = \begin{pmatrix} 5 \\ 5 \\ 0° \end{pmatrix}$$

We iterate maximum 10 times or we wait for the change in the $\Delta x$ to be smaller than 0,001. The details of all the iteration step can be found in Annexes C. The final result is for 7 iteration :

```
We are doing iteration 7
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9968  y: 9.99801  theta: 92.3474
 Position node 2  x: 19.584  y: 19.9876  theta: 92.8256
 Position node 3  x: -0.394819  y: 18.9997  theta: 181.21
 Position node 4  x: 0.00318371  y: 0.00198659  theta: 358.105
the error_Tot is 0.000233329
```

## PROBLEM 3: LANDMARKS




| | |
|---|---|
| $\hat{m}_{00}$ | $\begin{pmatrix} 26 \\ 70^o \end{pmatrix}$ |
| $\hat{m}_{01}$ | $\begin{pmatrix} 31 \\ 60^o \end{pmatrix}$ |
| $\hat{m}_{10}$ | $\begin{pmatrix} 19 \\ 35^o \end{pmatrix}$ |
| $\hat{m}_{11}$ | $\begin{pmatrix} 19 \\ 15^o \end{pmatrix}$ |
| $\hat{m}_{20}$ | $\begin{pmatrix} 13 \\ 70^o \end{pmatrix}$ |
| $\hat{m}_{21}$ | $\begin{pmatrix} 10 \\ 40^o \end{pmatrix}$ |

$$\hat{\Sigma}_{01} = \hat{\Sigma}_{12} = \hat{\Sigma}_{23} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\hat{\Sigma}_{00} = \hat{\Sigma}_{01} = \hat{\Sigma}_{10} = \hat{\Sigma}_{11} = \hat{\Sigma}_{20} = \hat{\Sigma}_{21} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

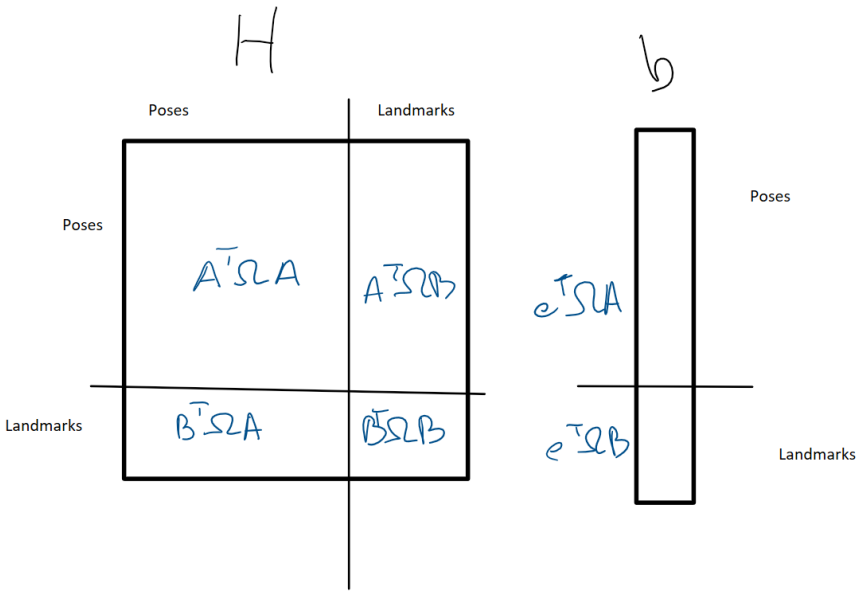
### THEORY

The landmarks are represented by only a x and y coordinates, they do not have orientation information. Also we have different measurement information (distance and angle), the error function will have a different form for landmark/position measurement:

$$\boldsymbol{e}_{il}(x_i, x_l, z_m) = \begin{cases} e_x = \sqrt{(x_l - x_i)^2 + (y_l - y_i)^2} - d_m \\ e_y = \text{atan}\dfrac{y_l - y_i}{x_l - x_i} - \theta_i - \theta_m \end{cases}$$

This form comes from the conversion of the supposed position into a bearing and distance along this axis.

We need to derive the matrix $A_{il}$ and $B_{il}$ so we can fill the matrix $H$ and $b$ with it similarly as what we have done for the position before. Here the information about the landmark is add after all the node position as the figure shows:




$$A_{ij} = \begin{bmatrix} \frac{-(x_l - x_i)}{\sqrt{(x_l - x_i)^2 + (y_l - y_i)^2}} & -\frac{-(y_l - y_i)}{\sqrt{(x_l - x_i)^2 + (y_l - y_i)^2}} & 0 \\ \frac{(y_l - y_i)}{(x_l - x_i)^2 + (y_l - y_i)^2} & \frac{-(x_l - x_i)}{(x_l - x_i)^2 + (y_l - y_i)^2} & -1 \end{bmatrix}$$

$$B_{ij} = \begin{bmatrix} \frac{(x_l - x_i)}{\sqrt{(x_l - x_i)^2 + (y_l - y_i)^2}} & \frac{(y_l - y_i)}{\sqrt{(x_l - x_i)^2 + (y_l - y_i)^2}} \\ \frac{-(y_l - y_i)}{(x_l - x_i)^2 + (y_l - y_i)^2} & \frac{(x_l - x_i)}{(x_l - x_i)^2 + (y_l - y_i)^2} \end{bmatrix}$$

## CODE

The new problem has landmarks. We represent them also by nodes; however, they contain different information compared to the position node as a landmark has no orientation in this application. So, I choose to create a different structure to represent them. The graph instance will save a second list of nodes which are the landmark.

The list of position and landmark are completely decorrelate here, which means that we can find a position of index $i$ and a landmark with index $i$. All the function that takes into entry a node/list of nodes or output a node/list of nodes are defined again with the same name using overloading property of C++. (i.e we can define 2 function with the same name but with different input/output, the code will choose automatically the appropriate one).

## RESULTS

We start from the initial Node position and landmarks:

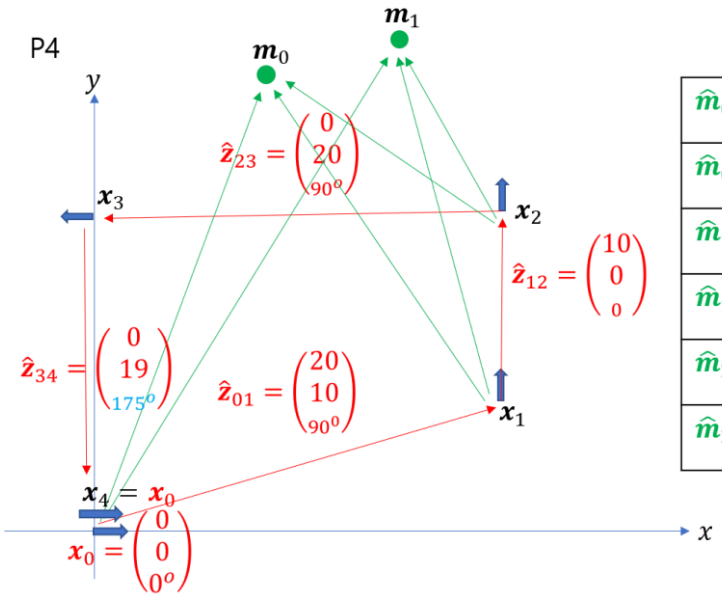
$$x_1 = \begin{pmatrix} 19 \\ 9.5 \\ 88° \end{pmatrix}, x_2 = \begin{pmatrix} 19 \\ 18 \\ 92° \end{pmatrix}, x_3 = \begin{pmatrix} 0 \\ 21 \\ 170° \end{pmatrix}, x_3 = \begin{pmatrix} 5 \\ 5 \\ 0° \end{pmatrix}$$

$$m_0 = \begin{pmatrix} 7 \\ 24 \end{pmatrix}, m_1 = \begin{pmatrix} 15 \\ 29 \end{pmatrix}$$

We iterate maximum 10 times or we wait for the change in the $\Delta x$ to be smaller than 0, 001.The details of all the iteration step can be found in Annexe D. The result is:

```
We are doing iteration 3
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.997  y: 9.99318  theta: 90.0885
 Position node 2  x: 19.9948  y: 20.0156  theta: 89.1797
 Position node 3  x: -0.00318334  y: 20.3019  theta: 179.18
 landmark node 0  x: 7.93227  y: 24.737
 landmark node 1  x: 13.6943  y: 27.8444
the error Tot is 8.96977e-05
```

## PROBLEM 4: LOOP CLOSURE AND LANDMARK




| | |
|---|---|
| $\hat{m}_{00}$ | $\begin{pmatrix} 26 \\ 70^o \end{pmatrix}$ |
| $\hat{m}_{01}$ | $\begin{pmatrix} 31 \\ 60^o \end{pmatrix}$ |
| $\hat{m}_{10}$ | $\begin{pmatrix} 19 \\ 35^o \end{pmatrix}$ |
| $\hat{m}_{11}$ | $\begin{pmatrix} 19 \\ 15^o \end{pmatrix}$ |
| $\hat{m}_{20}$ | $\begin{pmatrix} 13 \\ 70^o \end{pmatrix}$ |
| $\hat{m}_{21}$ | $\begin{pmatrix} 10 \\ 40^o \end{pmatrix}$ |

$$\hat{\Sigma}_{01} = \hat{\Sigma}_{12} = \hat{\Sigma}_{23} = \hat{\Sigma}_{34} = \hat{\Sigma}_{40} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\hat{\Sigma}_{00} = \hat{\Sigma}_{01} = \hat{\Sigma}_{10} = \hat{\Sigma}_{11} = \hat{\Sigma}_{20} = \hat{\Sigma}_{21} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

## CODE

Here we only put together what was develop before in the problem 2 and 3. So the Base_graph.h file do not change.

### RESULTS

We start from the initial Node position and landmarks:

$$x_1 = \begin{pmatrix} 19 \\ 9.5 \\ 88° \end{pmatrix}, x_2 = \begin{pmatrix} 19 \\ 18 \\ 92° \end{pmatrix}, x_3 = \begin{pmatrix} 0 \\ 21 \\ 170° \end{pmatrix}, x_3 = \begin{pmatrix} 5 \\ 5 \\ 0° \end{pmatrix}$$

$$m_0 = \begin{pmatrix} 7 \\ 24 \end{pmatrix}, m_1 = \begin{pmatrix} 15 \\ 29 \end{pmatrix}$$

We iterate maximum 10 times or we wait for the change in the $\Delta x$ to be smaller than 0,001. The details of all the iteration step can be found in Annexe E. The final result is:

```
We are doing iteration 7
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9982  y: 9.99576  theta: 90.2324
 Position node 2  x: 19.9763  y: 20.0135  theta: 92.8012
 Position node 3  x: -0.00460822  y: 19.024  theta: 180.043
 Position node 4  x: 0.00480127  y: 0.0120219  theta: 357.521
 landmark node 0  x: 7.91782  y: 24.7337
 landmark node 1  x: 13.6769  y: 27.8437
the error_Tot is 8.87225e-05
```

## PROBLEM 5: COMPLEX PROBLEM

### DATA

I will use as complex data the Victoria Park dataset as explain by the TA. First, I need to open the data file and read it correctly.

The ground truth and initial data are easy to read as they directly contain the position and direction of our points and landmarks. The odometry/landmark measurement entry data information is under a different format:

| Type of measurement | Index node i | Number of node associate in case of ambiguous measurement | Index node j (maybe node j') | Loop closure (0 or 1) | Number of measurement<br>( not use in our case, always 1) | X | Y | $\theta$ or nothing for landmark<br>Radian |
|---|---|---|---|---|---|---|---|---|

For reading, we first check the type of data: ODOMETRY or LANDMARK.

- Odometry: check if loop closure or no.
  - Loop closure: add a loop closure edge
  - Normal measurement: Add an odometry edges
- Landmark: check the number of landmark associates
  - If one landmark then add landmark edges
  - If 2 landmark then add landmark edges with the first information but less precision so making covariance bigger.

Comment:

This is a set of real data so they are some ambiguities between data association, this is one of the main problem in SLAM. If we create wrong edges in the graph, we will impact the optimization results.

The data do not contain any information about the incertitude matrix. We don't know which captor have been use to measure them so it is hard to estimate the incertitude value. We can then make the hypothesis that we use the Identity matrix for the incertitude. When we have multiple data for the edges we can change the covariance matrix to put less importance to this data.

## CODE MODIFICATION

In order to apply my graph structure to this complex dataset, I need to change some part of it. I create a file Complex_graph.h which is a copy of Base_graph with some modification:

- AddEdge and AddNode store radian data so we don't convert the angle value.
- AddEdge change the data store when we have a landmark => (x,y)
- AddLoopClosure will not be used as we will just add a normal edge with the relative pose given in the data
- Error model for the landmark measurement as we have directly the (x,y) localization as measurement

$$\boldsymbol{e}_{il}(x_i, x_l, z_m) = \begin{cases} \mathrm{e_x} = \mathrm{x_l} - x_m \\ \mathrm{e_y} = \mathrm{y_l} - y_m \end{cases}$$

$$A_{ij} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

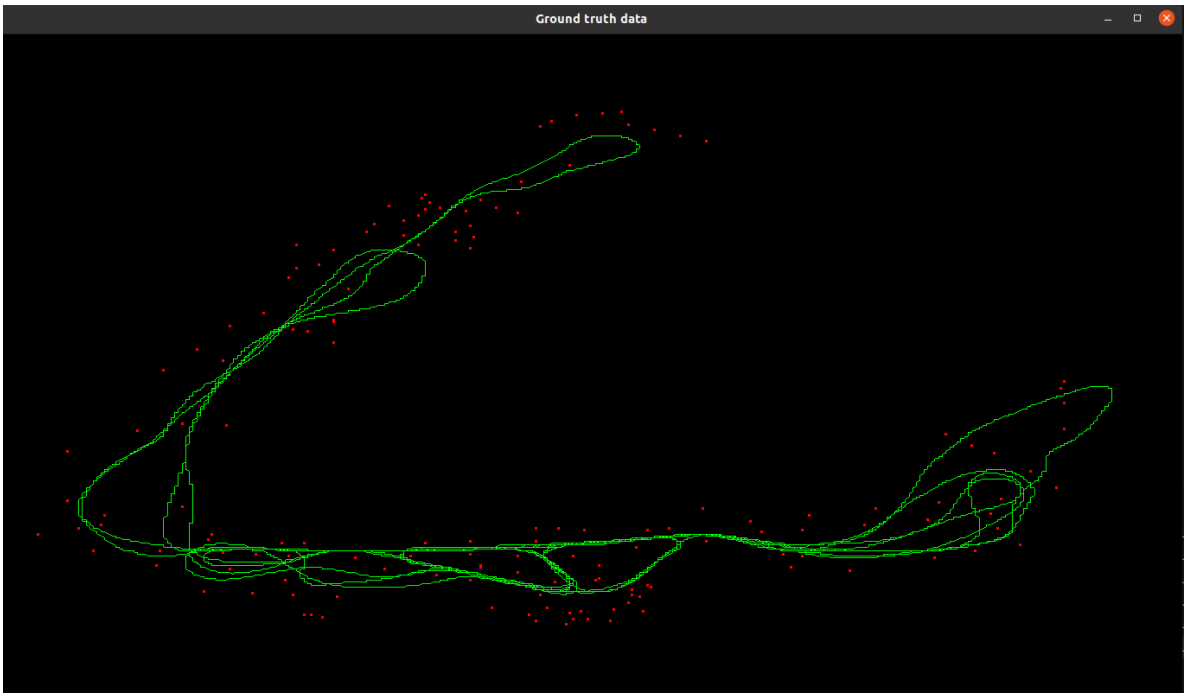
$$B_{ij} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

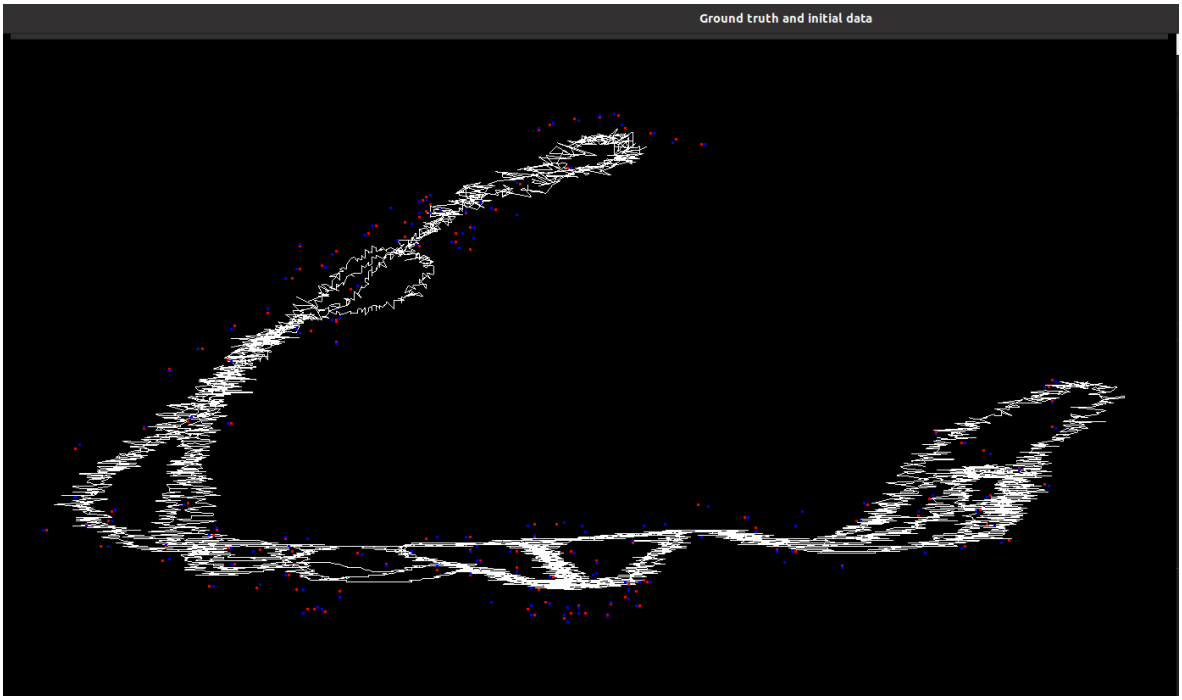
- Create function save_nodes() / save_landmark() in order to save the data into a txt file
- When we create an edge between an unknow node index: we add the empty node to the list.
- Initialize an existing node: We change the value into the existing node.

## VISUALISATION AND RESULTS

For plotting the trajectory, I use Open_GL and more precisely the free_glut library. The code for plotting the result txt file of the cpp is in the Drawing. cpp

First, I plot the initial Dataset with the ground truth in figure 1, we can see that the initial is too noisy and cover entirely the GT truth.




The computation of the inverse matrix is not optimized so it takes some time, so I run only one time the optimization first to see the results. It takes too much time, as dimension of the H matrix is (21206x21206)

Finally, we can compare the optimized one with the Ground truth using the drawing_GT_OPTI.cpp file.

The time taken to compute the optimization show us why during online SLAM, the practical application is to only optimize part of the graph and fixing the rest of the node which are uncorrelated to the actual new measurement. Resolving the equation for the full problem is taking to much and can be done only during offline application.

## ANNEXE A: CMAKELISTS

CMakeLists.txt

```
cmake_minimum_required( VERSION 2.8 )
project( HomeworkP )

set(CMAKE_CXX_FLAGS "-std=c++14")

include_directories( "/usr/include/eigen3" )

set(OpenGL_GL_PREFERENCE "LEGACY")    # warming open GL cleaning
find_package(OpenGL REQUIRED)

find_package(GLUT REQUIRED)

add_executable( HomeworkP1 Problem1.cpp )
add_executable( HomeworkP2 Problem2.cpp )
add_executable( HomeworkP3 Problem3.cpp )
add_executable( HomeworkP4 Problem4.cpp )
add_executable( HomeworkP5 Problem5.cpp )

add_executable(Drawing drawing_GT.cpp)

include_directories( ${OPENGL_INCLUDE_DIRS}  ${GLUT_INCLUDE_DIRS} )
target_link_libraries(Drawing ${OPENGL_LIBRARIES} ${GLUT_LIBRARY} )


add_executable(Drawing2 drawing_GT_IN.cpp)

include_directories( ${OPENGL_INCLUDE_DIRS}  ${GLUT_INCLUDE_DIRS} )
target_link_libraries(Drawing2 ${OPENGL_LIBRARIES} ${GLUT_LIBRARY} )

add_executable(Drawing3 drawing_GT_OPTI.cpp)

include_directories( ${OPENGL_INCLUDE_DIRS}  ${GLUT_INCLUDE_DIRS} )
target_link_libraries(Drawing3 ${OPENGL_LIBRARIES} ${GLUT_LIBRARY} )
```

## ANNEXE B: problem 1 running details

```
ckesseler@ckesseler-pc:~/Homework1_cmake/build$ /home/ckesseler/Homework1_cmake/build/HomeworkP1
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19  y: 9.5  theta: 88
 Position node 2  x: 19  y: 18  theta: 92
 Position node 3  x: 0  y: 21  theta: 170
Start the optimization
We are doing iteration 0
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 20  y: 10  theta: 90
 Position node 2  x: 20.0523  y: 19.9939  theta: 90
 Position node 3  x: 0.169192  y: 19.9591  theta: 180
the error_Tot is 2.73457
We are doing iteration 1
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 20  y: 10  theta: 90
 Position node 2  x: 20  y: 20  theta: 90
 Position node 3  x: 2.08167e-15  y: 20  theta: 180
the error_Tot is 0.181842
We are doing iteration 2
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 20  y: 10  theta: 90
 Position node 2  x: 20  y: 20  theta: 90
 Position node 3  x: -1.47105e-15  y: 20  theta: 180
the error_Tot is 1.33903e-14
```

## ANNEXE C: Problem 2 running details

```
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19  y: 9.5  theta: 88
 Position node 2  x: 19  y: 18  theta: 92
 Position node 3  x: 0  y: 21  theta: 170
 Position node 4  x: 5  y: 5  theta: 0
Start the optimization
We are doing iteration 0
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9153  y: 10.0581  theta: 62.2039
 Position node 2  x: 24.0065  y: 20.1101  theta: -6.84791
 Position node 3  x: 9.10963  y: 52.2494  theta: 62.7882
 Position node 4  x: -24.2136  y: 24.2402  theta: 278.438
the error_Tot is 48.4158
We are doing iteration 1
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9724  y: 10.009  theta: 101.556
 Position node 2  x: 17.7041  y: 21.674  theta: 95.1039
 Position node 3  x: -37.1274  y: 15.0329  theta: 135.466
 Position node 4  x: -18.5231  y: -18.5394  theta: 322.286
the error_Tot is 73.7116
We are doing iteration 2
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9894  y: 9.98309  theta: 111.178
 Position node 2  x: 16.0165  y: 19.3825  theta: 123.064
 Position node 3  x: -0.674051  y: -9.17083  theta: 175.852
 Position node 4  x: 9.65413  y: -9.61816  theta: 352.089
the error_Tot is 52.89
We are doing iteration 3
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 20.0459  y: 10.0429  theta: 84.6573
 Position node 2  x: 20.8298  y: 21.2494  theta: 113.772
 Position node 3  x: -0.515907  y: 13.0874  theta: 198.888
 Position node 4  x: -1.66442  y: -1.66751  theta: 342.447
the error_Tot is 26.7183
```

```
We are doing iteration 4
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.992  y: 9.99011  theta: 97.2869
 Position node 2  x: 18.445  y: 20.1096  theta: 99.8987
 Position node 3  x: -1.84245  y: 17.2063  theta: 184.146
 Position node 4  x: 0.503757  y: -0.484977  theta: 359.482
the error_Tot is 5.66379
We are doing iteration 5
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9967  y: 9.99772  theta: 92.5266
 Position node 2  x: 19.5658  y: 20.0432  theta: 92.9555
 Position node 3  x: -0.491526  y: 19.0612  theta: 181.276
 Position node 4  x: -0.00744736  y: -0.00884945  theta: 358.216
the error_Tot is 2.65317
We are doing iteration 6
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9968  y: 9.99802  theta: 92.348
 Position node 2  x: 19.5841  y: 19.9877  theta: 92.826
 Position node 3  x: -0.394633  y: 18.9998  theta: 181.209
 Position node 4  x: 0.00314711  y: 0.00199194  theta: 358.105
the error_Tot is 0.129736
We are doing iteration 7
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9968  y: 9.99801  theta: 92.3474
 Position node 2  x: 19.584  y: 19.9876  theta: 92.8256
 Position node 3  x: -0.394819  y: 18.9997  theta: 181.21
 Position node 4  x: 0.00318371  y: 0.00198659  theta: 358.105
the error_Tot is 0.000233329
```

## ANNEXE D: Problem 3 running details

```
ckesseler@ckesseler-pc:~/Homework1_cmake/build$ /home/ckesseler/Homework1_cmake/build/HomeworkP3
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19  y: 9.5  theta: 88
 Position node 2  x: 19  y: 18  theta: 92
 Position node 3  x: 0  y: 21  theta: 170
 landmark node 0  x: 7  y: 24
 landmark node 1  x: 15  y: 29
Start the optimization
We are doing iteration 0
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 20.0025  y: 9.99569  theta: 89.5037
 Position node 2  x: 20.1487  y: 19.9508  theta: 88.6426
 Position node 3  x: 0.336705  y: 20.3662  theta: 178.643
 landmark node 0  x: 8.01982  y: 24.7465
 landmark node 1  x: 13.2581  y: 28.0446
the error_Tot is 3.5302
We are doing iteration 1
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9977  y: 9.99348  theta: 90.1115
 Position node 2  x: 19.9922  y: 20.0149  theta: 89.2057
 Position node 3  x: -0.00626144  y: 20.294  theta: 179.206
 landmark node 0  x: 7.9308  y: 24.737
 landmark node 1  x: 13.6936  y: 27.8488
the error_Tot is 0.622722
We are doing iteration 2
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9971  y: 9.99318  theta: 90.0883
 Position node 2  x: 19.9948  y: 20.0156  theta: 89.1798
 Position node 3  x: -0.00313147  y: 20.3019  theta: 179.18
 landmark node 0  x: 7.9323  y: 24.737
 landmark node 1  x: 13.6943  y: 27.8443
the error_Tot is 0.0102005
We are doing iteration 3
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.997  y: 9.99318  theta: 90.0885
 Position node 2  x: 19.9948  y: 20.0156  theta: 89.1797
 Position node 3  x: -0.00318334  y: 20.3019  theta: 179.18
 landmark node 0  x: 7.93227  y: 24.737
 landmark node 1  x: 13.6943  y: 27.8444
the error_Tot is 8.96977e-05
```

## ANNEXE E: Problem 4 running details

```
ckesseler@ckesseler-pc:~/Homework1_cmake/build$ /home/ckesseler/Homework1_cmake/build/HomeworkP4
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19  y: 9.5  theta: 88
 Position node 2  x: 19  y: 18  theta: 92
 Position node 3  x: 0  y: 21  theta: 170
 Position node 4  x: 5  y: 5  theta: 0
 landmark node 0  x: 7  y: 24
 landmark node 1  x: 15  y: 29
Start the optimization
We are doing iteration 0
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.8093  y: 9.98873  theta: 88.6518
 Position node 2  x: 19.9111  y: 19.9281  theta: 3.83441
 Position node 3  x: 4.45122  y: 48.702  theta: 84.2309
 Position node 4  x: -22.8877  y: 22.741  theta: 263.286
 landmark node 0  x: 7.92015  y: 24.8731
 landmark node 1  x: 13.515  y: 28.1046
the error_Tot is 43.766
We are doing iteration 1
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9879  y: 10.0328  theta: 90.872
 Position node 2  x: 19.8695  y: 20.0476  theta: 101.908
 Position node 3  x: -30.7545  y: 13.5185  theta: 154.348
 Position node 4  x: -17.9216  y: -18.0499  theta: 308.525
 landmark node 0  x: 7.81984  y: 24.738
 landmark node 1  x: 13.5704  y: 27.8688
the error_Tot is 64.5838
We are doing iteration 2
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 20.0418  y: 10.0379  theta: 90.5019
 Position node 2  x: 20.0119  y: 20.1001  theta: 115.965
 Position node 3  x: 2.0555  y: 3.51908  theta: 180.003
 Position node 4  x: 7.97676  y: -7.89682  theta: 333.882
 landmark node 0  x: 7.90034  y: 24.7126
 landmark node 1  x: 13.5799  y: 27.8413
the error_Tot is 44.1677
We are doing iteration 3
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 20.0494  y: 10.033  theta: 89.474
 Position node 2  x: 20.1709  y: 20.1497  theta: 95.6179
 Position node 3  x: -3.64217  y: 17.8241  theta: 205.496
 Position node 4  x: 1.49463  y: 1.51273  theta: 345.136
 landmark node 0  x: 8.00863  y: 24.7312
 landmark node 1  x: 13.6846  y: 27.846
the error_Tot is 19.1851
We are doing iteration 4
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9937  y: 9.99514  theta: 90.2079
 Position node 2  x: 19.9712  y: 20.0111  theta: 90.7826
 Position node 3  x: -0.136042  y: 20.0554  theta: 176.318
 Position node 4  x: -0.271092  y: 0.282312  theta: 355.672
 landmark node 0  x: 7.915  y: 24.7362
 landmark node 1  x: 13.6795  y: 27.8472
the error_Tot is 4.71976
We are doing iteration 5
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9984  y: 9.99598  theta: 90.2343
 Position node 2  x: 19.9763  y: 20.0139  theta: 92.8994
 Position node 3  x: -0.0283294  y: 18.9857  theta: 179.94
 Position node 4  x: -0.00339344  y: 0.00424484  theta: 357.337
 landmark node 0  x: 7.91759  y: 24.7337
 landmark node 1  x: 13.6763  y: 27.8437
the error_Tot is 1.14501
We are doing iteration 6
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9982  y: 9.99576  theta: 90.2324
 Position node 2  x: 19.9763  y: 20.0135  theta: 92.8016
 Position node 3  x: -0.00464739  y: 19.024  theta: 180.043
 Position node 4  x: 0.0047845  y: 0.0120069  theta: 357.52
 landmark node 0  x: 7.91782  y: 24.7337
 landmark node 1  x: 13.6769  y: 27.8437
the error_Tot is 0.0465831
```

```
We are doing iteration 7
 Position node 0  x: 0  y: 0  theta: 0
 Position node 1  x: 19.9982  y: 9.99576  theta: 90.2324
 Position node 2  x: 19.9763  y: 20.0135  theta: 92.8012
 Position node 3  x: -0.00460822  y: 19.024  theta: 180.043
 Position node 4  x: 0.00480127  y: 0.0120219  theta: 357.521
 landmark node 0  x: 7.91782  y: 24.7337
 landmark node 1  x: 13.6769  y: 27.8437
the error_Tot is 8.87225e-05
```