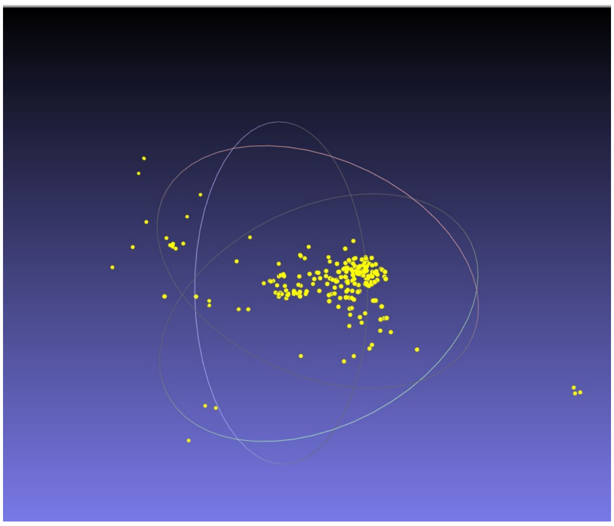
HUYNH Cong Lap – Travail Seul
11419778 – Matching and Reconstruction

In the last TP, ***we get : Distortion Matrix, Parameter intrinsic - Camera matrix (K) and parameters extrinsic (R|t). This information will be used in this TP, to reconstruct the matching point from 2D to 3D***



Before doing this TP, we've already had corrected photo from last TP. This will give us precision.

Process in this TP:
3 main steps

1. Feature detection and description → 2. Matching Feature between 2 images → Find Good Matched → 3. Triangulation between 2 matched points (Hartley and Andrew Zisserman) → Regain the Point in 3D.

1. Feature detection and description

Feature Matching with FLANN

Use the cv::FlannBasedMatcher interface in order to perform a quick and efficient matching by using the Clustering and Search in Multi-Dimensional Spaces module

Use the cv::FeatureDetector interface in order to find interest points. Specifically:

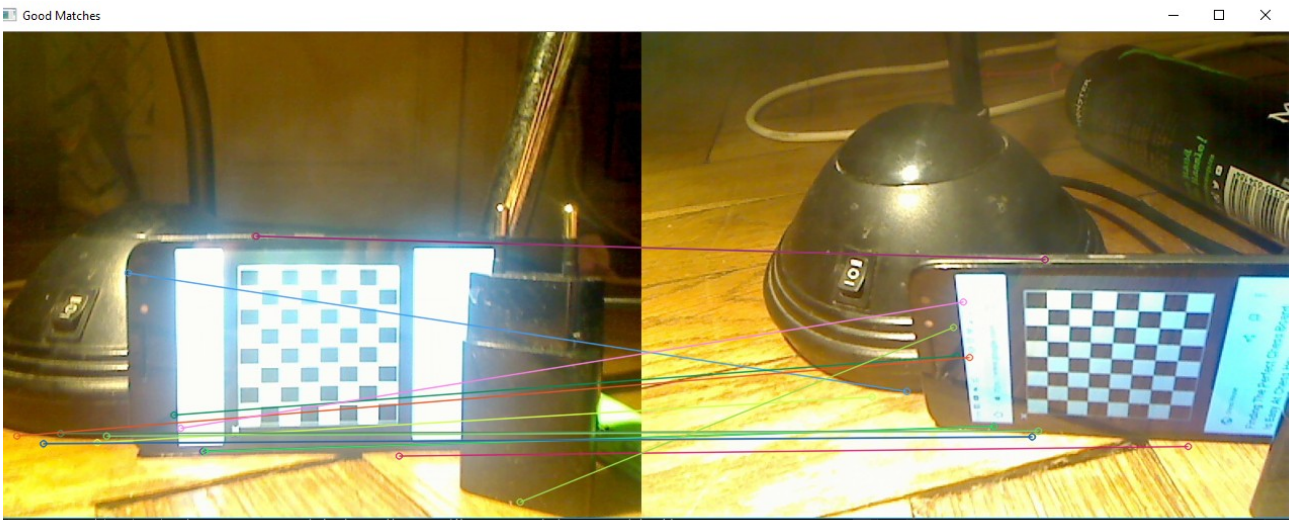
- Use the cv::xfeatures2d::SURF and its function cv::xfeatures2d::SURF::detect to perform the detection process
- Use the function cv::drawKeypoints to draw the detected keypoints

```
//-- Step 1: Detect the keypoints using SURF Detector, compute the descriptors
int minHessian = 400;
Ptr<SURF> detector = SURF::create();
detector->setHessianThreshold(minHessian);
std::vector<KeyPoint> keypoints_1, keypoints_2;
Mat descriptors_1, descriptors_2;
detector->detectAndCompute(img_1, Mat(), keypoints_1, descriptors_1);
detector->detectAndCompute(img_2, Mat(), keypoints_2, descriptors_2);
```

2. Matching Feature between 2 images

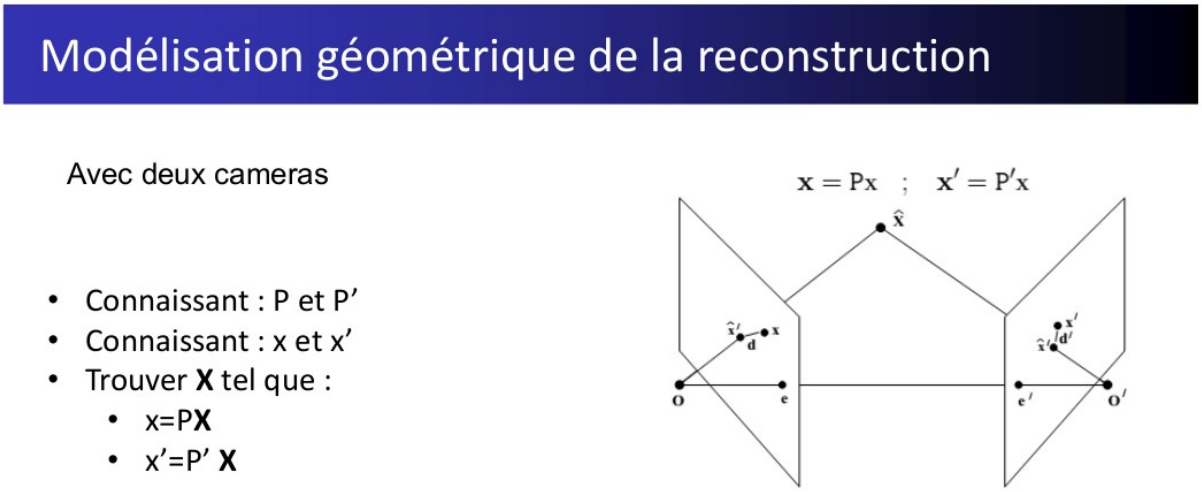
```
//-- Step 2: Matching descriptor vectors using FLANN matcher
FlannBasedMatcher matcher;
std::vector< DMatch > matches;
matcher.match(descriptors_1, descriptors_2, matches);
double max_dist = 0; double min_dist = 100;
//-- Quick calculation of max and min distances between keypoints
for (int i = 0; i < descriptors_1.rows; i++)
{
    double dist = matches[i].distance;
    if (dist < min_dist) min_dist = dist;
    if (dist > max_dist) max_dist = dist;
}
printf("-- Max dist : %f \n", max_dist);
printf("-- Min dist : %f \n", min_dist);
//-- Draw only "good" matches (i.e. whose distance is less than 2*min_dist,
//-- or a small arbitary value ( 0.02 ) in the event that min_dist is very
//-- small)
//-- PS.- radiusMatch can also be used here.
std::vector< DMatch > good_matches;
for (int i = 0; i < descriptors_1.rows; i++)
{

    if (matches[i].distance <= max(2 * min_dist, 0.02))
    {
        good_matches.push_back(matches[i]);
    }
}
//-- Draw only "good" matches
Mat img_matches;
drawMatches(img_1, keypoints_1, img_2, keypoints_2,
    good_matches, img_matches, Scalar::all(-1), Scalar::all(-1),
    vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
```

Result:



3. Triangulation between 2 matched points

Theory:



In the last TP, we had intrinsic (K) of this camera and extrinsic (R|t) of every scene

I read it from output.yml . Then we find the 3x4 Projection Matrix (P).

Simply by: P = K*[R | t]

From that we can have P and P' from the formula above.

The next step is to find x and x'. And we already have them from Step 1 and Step 2 in this TP (Detection and Matching).

Now we can be able to find the real point in 3D. I tried cv::triangulatePoints, but somehow it calculates garbage. I was forced to implement a linear triangulation method manually, which returns a 4x1 matrix for the triangulated 3D point:

```
Mat triangulate_Linear_LS(Mat mat_P_l, Mat mat_P_r, Mat left_pixel, Mat right_pixel)
{
   // resolve equations …
   solve(A,b,X,DECOMP_SVD);
   vconcat(X,W,X_homogeneous);
   return X_homogeneous;
}
```

The input parameters are two 3x4 camera projection matrices and a corresponding Left/Right pixel pair (x,y,w).

I repeat step 1 , 2, 3 with every pair of images.

For exemple : image 1 vs image 2, 3, 4 …

image 2 vs image 3, 4, 5…

image 3 vs image 4, 5, 6...

So the time of processing will be increase very fast when we increase the number of image

After Step 3:

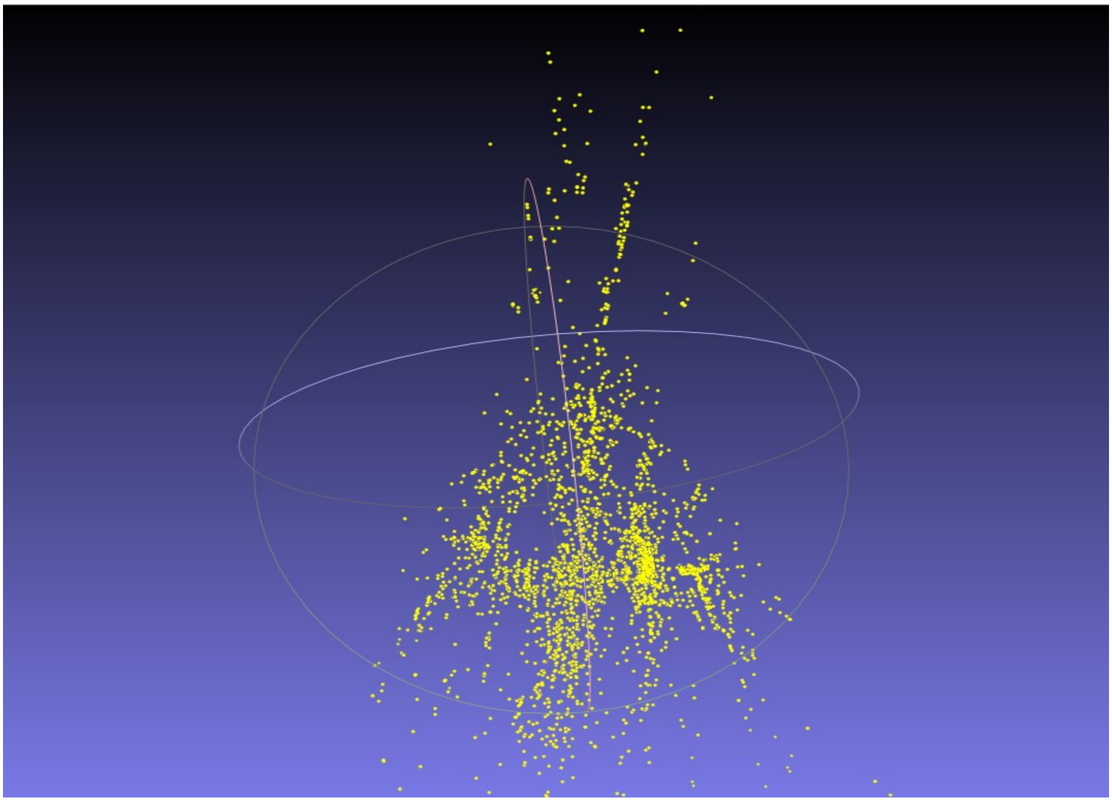
I have a file (cloudpoint.txt) containing 3D coordinates (X,Y,Z) of every pair matched points.

cloudpoint - Notepad

File Edit Format View Help

```
4.0699 2.30154 -10.1286
2.40378 5.64405 4.83641
4.17577 2.59594 -11.4828
4.01704 2.44304 -11.5615
4.57265 4.93976 -1.15586
-3.29952 1.9202 -15.0906
4.50804 2.74857 -12.8135
3.23544 5.08457 1.41272
4.50502 5.04802 0.616849
4.41154 4.03417 0.600475
5.16315 4.39158 0.333646
3.13182 5.20616 3.25812
-3.51312 0.888522 -12.4141
-4.76505 0.749367 -11.428
-5.38878 0.430427 -11.8833
4.39438 3.94838 2.95738
1.76358 5.18063 5.61233
3.88131 3.41821 2.615
4.61892 3.4209 3.03063
-4.68726 0.691153 -12.2219
3.62072 3.57169 3.05481
-8.06602 5.55421 -0.0265562
3.98721 3.98782 3.8065
5.6974 3.23629 -18.0524
-4.68525 0.683328 -12.2047
4.54196 2.8861 1.49186
2.99609 2.45548 -8.6471
6.42401 2.43967 -19.1687
-7.50625 -1.45543 0.509136
5.78164 5.10605 -0.357694
-3.34312 2.00207 -14.6668
4.58923 2.87745 -12.4075
-2.49685 2.87876 -15.9805
-2.45967 2.88359 -15.9994
5.11269 2.94832 -13.5107
```

We can use **MeshLab** to visualize the result:



END