SEOUL NATIONAL UNIVERSITY
Department of Naval Architecture and Ocean Engineering

AUTONOMOUS MOBILE ROBOTICS

Programming assignment #3

KESSELER CAMILLE 24/11/2022

CONTENTS

FEATURE EXTRACTION 2
eXTRACTING AND VISUALIZING 2
Oriented Fast Key Point 2
brief Descriptor 3
Calculation 3
mATCHING 4
Epipolar geometry 5
Fundamental and essential matrix 6
Pose computation 7
Epipolar constraints 8
Reference 10
ANNEXE A: CMakeLists 11

FEATURE EXTRACTION

In this part, we focus on feature extraction in an image. Features are usually defined as some special places in the image, for instance corners, edges or blocks. But this simple geometry are not enough in most application. So the researcher have created some stable local image features such as the SIFT, SURF, ORB and others.

They should follow some properties to be good feature: Repeatability (can be found in different images), Distinctiveness (different features= different expressions), Efficiency (Nb of feature points << Nb of pixel), Locality (Related to a small image area).

We can decompose a feature point into 2 parts: Key point (usually 2D position) and descriptor (usually a vector describing information of the pixel around).We will focus on ORB feature in our code and use the OpenCV library to apply it.

The ORB feature is composed of:

- Key points: oriented FAST corner point.
- Descriptor: BRIEF (Binary Robust Independent Elementary Feature)

The code for this exercise is in the ORB_feature.cpp file. As usual the makefile used for running it is given in Annexe A. (Also the explanation are based on the one from the "Introduction to Visual SLAM From Theory to Practice" book.

EXTRACTING AND VISUALIZING

ORIENTED FAST KEY POINT

The FAST algorithm is known to be very fast and is based on comparing pixel brightness. The basic procedure is as followed:

- Take a pixel p with brightness I_p
- Set Threshold T (can be for instance: $T = 0{,}2 * I_p$)
- Set p as the center pixel and select the 16 pixels on a circle of radius 3 around the center
- Look if : N consecutive points have brightness outside the threshold. Then it is a feature point (usually N=12)

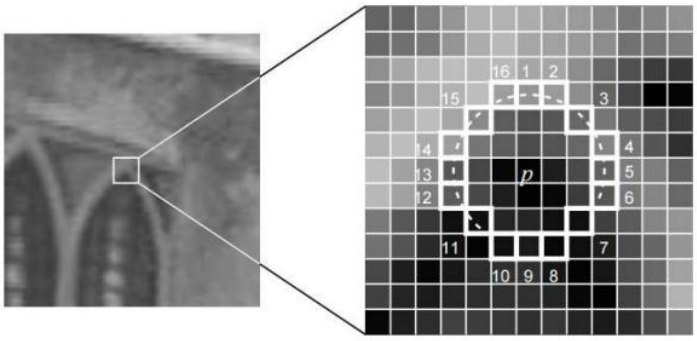


Figure 6-2: FAST key points [39]. This figure is from OpenCV's document.

To speed up process for FAST-12, we can look at pixel 1, 5, 9 and 13 as only if 3 of this pixel is outside the threshold we can potentially have a corner point. Another improvement is done by only selecting some pixel with maximum response in a specific location as they tend to cluster in the same place on the image.

It is very fast to compute but can be improve for direction and scale. Here ORB has 2 extra process to adds them:

- Scale invariance is achieved by using an image pyramid (Matching between different scale image)
- Rotation of features is compute using intensity centroid method (Gray value of the image block as the center of weight)

BRIEF DESCRIPTOR

This is a binary descriptor, i.e a vector containing only 0 and 1. $[0..1\ 0\ 1\ 1\ \ldots 1]$.

This 0 and 1 describe the relation between 2 pixels: if p > q => 1 / else => 0

We compare randomly selected points which is very fast (maybe around 128), also as it is binary it is easy to store and read so suitable for real-time.

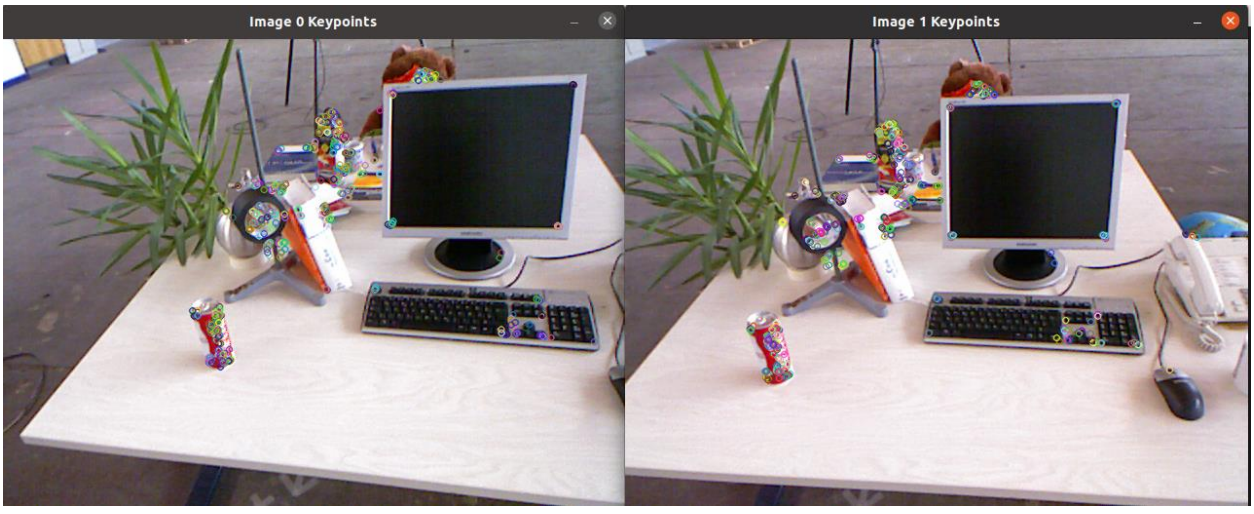
As it does not contain information about the rotation invariance, we use the information of rotation contains in the key point to compute the Steer BRIEF feature.

CALCULATION

To extract ORB feature using OpenCV library, we need to do the following:

- Store the image in Mat format compatible with OpenCV
- Define the ORB Keypoints detector
- Detect it and store in a vector of Keypoints
- Define the ORB descriptor calculator
- Compute it and store in Mat OpenCV descriptor format

Here we can see the resulting extracted Feature points for the Image0 and Image1 in the figure 2:

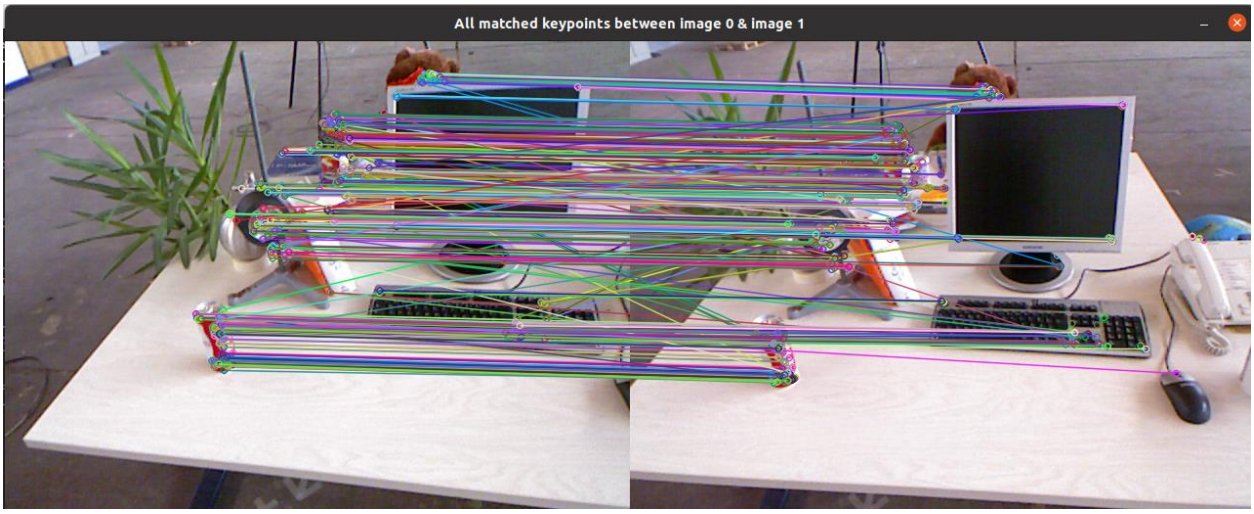


MATCHING

Now we want to match the feature points compute before, to do these steps:

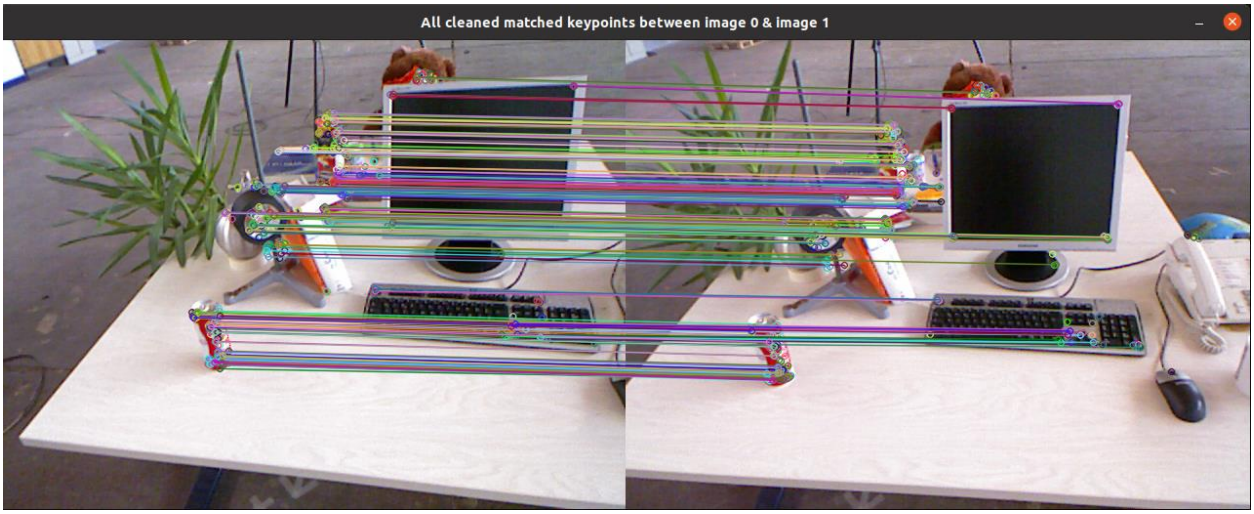
- Define the ORB matcher types, here I choose Brute Force Hamming.
- Match the descriptors of the two images

When doing just these 2 steps we can obtain the following image in the figure 3:



One problem that appears in this image is the presence of outliers in the matching, we can see quite a lot of them so we should try to take them out or reduce their number at least. A simple approach that is based on empirical law is to keep as good matches, the one where the hamming distance satisfies the condition: $H_d^i < 2 * Min(H_d)$.

We add this remove outliers function called CleanerMatches and we obtain a way better matching set as show the following figure 4:



EPIPOLAR GEOMETRY

We know have matched point and want to retrieve the pose of image using this information. For this we can use epipolar geometry.

Briefly, epipolar geometry is a method based on the geometric relation between 2 matched points in 2 images

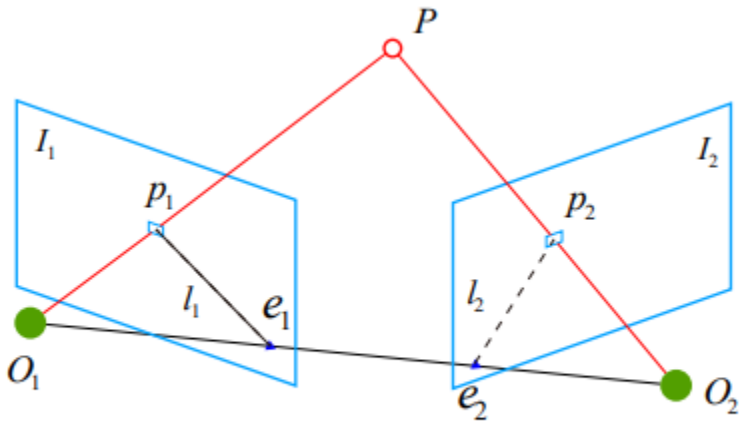


Figure 6-9: The epipolar constraints.

Here we will use homogeneous pixel coordinate so we can define it up to a scale, i.e $s\boldsymbol{p} \simeq \boldsymbol{p}$ **.** From the pinhole model we know the relation between two projection : $\boldsymbol{p}_1 \simeq \boldsymbol{KP}$ and $\boldsymbol{p}_2 \simeq \boldsymbol{K}(\boldsymbol{RP} + \boldsymbol{t})$**.**

We can rewritte it using the coordinates on the normalized plane: $\boldsymbol{x}_1 = \boldsymbol{K}^{-1}\boldsymbol{p}_1, \boldsymbol{x}_2 = \boldsymbol{K}^{-1}\boldsymbol{p}_2$. This lead to :

$$\boldsymbol{x_2} \simeq \boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{t}$$

First left multiply by $\boldsymbol{t}^\wedge$ ($\boldsymbol{t}^\wedge\boldsymbol{t} = 0$), then left multiply again by $\boldsymbol{x}_2^T$ ($\boldsymbol{x}_2^T\boldsymbol{t}^\wedge\boldsymbol{x}_2 = 0$ as orthogonal vector multiplication) and $\simeq$ change to $=$ (as $0 = 0 * s$) , we obtain:

$$\boldsymbol{x}_2^T\boldsymbol{t}^\wedge\boldsymbol{R}\boldsymbol{x}_1 = 0$$

Or repassing in pixel coordinates again:

$$\boldsymbol{p}_2^T\boldsymbol{K}^{-T}\boldsymbol{t}^\wedge\boldsymbol{R}\boldsymbol{K}^{-1}\boldsymbol{p}_1 = 0$$

This 2 relations are called epipolar constraints, from them we define 2 matrices: Fundamental matrix $\boldsymbol{F}$ and essential matrix $\boldsymbol{E}$.

$$\boldsymbol{E} = \boldsymbol{t}^\wedge\boldsymbol{R}$$

$$\boldsymbol{F} = \boldsymbol{K}^{-T}\boldsymbol{E}\boldsymbol{K}^{-1}$$

FUNDAMENTAL AND ESSENTIAL MATRIX

We want to compute the matrix $\boldsymbol{F}, \boldsymbol{E}$, we can start from the epipolar constraints:

$$\boldsymbol{p}_2^T \boldsymbol{F} \boldsymbol{p}_1 = 0$$

If we rewrite $\boldsymbol{F} = \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix}$ matrix in a vector form $\boldsymbol{f} = [f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9]$, $\boldsymbol{p}_2 = \boldsymbol{p}_R = [u_R, v_R, 1]^T$, $\boldsymbol{p}_1 = \boldsymbol{p}_L = [u_L, v_L, 1]^T$ then the equation becomes:

$$[u_R u_L, u_R v_L, u_R, v_R u_L, v_R v_L, v_R, u_L, v_L, 1] * \boldsymbol{f} = 0$$

This equation is for one matched set of points, we can write for n matched point the following linear system:

$$\boldsymbol{A} * \boldsymbol{f} = \begin{bmatrix} u_R^0 u_L^1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ u_R^n u_L^n & \cdots & 1 \end{bmatrix} * \boldsymbol{f} = \boldsymbol{0}$$

To solve this equation, we perform a singular value decomposition of $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\boldsymbol{T}}$ and $\boldsymbol{f}$ is the last columns of the matrix $\boldsymbol{V}$ as it is the eigenvector associated to the eigenvalues 0.

To solve this, we need at least 8 corresponding point as $\boldsymbol{f}$ has dimension 9 so $\boldsymbol{A}$ has at most rank 8. In reality, we usually have more than 8 points so the matrix $\boldsymbol{A}$ will become regular, and the equality problem will become a minimization problem. Then the eigenvector used for $\boldsymbol{f}$ is the one with the smallest eigenvalues which is also the last columns of the matrix $\boldsymbol{V}$.

Then we can reshape it into the Fundamental matrix $\boldsymbol{f} \rightarrow \boldsymbol{F}$

One problem here is that the fundamental matrix is an approximation, so it is not necessarily a matrix of rank 2. So we want to constraint the rank to be 2 and be as close as possible to our estimate. To do so we compute the SVD of $\boldsymbol{F} = \boldsymbol{U_F D V_F^T}$, and set the last eigenvalues of $\boldsymbol{D}$ to be zero. We rebuilt $\boldsymbol{F} = \boldsymbol{U_F Diag}(\boldsymbol{D_1}, \boldsymbol{D_2}, \boldsymbol{0}) \boldsymbol{V_F^T}$.

We could use the algorithm as it is if the problem was well-conditioned. However, when we used pixel coordinated, we can have an important difference between the number in the position pixel vector for instance: $\boldsymbol{p} = [2000,1500,1]^T$ which actually make the resolution numerically instable. So, before it is better to normalize the image matched point data $\boldsymbol{Tp} = \widehat{\boldsymbol{p}}$ in order to compute $\widehat{\boldsymbol{F}}$. And then retrieving the wanted Fundamental matrix using $\boldsymbol{F} = \boldsymbol{T}^{\boldsymbol{T}} \widehat{\boldsymbol{F}} \boldsymbol{T}$

From the derivation explain, we can create a function named ComputeFandEMatrix which take in input minimum 8 matched points to works correctly. It performs the following steps:

- Determining the transformation matrix $\boldsymbol{T}$
- Form the matrix $\boldsymbol{A}$ using the transformed matched point $\boldsymbol{Tp}$
- Compute SVD of $\boldsymbol{A}$, and $\boldsymbol{f} = \boldsymbol{V}[\boldsymbol{8}]$
- Reshape into intermediary fundamental matrix $\boldsymbol{F}$
- Compute SVD of $\boldsymbol{F}$, and $\boldsymbol{F_2} = \boldsymbol{U_F Diag}(\boldsymbol{D_1}, \boldsymbol{D_2}, \boldsymbol{0}) \boldsymbol{V_F^T}$
- Retrieve wanted Fundamental matrix $\boldsymbol{F} = \boldsymbol{T}^{\boldsymbol{T}} \boldsymbol{F_2} \boldsymbol{T}$
- Computing the essential matrix using intrinsic parameters $\boldsymbol{E} = \boldsymbol{K}^{\boldsymbol{T}} \boldsymbol{F} \boldsymbol{K}$

I test with the 2 first images imaging done in Exercise 1, we can find the matrix as the following screenshot figure 6 show:

```
ckesseler@ckesseler-pc:~/PA_3/build$ /home/ckesseler/PA_3/build/PA_epipolar
F:
[2.829035003354215e-08, 8.821579926725391e-06, -0.002009822932090699;
 -8.638472642845142e-06, -2.220705079743235e-08, 0.001678933192721668;
 0.001908980696841524, -0.00211599762707475, 0.110867841962991]
E:
[0.00767621333188472, 2.394078872576084, 0.1052857500640059;
 -2.344385588221836, -0.006027904075505835, -0.591324248670783;
 -0.1244163030824746, 0.3888498642966885, -0.01458137616841611]
```

POSE COMPUTATION

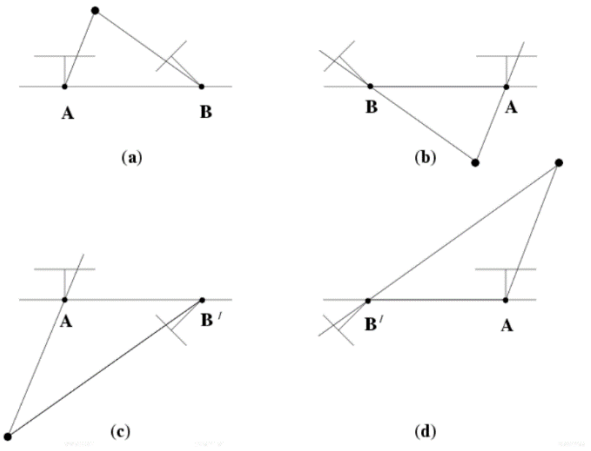
Now we have the Essential matrix E which we know can be written as:

$$\boldsymbol{E = t^{\wedge} R}$$

So, we can retrieve the translation and rotation between two pictures. We start by doing the Singular Value Decomposition of $\boldsymbol{E = U}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}\boldsymbol{V^T}$. The matrix $\boldsymbol{U} = [\boldsymbol{u_1}, \boldsymbol{u_2}, \boldsymbol{u_3}]$ give us the translation information as $\boldsymbol{t = u_3}$ $\boldsymbol{or}$ $\boldsymbol{t = -u_3}$. For the Rotation matrix too, there is 2 possible expression from the SVD:

$$\boldsymbol{R = U}\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}\boldsymbol{V^T}\ \boldsymbol{or}\ \boldsymbol{R = U}\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}\boldsymbol{V^T}$$

So, we have in conclusion 4 possible solution for the Transformation matrix between 2 frames, but only one configuration has the point in front of the 2 cameras (a) as show the figure 7



Maybe the determinant of the Rotation matrix will be -1 , in this case we need to corrected : $\boldsymbol{C \leftarrow -C, R \leftarrow -R}$

To find the good one, we need to use triangulated point computation. First for each set of cameras pose configuration, we compute the triangulate point $\boldsymbol{X}$, using opencv function. Then look at the value of $\boldsymbol{r_3(X - C)}$, if it > 0 then it is a good pose configuration. As there is noise in the correspondence, we choose the one with the

maximum number of points satisfying the cheirality condition. This whole process has been already implement in open cv so we can use the function recoverPose().

I plot the translation matrix in the console as show the figure 8:

```
Pose compute:
  0.829998  -0.556957 -0.0300339  0.0497459
 -0.554344  -0.829664  0.0660379  -0.237183
-0.0616984 -0.0381622  -0.997365    0.97019
         0          0          0          1
Epipolar Constraint sum image1=-1.36437
Pose compute:
  0.692611 -0.0548506  -0.719223  0.0309896
 -0.540968  -0.699047   -0.46764  -0.104753
  -0.47712   0.712969   -0.51384   0.994015
         0          0          0          1
Epipolar Constraint sum image2=18.1604
Pose compute:
-0.914167  0.384225   0.12911  0.132933
 0.404522  0.844614  0.350699 -0.140626
0.0256987  0.372825 -0.927546  0.981098
        0         0         0         1
Epipolar Constraint sum image3=-3.44856
Pose compute:
 0.809405 -0.378062 -0.449369  0.239028
-0.416703 -0.908933 0.0141334 -0.109167
 -0.41379  0.175814 -0.893234  0.964857
        0         0         0         1
Epipolar Constraint sum image4=-1.08781
Pose compute:
   -0.877633     0.479333 -0.000455814     0.334962
    0.473237     0.866624     0.158144    -0.122061
   0.0761984     0.138576    -0.987416     0.934292
           0            0            0            1
Epipolar Constraint sum image5=1.79783
```

EPIPOLAR CONSTRAINTS

The epipolar constraints are defined as:

$$x_2^T t\hat{}Rx_1 = 0$$

In our case we know all information in this equation for the matched point between 2 successive images so $p_1 = p_i$ and $p_2 = p_{i+1}$.

For the rest of the trajectory it is too fastidious to look at one by one matched points so I compute the sum of the error $\sum_{all\_matched} x_{i+1}^T t\hat{}Rx_i$ for every movement between image taken.

```
Epipolar Constraint sum image124=1.59742
Pose compute:
-0.731266  0.276979  0.623324  0.624789
  0.39118 -0.578324  0.715904 -0.774186
 0.558774  0.767349   0.31456 -0.101359
        0         0         0         1
Epipolar Constraint sum image125=498.234
Pose compute:
0.0790474 -0.883711 -0.461309   0.54366
 0.855018 -0.177822  0.487159 -0.154635
-0.512539 -0.432936  0.741532 -0.824938
        0         0         0         1
Epipolar Constraint sum image126=58.0152
Pose compute:
 0.145161 -0.960279 -0.238311  0.237312
  0.97895  0.104471  0.175333 0.0614971
-0.143472 -0.258747  0.955231 -0.969485
        0         0         0         1
Epipolar Constraint sum image127=5.63898
Pose compute:
  0.131061   0.988598 -0.0741463   0.179607
 -0.919444  0.0932418  -0.382006  0.0926967
 -0.370737   0.118239   0.921181  -0.979361
         0          0          0          1
Epipolar Constraint sum image128=-2.67017
Pose compute:
 0.0378067  -0.949802   0.310558  -0.387664
 -0.997918 -0.0196378  0.0614251 -0.0807718
-0.0522431  -0.312234  -0.948568   0.918255
         0          0          0          1
Epipolar Constraint sum image129=-12.6669
Pose compute:
-0.0160393   0.996989 -0.0758687   0.175293
  0.984731  0.0289053   0.171664 -0.0646314
   0.17334 -0.0719569   -0.98223   0.982393
         0          0          0          1
Epipolar Constraint sum image130=-6.61512
```

REFERENCE

Introduction to Visual SLAM From Theory to Practice Xiang Gao and Tao Zhang June 24, 2021

Direct Solution for Estimating the Fundamental and Essential Matrix (Cyrill Stachniss), Youtube video

Course presentation : LN_8_Chap_6_Epipolar Geometry_2022_10_25.pdf

Course lecture slide Berkeley: https://inst.eecs.berkeley.edu/~ee290t/fa19/lectures/lecture10-3-decomposing-F-matrix-into-Rotation-and-Translation.pdf

ANNEXE A: CMAKELISTS

CMakeLists.txt

```
cmake_minimum_required( VERSION 2.8 )
project( PA_3 )

find_package(OpenCV REQUIRED)

include_directories(${OpenCV_INCLUDE_DIRS} )

# Eigen
find_package(Eigen3 3.3 REQUIRED NO_MODULE)

add_executable( PA_feature ORB_feature.cpp )
target_link_libraries(PA_feature ${OpenCV_LIBS})

add_executable(PA_epipolar Epipolar_geometry.cpp)
target_link_libraries(PA_epipolar ${OpenCV_LIBS} Eigen3::Eigen)
```