SEOUL NATIONAL UNIVERSITY
Department of Naval Architecture and Ocean Engineering

AUTONOMOUS MOBILE ROBOTICS

Programming assignment #5

KESSELER CAMILLE 29/12/2022

CONTENTS

PROJECTION FACTOR

THEORY

The projection factor is the function evaluating the error for each measurement. In our case the measurement is $\boldsymbol{z} = [x_{pixel}, y_{pixel}]^T$ the position of a feature point in the frame. As the association is already provide, we know which landmark is associate to it and it position in the 3D world space $P = [X, Y, Z]$. We want to compute the error:

$$\boldsymbol{e} = \boldsymbol{z} - \boldsymbol{f}(P)$$

where the function $\boldsymbol{f}$ is the conversion between Point in 3D world space and the predicted pixel position. To convert we do the following steps:

- Convert P coordinates into the coordinates of P in the frame reference P_f
- Then normalize it by dividing by the 3rd coordinates. $P' = P_f / Z_f$
- Finally reprojected in the pixel world using the intrinsic parameters.

IMPLEMENTATION

PROJECTION FACTOR

The code should be coherent with the auto differentiation function in ceres, already defined as

new ceres::AutoDiffCostFunction<ProjectionFactor, ***2, 4, 3, 3****>(new ProjectionFactor(****point2D, intrinsic****))*

When creating the cost function object , we need to pass 2 information: the point2D pixel position and the intrinsic of the camera. Also when writing the operator function of the cost function, we know that the results should be of size 2, and then the 3 input of size 4, 3, 3. (which correspond to input of quaternion frame, translation frame, point3D for map point). Then when adding a residual block, we need to pass:

- Cost function
- Robust kernel function if needed for outliers
- Quaternion of frame
- Translation of frame
- Map point

The operator function code is as shown below:

```
bool operator()(const T *const qvec,
                const T *const tvec,
                const T *const point3D,
                T *residuals) const
{
    // Now you have to implement here

   //Pass point from world coordinates to camera coordinates
    T Point_in_camera[3];
    ceres::QuaternionRotatePoint(qvec,point3D, Point_in_camera);
    cout<<"Point_in_camera x"<<Point_in_camera[0]<<" y "<<Point_in_camera[1]<<" z "<<Point_in_camera[2]<<endl;

    //translation
    Point_in_camera[0]+=tvec[0];
    Point_in_camera[1]+=tvec[1];
    Point_in_camera[2]+=tvec[2];
    cout<<"translated point x "<<Point_in_camera[0]<<" y "<<Point_in_camera[1]<<" z "<<Point_in_camera[2]<<endl;

    //Normalized coordinates  (check if +1 or -1)
    T Pc[2];
    Pc[0] =Point_in_camera[0]/Point_in_camera[2];
    Pc[1] =Point_in_camera[1]/Point_in_camera[2];
    cout<<"normalized point "<<Pc[0]<<" other "<<Pc[1]<<endl;

    // Pixel coordinate using intrisic
    T u[2];
    u[0]=T(intrinsic_(0,0))*Pc[0]+T(intrinsic_(0,2));
    u[1]=T(intrinsic_(1,1))*Pc[1]+T(intrinsic_(1,2));
    cout<<"pixel point u "<<u[0]<<" v "<<u[1]<<endl;

    //Residual solution
    residuals[0]=u[0] - T(observed_x_);
    residuals[1]=u[1] - T(observed_y_);
    cout<<"residuals u "<<residuals[0]<<" and v "<<residuals[1]<<endl;

    return true;
```

MAIN CODE

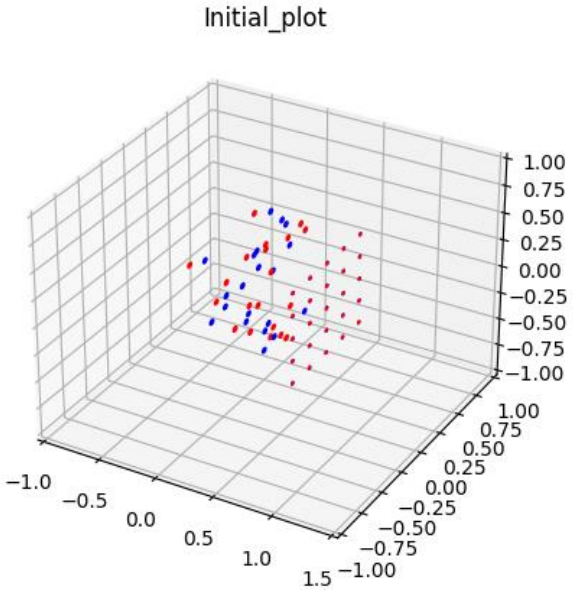
When passing the data to the ceres problem, we need to pass the reference to an array of parameters. This will be our parameters blocks containing the frame SE3 information under the form translation 3 data, then rotation quaternion 4 data. It also contains after all the frame information, the map point position information. These are passed as parameters so we could optimize they position too. However, in this case we have a fixed map known so we can fix the block parameters associate, this means we do not search to optimize the map point localization.

RESULTS

INITIAL DATA AND RESULTS VISUALISATION

To visualize the map point and camera frame position at time t. I choose to use python as it offers simple and quick plot option for 2D visualization. The python script is inside the plot folder. The data are saved in csv file during the main cpp computation and then we open then, read data, plot them and finally save the image in the plot folder.

We can see here the initial noisy data plot with the ground truth :



OPTIMIZATION RESULTS

When running ceres we ask for the summary of the optimization which is the following:

```
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                           65                       40
Parameters                                215                      140
Residual blocks                           473                      473
Residuals                                 946                      946

Minimizer                        TRUST_REGION

Sparse linear algebra library    SUITE_SPARSE
Trust region strategy                  DOGLEG (TRADITIONAL)

                                       Given                     Used
Linear solver          SPARSE_NORMAL_CHOLESKY   SPARSE_NORMAL_CHOLESKY
Threads                                    4                         4
Linear solver ordering                AUTOMATIC                     40
```

We can see that the number of parameter block and parameter is good (20 frames x2 +25 map points =65 original which is reduced to 40 as the landmark are fixed)

Actually, the maximum solver time is reach so there is some problem in the code, I guess. I checked and the reprojection error is too big. The problem is either the rotation of the point or I forgot one step for the reprojection.

We could also visualize the results compare to the ground truth data in the final_plot but the solver fails so there are no update of the parameters vector then it is the same as the initial one.

RMSE COMPUTATION

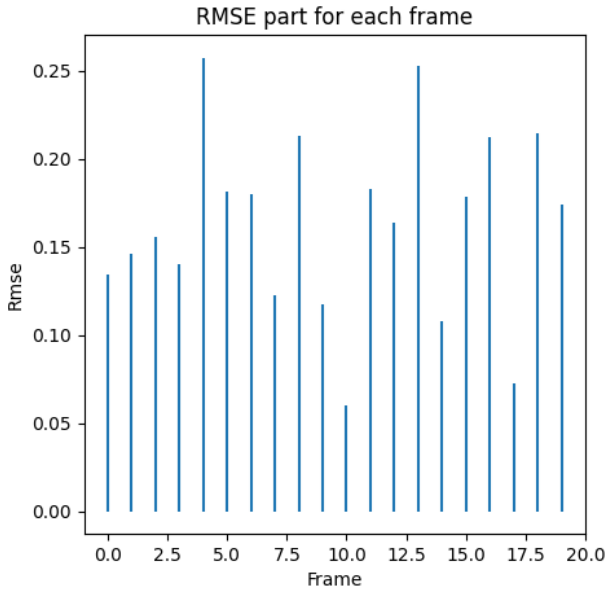
To compare the quality of the trajectory and the map, we compute the RMSE of the frame SE3 information.

The final parameters vector after optimization is repassed in a Sophus SE3 form for easy calculation of multiplication. The formula used is:

$$RMSE = \sqrt{\frac{1}{N_{frame}} \sum_{Frame} \left\| \log\left(T_{gt}^{-1} T_{esti}\right)^{\vee} \right\|_2^2}$$

The final RMSE is 0.171315 which is actually the initial RMSE as we do not update the parameter as the solver do not work.

We plot the term of the RMSE for every frame to see in more details:



REFERENCE

Introduction to Visual SLAM From Theory to Practice book and code from github.