SEOUL NATIONAL UNIVERSITY
Department of Naval Architecture and Ocean Engineering

AUTONOMOUS MOBILE ROBOTICS

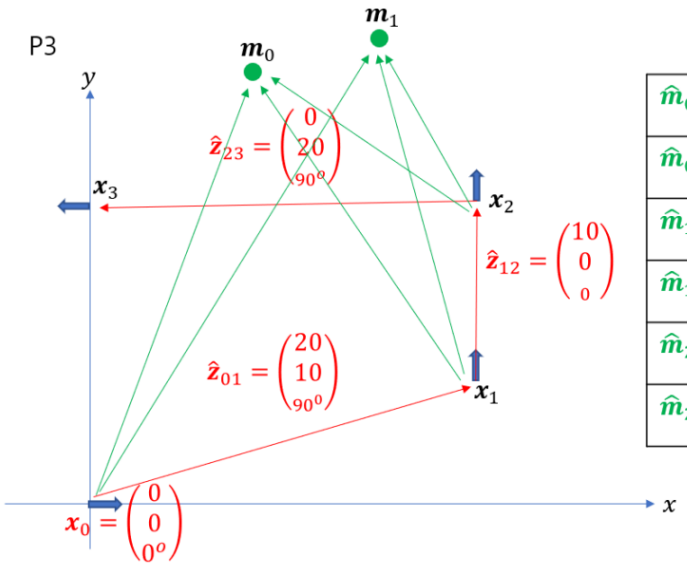
Programming assignment #2

KESSELER CAMILLE 13/11/2022

CONTENTS

POSE ESTIMATION

We recapitulate the data of the PA#1- p3 that we will be using for the pose estimation problem.



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

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

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

And the initial guess for the positions and the landmarks are:

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

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

CERES

We want to use the Ceres library to solve the above problem. The details of the ceres function input/output used in the code is given in Annexe A. Just briefly we need to:

- Graph problem creation
- Filling the graph with odometry and landmark edges
- Fix the first node of graph
- define the solver options.

The Jacobian of the error reprojection function can be define analytically in our case but ceres solver have an automatic differentiation function that make life easier. The AutodiffcostFunction parameters are: error function , dimension of residual, dimension of x, dimension of y . Here the classfo the node/node implementation :

```
// Function that define the reprojection error for pose/pose edge
struct ErrorFunctionPosePose {
    ErrorFunctionPosePose(double observed_x, double observed_y,double observed_theta)
        : observed_x(observed_x), observed_y(observed_y),observed_theta(observed_theta) {}

    template <typename T>
    bool operator()(const T* const pose_i,
                    const T* const pose_j,
                    T* residuals) const {

    //Cos and Sin computation as it is in radian

    T sin_ij=sin(T(observed_theta)*M_PI/ T(180));
    T cos_ij=cos(T(observed_theta)*M_PI/ T(180));
    T sin_i=sin((pose_i[2]+T(observed_theta))*M_PI/ T(180));
    T cos_i=cos((pose_i[2]+T(observed_theta))*M_PI/ T(180));

    // The error is the difference between the predicted and observed position

    residuals[0] = (pose_j[0]-pose_i[0])*cos_i+ (pose_j[1]-pose_i[1])*sin_i - T(observed_x)*cos_ij- T(observed_y)*sin_ij;
    residuals[1] = -(pose_j[0]-pose_i[0])*sin_i+ (pose_j[1]-pose_i[1])*cos_i + T(observed_x)*sin_ij- T(observed_y)*cos_ij;
    residuals[2] = pose_j[2]-pose_i[2]-T(observed_theta);
    return true;
  }

   // Factory to hide the construction of the CostFunction object from
   // the client code.
   static ceres::CostFunction* Create(const double observed_x,
                                      const double observed_y,
                                      const double observed_theta) {
     return (new ceres::AutoDiffCostFunction<ErrorFunctionPosePose, 3,3,3>(
                 new ErrorFunctionPosePose(observed_x, observed_y, observed_theta)));
   }

  // the observation data stored when create the structure
  double observed_x;
  double observed_y;
  double observed_theta;
};
```

The results found for the position and landmarks after optimization is:

```
ckesseler@ckesseler-pc:~/PA_2/build$ /home/ckesseler/PA_2/build/PE_Ceres
iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_radius  ls_iter  iter_time  total_time
   0  4.073520e+02    0.00e+00    1.18e+02   0.00e+00   0.00e+00  1.00e+04        0    4.29e-05    1.23e-04
   1  9.757564e+00    3.98e+02    1.79e+01   1.12e+01   9.85e-01  3.00e+04        1    5.10e-05    2.16e-04
   2  2.186389e+00    7.57e+00    1.08e+00   1.26e+00   9.98e-01  9.00e+04        1    1.72e-05    2.43e-04
   3  2.162062e+00    2.43e-02    2.54e-03   3.96e-02   1.00e+00  2.70e+05        1    1.19e-05    2.67e-04
```

```
Solver Summary (v 2.1.0-eigen-(3.3.7)-lapack-suitesparse-(5.7.1)-cxsparse-(3.2.0)-eigensparse-no_openmp)

                                     Original                  Reduced
Parameter blocks                            6                        5
Parameters                                 16                       13
Residual blocks                             9                        9
Residuals                                  21                       21

Minimizer                        TRUST_REGION

Dense linear algebra library            EIGEN
Trust region strategy     LEVENBERG_MARQUARDT

                                       Given                     Used
Linear solver                     DENSE_SCHUR               DENSE_SCHUR
Threads                                     1                         1
Linear solver ordering                AUTOMATIC                   3,2
Schur structure                         d,d,3                     d,d,d

Cost:
Initial                          4.073520e+02
Final                            2.162062e+00
Change                           4.051900e+02

Minimizer iterations                        4
Successful steps                            4
Unsuccessful steps                          0

Time (in seconds):
Preprocessor                     0.000080

  Residual only evaluation       0.000008 (4)
  Jacobian & residual evaluation 0.000046 (4)
  Linear solver                  0.000061 (4)
Minimizer                        0.000207

Postprocessor                    0.000002
Total                            0.000289

Termination:                     CONVERGENCE (Function tolerance reached. |cost_change|/cost: 3.873250e-07 <= 1.000000e-06)
```

```
Pose 0 x: 0 y: 0 theta: 0
Pose 1 x: 20.0938 y: 9.27235 theta: 89.9896
Pose 2 x: 21.2377 y: 20.0513 theta: 89.6631
Pose 3 x: 1.238 y: 20.1689 theta: 179.663
Landmark 1 x: 9.13047 y: 24.608
Landmark 2 x: 15.3093 y: 27.2113
```

G2O

We now want to do the same thing with g2o so first we need to define the optimizer and solver (details in Annexe B). Then we can add to this graph the pose, landmark and measurement, g2o has preimplement class for the 3D problem, but for 2D we need to refine some class. For this I re-use some already existing 2D class for representing our nodes and edges from the g2o simple example.

- Pose: g2o class .
- Landmark: g2o class .
- Pose-Pose measurement: g2o class
- Pose-landmark measurement: I add to create my class has our measurement is in terms of angle.

The g2o class for pose and landmark is actually matching our definition of the problem so we can use it directly. For the measurement edge between 2 pose nodes, we verify that the error measurement definition corresponds to our definition of the error as show the following screenshot of the g2o class definition:

```
class EdgeSE2PointXY_angle : public
BaseBinaryEdge<2,Eigen::Vector2d, VertexSE2, VertexPointXY>
{
    public:
        EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    void computeError()
    {
        const VertexSE2* Pose = static_cast<const VertexSE2*>
(_vertices[0]);
        const VertexPointXY* landmark = static_cast<const
VertexPointXY*>(_vertices[1]);

        //compute estimate : convert from xy to angle/distance

        Eigen::Vector2d estimate;

        Vector3d x=Pose->estimate().toVector();
        Vector2d l=landmark->estimate();

        estimate[0]=sqrt((l[0]-x[0])*(l[0]-x[0])+(l[1]-
x[1])*(l[1]-x[1]));
        estimate[1]=atan2(l[1]-x[1],l[0]-x[0])-x[2];


        //to do
        _error =  estimate -_measurement;
    }
```

The edge that represents measurement between a pose and landmark is a little bit more complicated as the g2o class predefined actually is using error measurement in terms of x and y axis. But in our case the error definition is in term of distance and angle as the data from the captor are like this. So we need to created a class for this edge called EdgeSE2PointXY_angle, this class inherit from the g2o class BaseBinaryEdge and just redefined the error measurement (the Jacobian is automatic compute internally in g2o)

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

I used the following code as reference:

https://vincentqin.gitee.io/blogresource-4/slam/g2o-details.pdf

https://github.com/RainerKuemmerle/g2o/tree/master/g2o/examples/tutorial_slam2d

Comment:

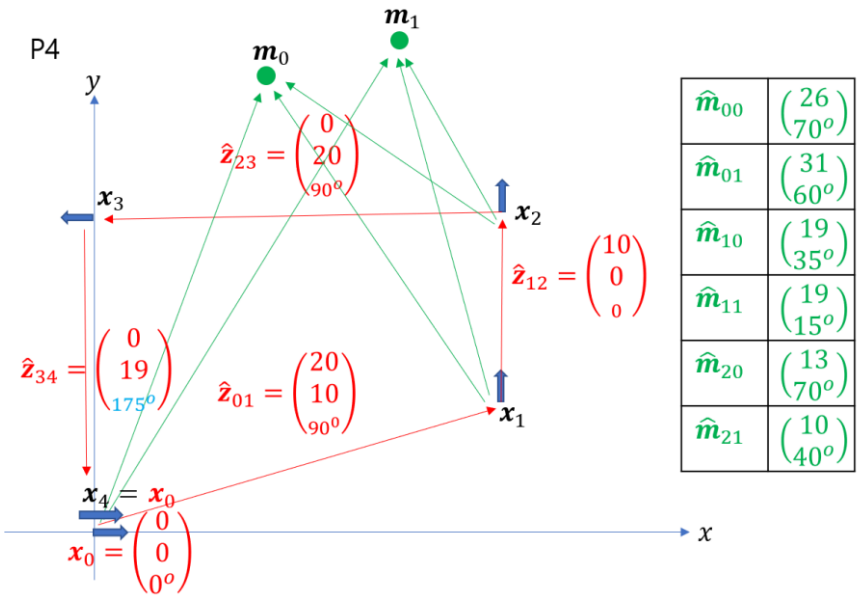
- As the SE2 class use sin and cos in radian, I pass the angle in radian into the node this time.

The result is :

```
ckesseler@ckesseler-pc:~/PA_2/build$ /home/ckesseler/PA_2/build/PE_g2o
start optimization
iteration= 0     chi2= 0.517794  time= 6.94962e-05       cumTime= 6.94962e-05    edges= 9        schur= 0        lambda= 0.001247
levenbergIter= 1
iteration= 1     chi2= 0.031559  time= 1.4652e-05        cumTime= 8.41483e-05    edges= 9        schur= 0        lambda= 0.000416
levenbergIter= 1
iteration= 2     chi2= 0.031500  time= 1.1547e-05        cumTime= 9.56953e-05    edges= 9        schur= 0        lambda= 0.000277
levenbergIter= 1
iteration= 3     chi2= 0.031500  time= 1.01123e-05       cumTime= 0.000105808    edges= 9        schur= 0        lambda= 0.000185
levenbergIter= 1
iteration= 4     chi2= 0.031500  time= 9.70531e-06       cumTime= 0.000115513    edges= 9        schur= 0        lambda= 0.000123
levenbergIter= 1
iteration= 5     chi2= 0.031500  time= 9.70298e-06       cumTime= 0.000125216    edges= 9        schur= 0        lambda= 0.000082
levenbergIter= 1
iteration= 6     chi2= 0.031500  time= 2.31871e-05       cumTime= 0.000148403    edges= 9        schur= 0        lambda= 1.793175
levenbergIter= 6
iteration= 7     chi2= 0.031500  time= 1.0551e-05        cumTime= 0.000158954    edges= 9        schur= 0        lambda= 1.195450
levenbergIter= 1
iteration= 8     chi2= 0.031500  time= 9.93395e-06       cumTime= 0.000168888    edges= 9        schur= 0        lambda= 0.796967
levenbergIter= 1
iteration= 9     chi2= 0.031500  time= 1.03218e-05       cumTime= 0.00017921     edges= 9        schur= 0        lambda= 0.531311
levenbergIter= 1
optimization done
Pose 0 x: 0 y: 0 theta: 0
Pose 1 x: 19.997 y: 9.99318 theta: 90.0885
Pose 2 x: 19.9948 y: 20.0156 theta: 89.1797
Pose 3 x: -0.00318286 y: 20.3019 theta: 179.18
Landmark 1 x: 7.93227 y: 24.737
Landmark 2 x: 13.6943 y: 27.8444
```

LOOP CLOSURE OPTIMIZATION

We will use the data from PA#1-p4 in this part, this means we had a loop closure to the data.



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

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

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

We start from the initial Node position and landmarks:

$$x_1 = \begin{pmatrix} 19 \\ 9.5 \\ 88° \end{pmatrix}, x_2 = \begin{pmatrix} 19 \\ 18 \\ 92° \end{pmatrix}, x_3 = \begin{pmatrix} 0 \\ 21 \\ 170° \end{pmatrix}, x_4 = \begin{pmatrix} 5 \\ 5 \\ 0° \end{pmatrix}$$

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

CERES

The only difference from the problem 1 is the addition of the loop closure. For Ceres a loop closure edge is the same as a classical pose/pose edge so we can just add the following line:

```
ckesseler@ckesseler-pc:~/PA_2/build$ /home/ckesseler/PA_2/build/PE_Ceres_loop
 iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_radius  ls_iter  iter_time  total_time
   0  1.247830e+05    0.00e+00    7.05e+02   0.00e+00   0.00e+00  1.00e+04        0    4.51e-05    1.28e-04
   1  1.599491e+01    1.25e+05    1.89e+01   3.59e+02   1.00e+00  3.00e+04        1    6.60e-05    2.34e-04
   2  5.993225e+00    1.00e+01    1.82e+00   1.57e+00   9.92e-01  9.00e+04        1    1.79e-05    2.65e-04
   3  5.925434e+00    6.78e-02    8.08e-03   5.49e-02   1.00e+00  2.70e+05        1    1.41e-05    2.88e-04
   4  5.925427e+00    6.65e-06    2.60e-04   2.34e-03   1.01e+00  8.10e+05        1    1.31e-05    3.08e-04
```

```
Solver Summary (v 2.1.0-eigen-(3.3.7)-lapack-suitesparse-(5.7.1)-cxsparse-(3.2.0)-eigensparse-no_openmp)

                                     Original                  Reduced
Parameter blocks                            7                        6
Parameters                                 19                       16
Residual blocks                            11                       11
Residuals                                  27                       27

Minimizer                        TRUST_REGION

Dense linear algebra library            EIGEN
Trust region strategy     LEVENBERG_MARQUARDT

                                       Given                     Used
Linear solver                     DENSE_SCHUR               DENSE_SCHUR
Threads                                     1                         1
Linear solver ordering                AUTOMATIC                   3,3
Schur structure                         d,d,3                     d,d,d

Cost:
Initial                          1.247830e+05
Final                            5.925427e+00
Change                           1.247770e+05

Minimizer iterations                        5
Successful steps                            5
Unsuccessful steps                          0

Time (in seconds):
Preprocessor                     0.000083

  Residual only evaluation       0.000011 (5)
  Jacobian & residual evaluation 0.000048 (5)
  Linear solver                  0.000079 (5)
Minimizer                        0.000246

Postprocessor                    0.000006
Total                            0.000335

Termination:                     CONVERGENCE (Function tolerance reached. |cost_change|/cost: 4.517184e-10 <= 1.000000e-06)
```

```
Pose 0 x: 0 y: 0 theta: 0
Pose 1 x: 20.1559 y: 9.27567 theta: 90.9386
Pose 2 x: 21.1152 y: 20.2313 theta: 91.5564
Pose 3 x: 0.465864 y: 19.4524 theta: 182.557
Pose 4 x: 0.656738 y: 0.235679 theta: 358.778
Landmark 1 x: 8.9946 y: 24.3579
Landmark 2 x: 15.0961 y: 27.0454
```

G2O

Similarly, to ceres, the only modification compares to the problem 1 solution is the addition of an edge to the graph.

```
ckesseler@ckesseler-pc:~/PA_2/build$ /home/ckesseler/PA_2/build/PE_g2o_loop
start optimization
iteration= 0     chi2= 2.589972  time= 7.75377e-05       cumTime= 7.75377e-05    edges= 11       schur= 0     lambda= 0.001247
    levenbergIter= 1
iteration= 1     chi2= 0.052238  time= 1.8795e-05        cumTime= 9.63327e-05    edges= 11       schur= 0     lambda= 0.000416
    levenbergIter= 1
iteration= 2     chi2= 0.050018  time= 1.50269e-05       cumTime= 0.00011136     edges= 11       schur= 0     lambda= 0.000277
    levenbergIter= 1
iteration= 3     chi2= 0.050018  time= 1.32411e-05       cumTime= 0.000124601    edges= 11       schur= 0     lambda= 0.000185
    levenbergIter= 1
iteration= 4     chi2= 0.050018  time= 1.32117e-05       cumTime= 0.000137812    edges= 11       schur= 0     lambda= 0.000123
    levenbergIter= 1
iteration= 5     chi2= 0.050018  time= 2.92333e-05       cumTime= 0.000167046    edges= 11       schur= 0     lambda= 2.689763
    levenbergIter= 6
iteration= 6     chi2= 0.050018  time= 1.62981e-05       cumTime= 0.000183344    edges= 11       schur= 0     lambda= 3.586350
    levenbergIter= 2
iteration= 7     chi2= 0.050018  time= 1.25612e-05       cumTime= 0.000195905    edges= 11       schur= 0     lambda= 2.390900
    levenbergIter= 1
iteration= 8     chi2= 0.050018  time= 1.25458e-05       cumTime= 0.000208451    edges= 11       schur= 0     lambda= 1.593933
    levenbergIter= 1
iteration= 9     chi2= 0.050018  time= 2.54307e-05       cumTime= 0.000233882    edges= 11       schur= 0     lambda= 1088.125200
    levenbergIter= 5
optimization done
Pose 0 x: 0 y: 0 theta: 0
Pose 1 x: 19.9982 y: 9.99576 theta: 90.2324
Pose 2 x: 19.9763 y: 20.0135 theta: 92.8012
Pose 3 x: -0.00460813 y: 19.024 theta: -179.957
Pose 4 x: 0.00480163 y: 0.0120218 theta: -2.47857
Landmark 1 x: 7.91782 y: 24.7337
Landmark 2 x: 13.6769 y: 27.8437
```

RESULTS COMPARISON

PROBLEM 5: COMPLEX PROBLEM

DATA

I will use as complex data the Victoria Park dataset as in the PA 1. For reading the data nothing as change. I just correct an error I had when there was multiple possibility for the landmark I need to change the index for the measurement data read.

Just for recap: 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 (not use in our case, always 1)	X	Y	θ or nothing for landmark Radian

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.

G2O

I choose to use g2o, the main part of the code stay similar, I only need to be careful about 1 things:

- the index of landmark and position to not mix them when adding edge later
- error form of the measurement

CODE MODIFICATION

In order to apply my graph structure to this complex dataset, I need to change some part of it:

- When adding the nodes, I add first the position from 0 to landmark_offset-1. Then I add the landmark from landmark_offset to N . So I can differentiate them during the edge passing process.

- Error model for the landmark measurement as we have directly the (x,y) localization as measurement but in the robot coordinates so we need to translate the initial value into the robot coordinate

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

$$A_{ij} = \begin{bmatrix} -cos\theta_i & -\sin\theta_\mathrm{i} & -(\mathrm{x_l} - x_i)\sin\theta_i + (\mathrm{y_l} - y_i)\cos(\theta_\mathrm{i}) \\ \sin\theta_\mathrm{i} & -\cos\theta_\mathrm{i} & (\mathrm{x_i} - \mathrm{x_l})\cos(\theta_\mathrm{i}) - (\mathrm{y_l} - \mathrm{y_i})\sin(\theta_\mathrm{i}) \end{bmatrix}$$

$$B_{ij} = \begin{bmatrix} cos\theta_i & sin\theta_i \\ -\sin\theta_\mathrm{i} & cos\theta_i \end{bmatrix}$$

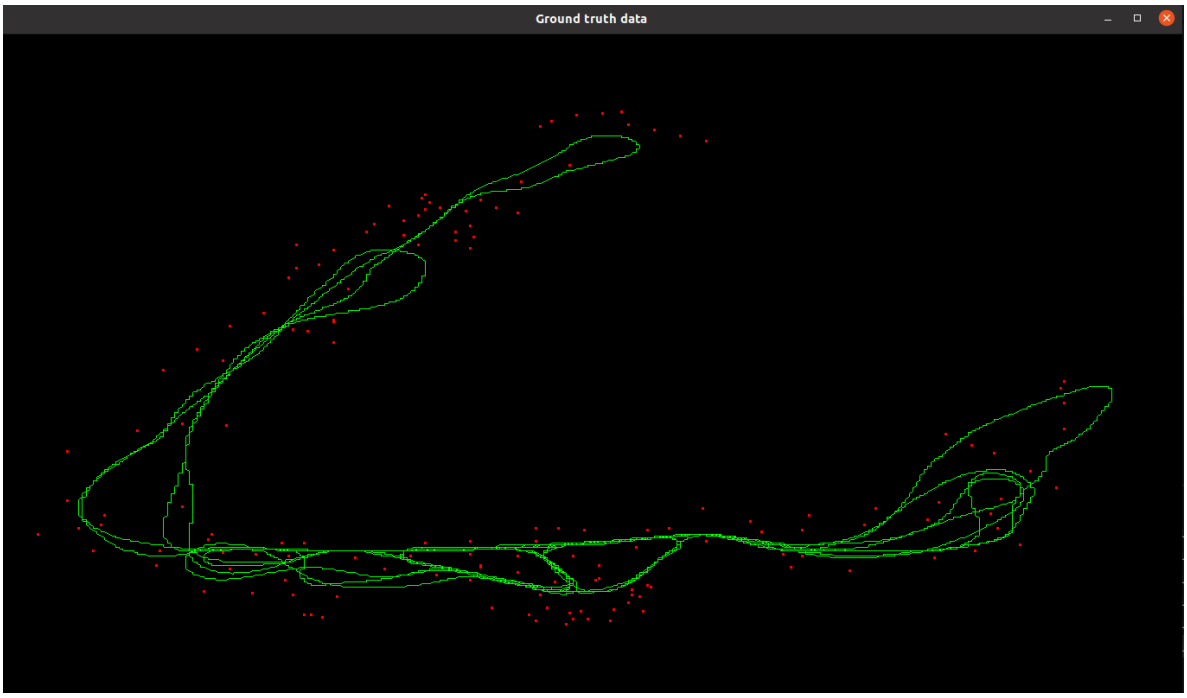
Comment:

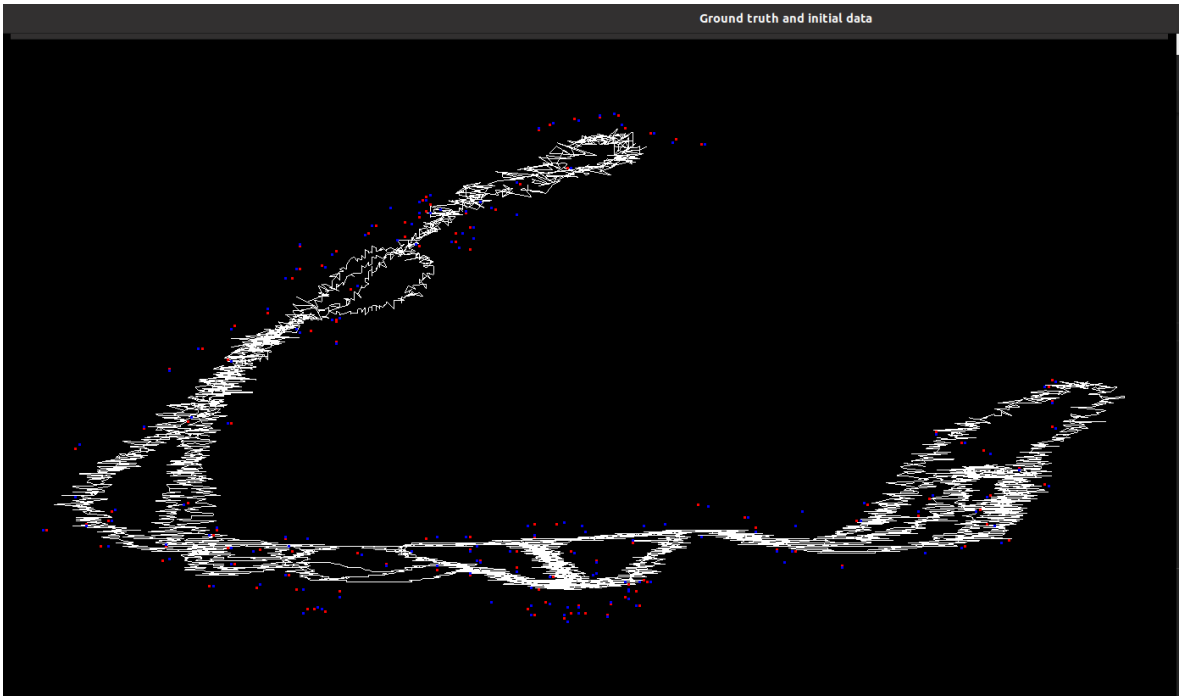
I made a mistake in the PA 1 when deriving the error so I change in this one

VISUALISATION AND RESULTS

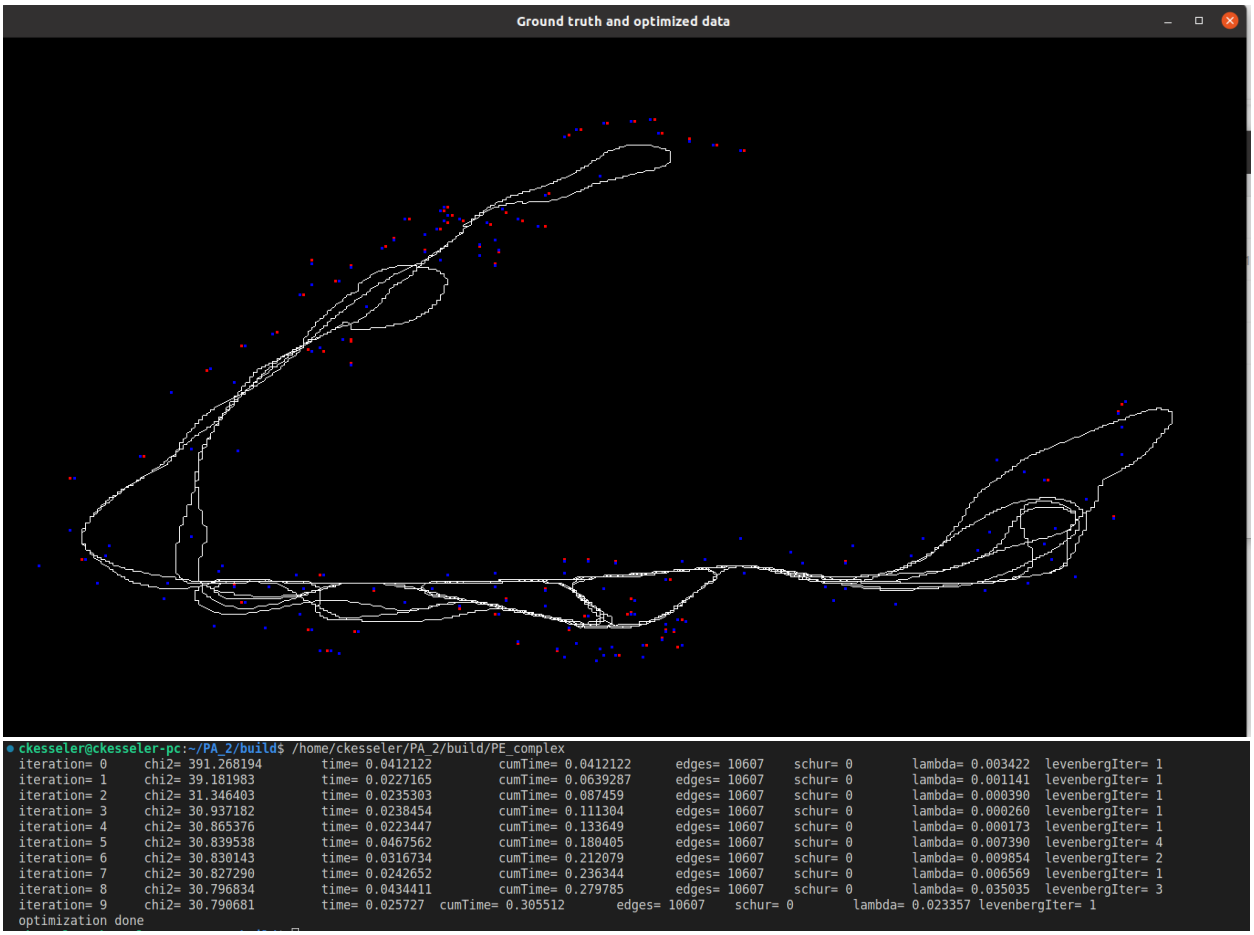
Visualisation is the same as for PA 1 : For plotting the trajectory, I use Open_GL and more precisely the free_glut library. The code for plotting the result txt file of the cpp is in the Drawing. cpp

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.





Finally, we can compare the optimized one with the Ground truth using the drawing_GT_OPTI.cpp file. The trajectory is clearly smoother, and the landmark have been repositioned. Even if for some of them there is still some error in localization. At least the noise in the trajectory have been reduced greatly.



ANNEXE A: CMAKELISTS

Complex_g2o.cpp | CMakeLists.txt

CMakeLists.txt

```
cmake_minimum_required( VERSION 2.8 )
project( PA_2 )

set(CMAKE_CXX_FLAGS "-std=c++14 -O3")

list(APPEND CMAKE_MODULE_PATH /home/ckesseler/slambook2/slambook2/3rdparty/g2o/cmake_modules)
#list(APPEND CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/cmake)

find_package(Ceres REQUIRED)
include_directories(${CERES_INCLUDE_DIRS})
#include_directories( "/usr/include/eigen3" )

# G2O
find_package(G2O REQUIRED)
include_directories(${G2O_INCLUDE_DIRS})

# Eigen
find_package(Eigen3 3.3 REQUIRED NO_MODULE)

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

find_package(GLUT REQUIRED)

add_executable( PE_Ceres Pose_estimation_ceres.cpp )
target_link_libraries(PE_Ceres ${CERES_LIBRARIES})


add_executable( PE_Ceres_loop Pose_estimation_loop_ceres.cpp )
target_link_libraries(PE_Ceres_loop ${CERES_LIBRARIES})

add_executable( PE_g2o Pose_estimation_g2o.cpp )
target_link_libraries(PE_g2o ${G2O_CORE_LIBRARY} ${G2O_STUFF_LIBRARY} Eigen3::Eigen)

add_executable( PE_g2o_loop Pose_estimation_loop_g2o.cpp )
target_link_libraries(PE_g2o_loop ${G2O_CORE_LIBRARY} ${G2O_STUFF_LIBRARY} Eigen3::Eigen)

add_executable( PE_complex Complex_g2o.cpp)
target_link_libraries(PE_complex ${G2O_CORE_LIBRARY} ${G2O_STUFF_LIBRARY} Eigen3::Eigen)
```