

# Marker Detection (과제 #5.1)

- 목표
  - OpenCV 내 Aruco Library 를 활용한 Marker Detection
  - 탐지된 Aruco 마커의 ID 와 Corner 점 시각화
- 
- 구현 방법

## 1. Aruco Marker Dictionary 가져오기

```
dictionary = cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_250)
parameters = cv2.aruco.DetectorParameters_create()
```

## 2. Aruco Marker Detection 하여 탐지된 마커의 corner 들과 ID 를 찾는다

```
frame = cv2.imread(image_path)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
corners, ids, _ = cv2.aruco.detectMarkers(gray, dictionary,
parameters=parameters)
```

## 3. 시각화 방법

1. 탐지된 마커는 초록색 테두리
2. 마커의 중심점은 빨강색 원
3. 탐지된 ID 시각화

```
if len(corners) > 0:
    ids = ids.flatten()

    for (markerCorner, markerID) in zip(corners, ids):
        corners = markerCorner.reshape((4, 2))
        (topLeft, topRight, bottomRight, bottomLeft) = corners

        topRight = (int(topRight[0]), int(topRight[1]))
        bottomRight = (int(bottomRight[0]), int(bottomRight[1]))
        bottomLeft = (int(bottomLeft[0]), int(bottomLeft[1]))
        topLeft = (int(topLeft[0]), int(topLeft[1]))



        cv2.line(frame, topLeft, topRight, (0, 255, 0), 2)
        cv2.line(frame, topRight, bottomRight, (0, 255, 0), 2)
        cv2.line(frame, bottomRight, bottomLeft, (0, 255, 0), 2)
        cv2.line(frame, bottomLeft, topLeft, (0, 255, 0), 2)

        #centerpoint of Aruco Marker
        cX = int((topLeft[0] + bottomRight[0]) / 2.0)
        cY = int((topLeft[1] + bottomRight[1]) / 2.0)
        cv2.circle(frame, (cX, cY), 4, (0, 0, 255), -1)

        #cv2.putText(image, text, org, font, fontScale, color(BGR)[], thickness[, lineType[, bottomLeftOrigin]])
        cv2.putText(frame, str(markerID), (topLeft[0], topLeft[1] - 15), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 3)
        print("[INFO] Aruco marker ID: {}".format(markerID))

cv2.imshow("Image", frame)
cv2.waitKey(0)
```

#### 4. 결과 예시

<p>원본그림 (예시)</p>	
<p>마커 탐지된 예시</p>	

## 5 Point Algorithm (과제 #5.2)

- 목표
  - 각 영상에서 마커 탐지하여 ID 및 corner 점 찾기
  - 영상이 순차적으로 들어오면, 이전 영상과 현재 영상으로부터 동일한 마커를 인식하였는지 확인
  - 이전 영상과 현재 영상에서 감지된 동일 마커의 특징점(각 코너점 4 개 또는 마커 중심점)을 각각 저장
  - OpenCV Library 를 이용한 5 Point Algorithm 수행하여 회전,이동행렬 연산
- 
- 구현방법
1. 순차적으로 입력되는 영상으로부터 ArUco marker detection 을 반복 수행하여, corners 점들과 ids 들을 계산한다.
    1. 첫 영상이 입력일 경우, 계산한 corner 점들과 ids 를 저장한다.
    2. 다음 영상에 대해 동일한 marker detection 을 수행한다. 만약 이전 영상에서 탐지된 Aruco Marker 가 있고, 현재 영상에서도 탐지된 Marker 가 있다면, 동일 ID 에 대한 코너점들을 저장한다

```
if ids is not None:
    if prev_ids is not None:
        #현재 영상으로부터의 corner들과 ids들 초기화
        updated_corners = []
        updated_ids = []
        #이전 영상으로부터의 corner들과 ids들 초기화
        prev_updated_corners = []
        prev_updated_ids=[]

    for present_id in ids:
        #두 이미지에서 동일 ID를 가진 Aruco marker를 가질 경우, 그 때의 마커들의 정보를 저장
        try:
            #첫번째 코너(좌측 위)의 정보만을 저장할 경우 : [0][0]
            #코너 전체를 저장할 경우 : [0]
            index = np.where(prev_ids == present_id)[0][0]
            updated_corners.append(corners[ids.tolist().index(present_id)][0][0])
            updated_ids.append(present_id)
            prev_updated_corners.append(prev_corners[index][0][0])
            prev_updated_ids.append(prev_ids[index])
        except IndexError:
            pass
```

2. 탐지된 두 영상으로부터 5point Algorithm 을 수행한다.

1. findEssentialMat() : 두 영상으로부터 Essential Matrix 계산하여 반환.

◆ findEssentialMat() [4/6]

```
Mat cv::findEssentialMat ( InputArray  points1,
                          InputArray  points2,
                          InputArray  cameraMatrix,
                          int          method = RANSAC ,
                          double       prob = 0.999 ,
                          double       threshold = 1.0 ,
                          int          maxIters = 1000 ,
                          OutputArray  mask = noArray()
                          )
```

Python:

```
cv.findEssentialMat( points1, points2, cameraMatrix[, method[, prob[, threshold[, maxIters[, mask]]]]) -> retval, mask
cv.findEssentialMat( points1, points2[, focal[, pp[, method[, prob[, threshold[, maxIters[, mask]]]]]]) -> retval, mask
cv.findEssentialMat( points1, points2, cameraMatrix1, distCoeffs1, cameraMatrix2, distCoeffs2[, method[, prob[, threshold[, mask]]]]) -> retval, mask
cv.findEssentialMat( points1, points2, cameraMatrix1, cameraMatrix2, dist_coeff1, dist_coeff2, params[, mask] -> retval, mask
```

```
#include <opencv2/calib3d.hpp>
```

Calculates an essential matrix from the corresponding points in two images.

**Parameters**

<b>points1</b>	Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).
<b>points2</b>	Array of the second image points of the same size and format as points1.
<b>cameraMatrix</b>	Camera intrinsic matrix $A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix. If this assumption does not hold for your use case, use another function overload or <code>undistortPoints</code> with <code>p = cv::noArray()</code> for both cameras to transform image points to normalized image coordinates, which are valid for the identity camera intrinsic matrix. When passing these coordinates, pass the identity matrix for this parameter.
<b>method</b>	Method for computing an essential matrix. <ul style="list-style-type: none"><li>• <b>RANSAC</b> for the RANSAC algorithm.</li><li>• <b>LMEDS</b> for the LMedS algorithm.</li></ul>
<b>prob</b>	Parameter used for the RANSAC or LMedS methods only. It specifies a desirable level of confidence (probability) that the estimated matrix is correct.
<b>threshold</b>	Parameter used for RANSAC. It is the maximum distance from a point to an epipolar line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise.
<b>mask</b>	Output array of N elements, every element of which is set to 0 for outliers and to 1 for the other points. The array is computed only in the RANSAC and LMedS methods.
<b>maxIters</b>	The maximum number of robust method iterations.

This function estimates essential matrix based on the five-point algorithm solver in [208] . [251] is also a related. The epipolar geometry is described by the following equation:

$$[p_2; 1]^T K^{-T} E K^{-1} [p_1; 1] = 0$$

where  $E$  is an essential matrix,  $p_1$  and  $p_2$  are corresponding points in the first and the second images, respectively. The result of this function may be passed further to `decomposeEssentialMat` or `recoverPose` to recover the relative pose between cameras.

2. recoverPose : essential matrix 와 두 영상에서의 매칭쌍으로부터 두 이미지간의 상대적인 회전,이동행렬 관계를 반환.

◆ **recoverPose()** [1/4]

```

int cv::recoverPose ( InputArray      E,
                    InputArray      points1,
                    InputArray      points2,
                    InputArray      cameraMatrix,
                    OutputArray     R,
                    OutputArray     t,
                    double           distanceThresh,
                    InputOutputArray mask = noArray() ,
                    OutputArray     triangulatedPoints = noArray()
                  )

```

Python:

```

cv.recoverPose( points1, points2, cameraMatrix1, distCoeffs1, cameraMatrix2, distCoeffs2[, E[, R[, t[, method[, prob[, threshold[, mask]]]]]] ) -> (retval, E, R, t, mask)
cv.recoverPose( E, points1, points2, cameraMatrix[, R[, t[, mask]]]) -> (retval, R, t, mask)
cv.recoverPose( E, points1, points2[, R[, t[, focal[, pp[, mask]]]]]) -> (retval, R, t, mask)
cv.recoverPose( E, points1, points2, cameraMatrix, distanceThresh[, R[, t[, mask[, triangulatedPoints]]]) -> (retval, R, t, mask, triangulatedPoints)

```

#include <opencv2/calib3d.hpp>

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**Parameters**

- E** The input essential matrix.
- points1** Array of N 2D points from the first image. The point coordinates should be floating-point (single or double precision).
- points2** Array of the second image points of the same size and format as points1.
- cameraMatrix** Camera intrinsic matrix  $A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix.
- R** Output rotation matrix. Together with the translation vector, this matrix makes up a tuple that performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Note that, in general, t can not be used for this tuple, see the parameter description below.
- t** Output translation vector. This vector is obtained by `decomposeEssentialMat` and therefore is only known up to scale, i.e. t is the direction of the translation vector and has unit length.
- distanceThresh** threshold distance which is used to filter out far away points (i.e. infinite points).
- mask** Input/output mask for inliers in points1 and points2. If it is not empty, then it marks inliers in points1 and points2 for the given essential matrix E. Only these inliers will be used to recover pose. In the output mask only inliers which pass the chirality check.
- triangulatedPoints** 3D points which were reconstructed by triangulation.

This function differs from the one above that it outputs the triangulated 3D point that are used for the chirality check.

### 3. 구현 코드

```

updated_corners_np = np.array(updated_corners)
prev_corners_np = np.array(prev_updated_corners)
updated_corners_flat = updated_corners_np.reshape(-1, 1, 2)
prev_corners_flat = prev_corners_np.reshape(-1, 1, 2)

E, _ = cv2.findEssentialMat(prev_corners_flat, updated_corners_flat, K,
                             method=cv2.RANSAC, prob=0.999, threshold=1.0)

_, R, t, _ = cv2.recoverPose(E, prev_corners_flat, updated_corners_flat, K, distanceThresh=99999)

```

4. 위 코드에서 계산한 R,t는 각각 두 영상으로부터 계산한 회전행렬과 이동행렬이다. Visualization은 다음과 같이 알 수 있다.

```

print("Rotation Matrix:")
print(R)
print("Translation Vector:")
print(t.T)

```



# SFM using VisualSFM (과제 #5.4)

- 목표
- 순차적인 영상들로부터 영상간 매칭쌍을 구하고 이들 관계성을 이용한 3 차원 복원
- VisualSFM 프로그램 사용

## 1. VisualSFM 설치 및 실행방법

### 1. 설치 경로 ( <http://ccwu.me/vsfm/> )

Download **v0.5.26** ([changelog](#) with new feature documentation)

Windows\* ([64-bit](#), [32-bit](#), [installation guide](#)), \*for nVidia CUDA or [CUDA Simulation](#).

Windows ([64-bit](#), [32-bit](#), [installation guide](#))

Linux ([64-bit](#), [32-bit](#), [installation guide](#)), see the tutorials for [Ubuntu](#) or [Fedora](#).

Mac OSX ([64-bit](#), [32-bit](#), [installation guide](#)), see the installer by [Dan Monaghan](#).

\* VisualSFM is free for personal, non-profit or academic use. See [README](#) for more details.

\* Please cite VisualSFM according to [README](#) in your publication.

-CUDA 를 사용하지 않기 때문에 두번째 설치 경로를 따라 설치하면 됨.

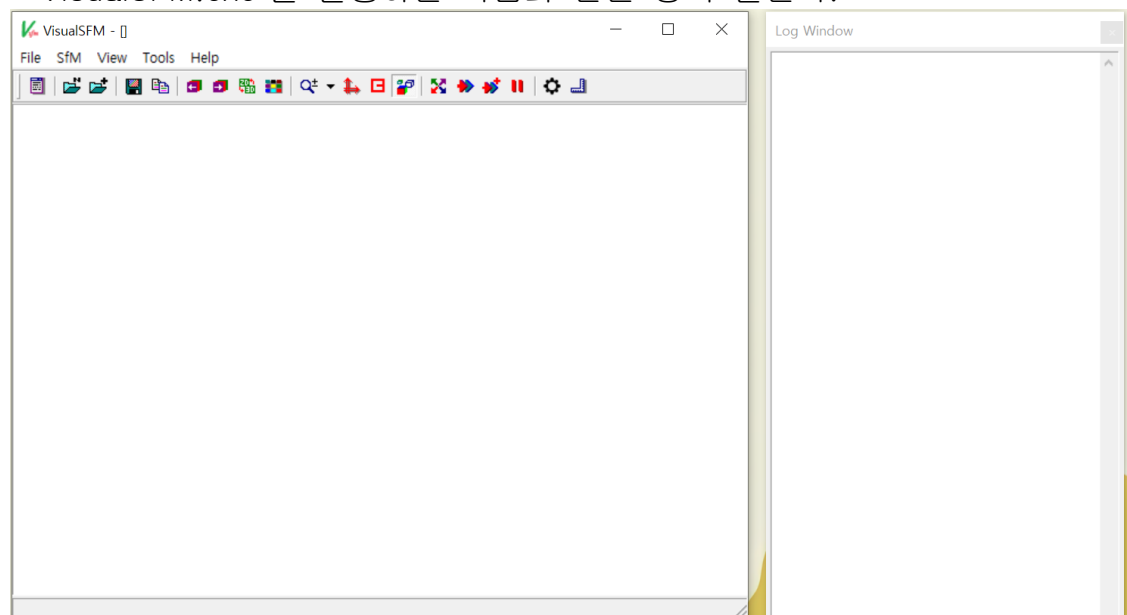
-간단한 설치 후, 해당 경로에 다음과 같은 폴더가 생성됨

VisualSFM\_windows\_64bit 2024-04-02 오후 8:06 파일 폴더

-폴더 내부는 다음과 같이 구성되어 있음

log	2024-05-02 오후 9:04	파일 폴더	
glew64.dll	2024-04-02 오후 8:03	응용 프로그램 확장	273KB
nv.ini	2024-04-30 오후 10:19	구성 설정	7KB
pba_x64.dll	2024-04-02 오후 8:03	응용 프로그램 확장	253KB
README.txt	2024-04-02 오후 8:03	텍스트 문서	3KB
SiftGPU64.dll	2024-04-02 오후 8:03	응용 프로그램 확장	424KB
VisualSFM.exe	2024-04-02 오후 8:03	응용 프로그램	1,353KB

-‘VisualSFM.exe’를 실행하면 다음과 같은 창이 열린다.





-자동생성코드 예시(마커당 코너 4 개를 특징점으로 사용할 경우)

1. python 에서 경로를 받기 위한 수정

```
directory_path = os.path.dirname(image_path)
directory_path = directory_path.replace('/', '\\')
_filename = os.path.basename(image_path)
```

2. SIFT 파일 자동생성

```
filename = os.path.splitext(_filename)[0]
sift_filename = f"{filename}.sift"
sift_file = os.path.join(directory_path,sift_filename)
with open(sift_file, "w") as output:
    output.write(f"{len(corners) * 4} 128\n")
    for corner_array in corners:
        for corner in corner_array:
            for point in corner:
                x, y = point[0], point[1]
                output.write(f"{x} {y}"+" 0 0\n")
                for i in range(0,6):
                    output.write (" 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0")
                output.write (" 0 0 0 0 0 0 0 0\n")
```

3.(선택)생성된 SIFT 파일 내용 확인을 위한 text 파일 자동생성

```
sift_filename = f"{filename}.txt"
sift_file = os.path.join(directory_path,sift_filename)
with open(sift_file, "w") as output:
    output.write(f"{len(corners) * 4} 128\n")
    for corner_array in corners:
        for corner in corner_array:
            for point in corner:
                x, y = point[0], point[1]
                output.write(f"{x} {y}"+" 0 0\n")
                for i in range(0,6):
                    output.write (" 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0")
                output.write (" 0 0 0 0 0 0 0 0\n")
```



## 2. 두 영상간의 매칭을 지정하기 위한 matchFile(.txt 파일)

-matchFile.txt Format

<1번 이미지의 절대경로>

<2번 이미지의 절대경로>

<매칭되는 Feature 개수>

<1번 이미지의 Matching 점 나열>

<2번 이미지의 Matching 점 나열>

-예시

C:\wsfm\trial2\3.jpg

1번 이미지의 절대경로

C:\wsfm\trial2\4.jpg

2번 이미지의 절대경로

10

두 영상에서의 매칭쌍 개수

0 1 2 3 4 5 6 7 8 9

1번 이미지의 Matching 점 나열

0 1 2 3 4 5 6 7 8 9

2번 이미지의 Matching 점 나열

\*i번 이미지의 Matching 점을 나열할 때 0번부터 시작하여 작성해야한다  
⇒ 0 1 2 3 4 5 6 .... <매칭되는 Feature 개수>-1

-자동생성코드 예시(마커당 코너 4 개를 특징점으로 사용할 경우)

```
array1_flat = prev_ids.T[0]
array2_flat = ids.T[0]

common_elements = [value for value in array2_flat if value in array1_flat]
common_indices_array1 = [np.where(array1_flat == value)[0][0] for value in common_elements]
common_indices_array2 = [np.where(array2_flat == value)[0][0] for value in common_elements]

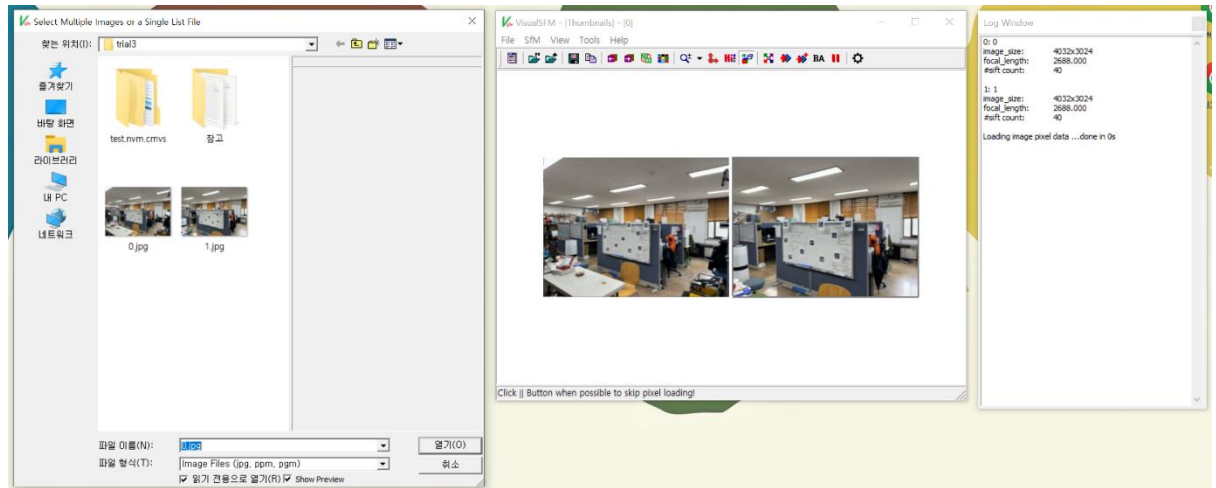
output_array1 = []
output_array2 = []
for idx in common_indices_array1:
    output_array1.extend(range(idx*4, (idx+1)*4))
for idx in common_indices_array2:
    output_array2.extend(range(idx*4, (idx+1)*4))
output_string1 = ' '.join(map(str, output_array1))
output_string2 = ' '.join(map(str, output_array2))

# <.mat> file Generation
print(os.path.splitext(prev_filename)[0])
print(os.path.splitext(_filename)[0])
output_filename = f"matchFile_{os.path.splitext(prev_filename)[0]}{os.path.splitext(_filename)[0]}.txt"
output_file = os.path.join(directory_path, output_filename)
with open(image_path, "r") as image_file, open(output_file, "w") as output:
    output.write(f"{os.path.join(directory_path, prev_filename)}\n")
    output.write(f"{os.path.join(directory_path, _filename)}\n")
    output.write(f"{len(common_elements)*4}\n")
    output.write(output_string1 + "\n")
    output.write(output_string2 + "\n")
```

\* SIFT 파일 생성 시, Aruco Marker 에 따라 정렬되도록 함.

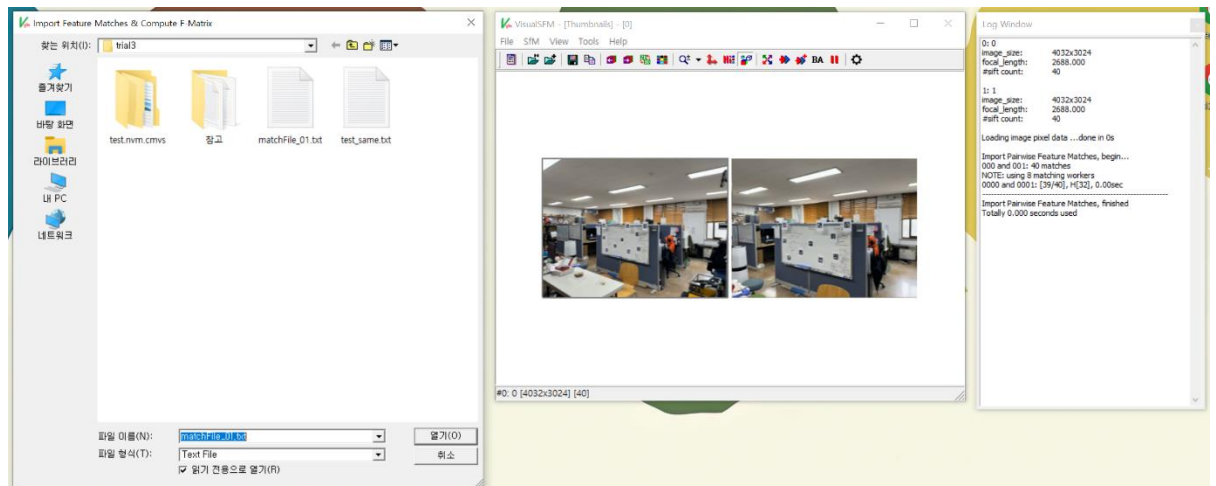
### 3. VisualSFM 사용 방법

1. SFM/More Functions/Set Fixed Calibration 에서 카메라 파라미터 입력
2. File/Open Multi Images



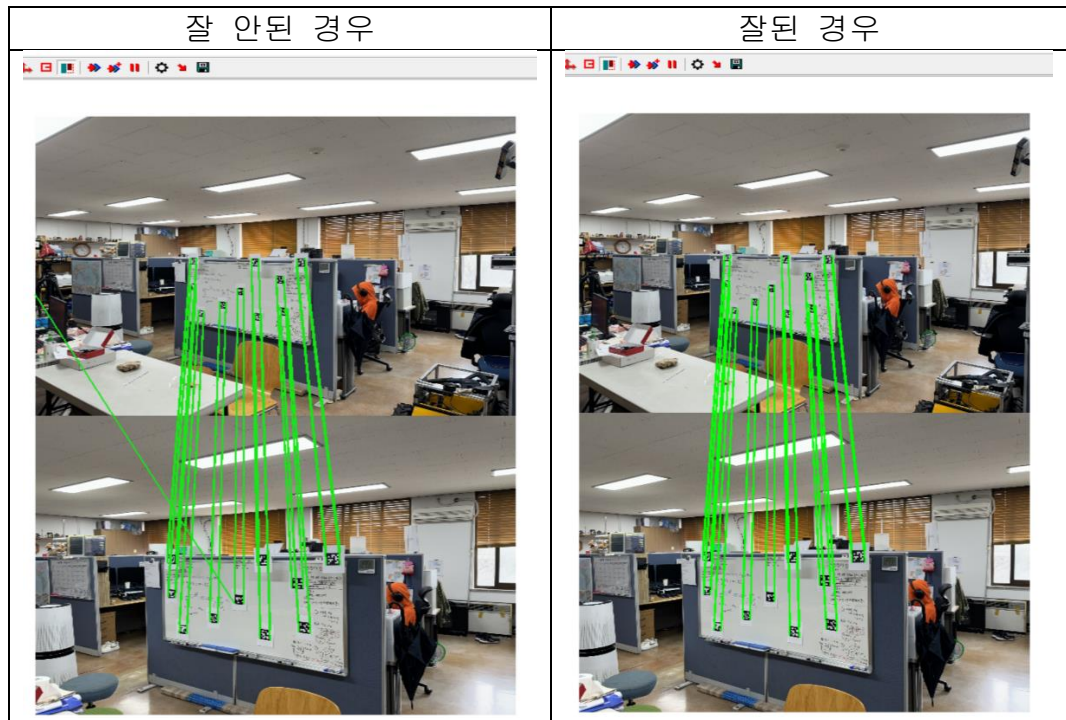
- 같은 파일 경로에 이미지 파일(예 0.jpg)과 같은 이름을 가진 SIFT 파일(예 0.sift)이 있음.
- SFM 에 이용할 파일들을 추가한다.(이미지만 추가하면 SIFT 파일도 자동으로 들어감)
- Log Window 에서 #sift count 가 지정한 갯수만큼 출력됨을 알 수 있다.

### 3. SfM/Pairwise Matching/Import Feature Matches 에서 Matches.txt import

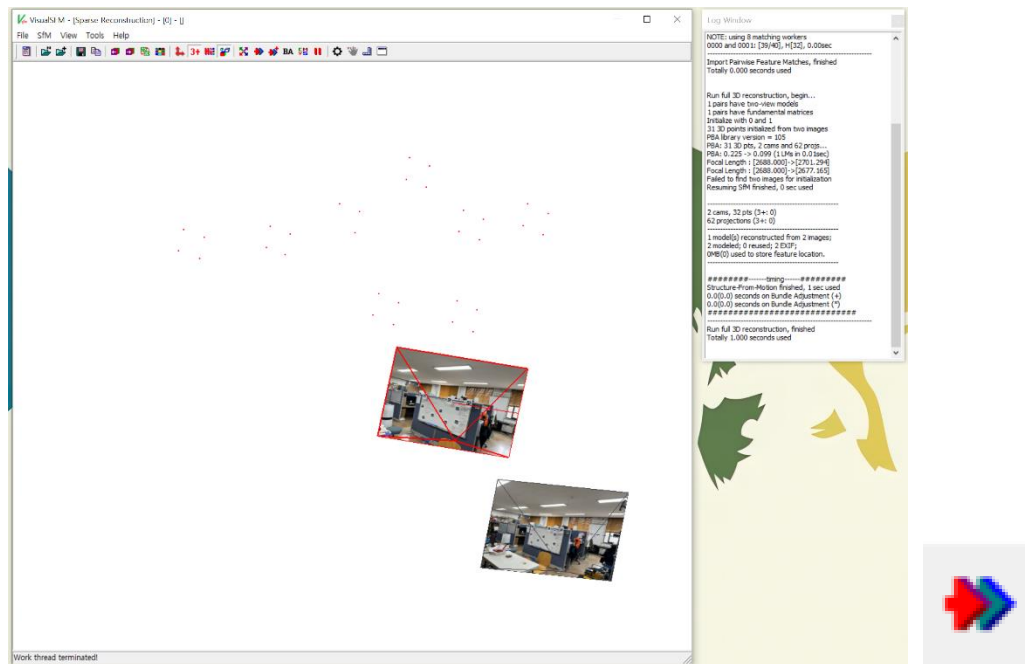


### 4. Match 적용여부 확인 방법

- 매칭파일을 잘 작성했으면, 깨끗한 매칭선을 가짐을 알 수 있을 것이다.



## 5. 3D Reconstruction(3 차원 포인트 복원)



- Compute 3D reconstruction 아이콘 또는 메뉴→SfM→Reconstruct Sparse 시, sparse 3D reconstruction 실행됨. 위의 그림은 reconstruction 된 후의 사진.
- SfM/save Nview match 를 통해 nvm 파일 생성할 수 있음.(확장자명만 txt 로 바꾸면 안의 내용을 볼 수 있다.)

-nvm 파일은 각 이미지를 찍은 카메라 파라미터와 복원된 특징점 3 차원 포인트들에 대한 정보를 가지고 있다.

-각 3 차원 특징점 위치는 앞에서부터 3 개 순서로  $x,y,z$  를 지칭한다.(자세한 설명은 밑의 그림과 함께 보면 된다.)

\*nvm 파일 : sfm 에 사용된 사진들과 카메라 파라미터, 복원된 3 차원 포인트 총 개수에 대한 정보를 가지고 있음.

\*예시 파일 및 설명

```
18
06.jpg 2889 0.99981107836 -0.00209734069348 -0.0164729660934 -0.0101250353875 0.108823260208 0.00320318627696 -0.0354217440756 0 0
07.jpg 2889 0.999568631877 0.00603584603037 -0.0279061724333 -0.006883301884 1.08323866467 -0.0578990597994 0.0615697842352 0 0
05.jpg 2889 0.998529983563 -0.0198238200429 -0.0502867243454 0.00406218648112 -0.986589131843 5.55111512313e-017 0.146725091317 0 0
04.jpg 2889 0.997985794636 -0.0208289519713 -0.0599227200319 0.000174376527712 -1.80459948453 0.0218541415325 0.192654656934 0 0
03.jpg 2889 0.992786865607 -0.0137584253507 -0.119015417363 0.00453572148733 -2.84076607033 0.0536711768324 0.292447014116 0 0
02.jpg 2889 0.978552016855 -0.0210436297318 -0.204886431552 0.00385044927054 -3.70864035716 -0.124182679929 0.746913092629 0 0
01.jpg 2889 0.975277847396 -0.0347248031131 -0.218082309287 0.00822941039278 -4.04045807468 2.77555756156e-017 1.45313353407 0 0
00.jpg 2889 0.956744308715 -0.0609644364422 -0.282213788345 0.0357627191505 -4.36717224513 0.199307924298 2.66633642136 0 0
2889 0.999801496688 0.018708734096 0.000128419193501 -0.00686270640424 2.29337901628 0.029467780706 -0.124079251872 0 0
09.jpg 2889 0.999580696141 0.0269494689402 -0.0104316497305 -0.00189280883479 3.05313778393 0.0155616729566 -0.132982480927 0 0
10.jpg 2889 0.999176043267 0.0300238116532 -0.0268493440881 -0.00499200330446 4.22966437255 -0.0370188659896 -0.416014876496 0 0
11.jpg 2889 0.99964749651 0.0232551105076 0.0115889442567 -0.00548417526202 5.32376997432 0.0939014144363 -0.189395468836 0 0
12.jpg 2889 0.998978139534 0.0187658443531 0.0393789871076 -0.0118251961245 6.65398837853 0.0821978648126 -0.101015034284 0 0
13.jpg 2889 0.997315654668 0.00925047919073 0.0701567401972 -0.0188122939236 8.04075430023 -0.000449847653023 -0.252643021794 0 0
14.jpg 2889 0.989615148291 0.0207456808843 0.140253651089 -0.0236723704326 9.25642141927 2.77555756156e-017 -0.457952170399 0 0
15.jpg 2889 0.987193020355 0.0181831569026 0.156547279498 -0.024743699612 10.1156675318 0.00045487851543 -0.62928424207 0 0
16.jpg 2889 0.972365322438 0.0222134833819 0.228787300538 -0.0408490508013 11.0241747379 0.0264296609322 -0.517974614543 0 0
(상) 17.jpg 2889 0.936561026113 0.0286532567999 0.346265245338 -0.0461824380325 12.0927317553 0.216691353221 0.383849574987 0 0
```

```
179
8.98694051733 0.966124191807 4.61831649538 152 163 183 5 13 8 1017.5 511.269714355 14 8 691.5 424.971008301 15 8 321.5 420.957092285 16 8 283.5 395.870269775 17 4 258.5 323.620178223
10.537851596 0.969875443792 3.72900878234 153 166 188 4 14 18 1906.5 557.429504395 15 18 1285.5 543.380859375 16 22 1097.5 509.262756348 17 22 862.5 459.089080811
5.82974855694 1.44864636775 4.85203898499 172 183 202 5 13 2 -760.5 761.13458252 14 2 -824.5 630.683044434 15 2 -1012.5 612.620544434 16 2 -859.5 583.519775391 17 2 -660.5 484.175933838
7.57341774734 0.709072644407 4.82146518344 145 152 161 7 11 17 1376.5 209.224212646 12 17 783.5 250.366622925 13 17 152.5 347.703552246 14 21 -61.5 271.439575195 15 21 -332.5 279.467346191 16 25 -
-1.30591365823 0.851193415548 5.30012028799 133 137 140 9 0 8 -862.5 495.214141846 1 8 -1507.5 498.22454834 2 0 -479.5 599.575378418 3 0 -64.5 509.540649414 4 0 167.5 524.314880371 5 0 215.5 675.835
-1.02581351606 0.859603319084 5.29504794548 137 145 151 9 0 9 -705.5 495.214141846 1 9 -1347.5 500.231506348 2 1 -318.5 602.58581543 3 1 91.5 591.547607422 4 1 314.5 525.318359375 5 1 353.5 668.815
-1.03490611075 1.13865151545 5.24259577889 139 148 156 9 0 10 -715.5 654.766418457 1 10 -1362.5 662.794189453 2 2 -327.5 771.169311523 3 3 92.5 758.124145508 4 2 320.5 684.870605469 5 2 366.5 834.31
-1.31677188585 1.12995799862 5.24584851679 132 141 144 8 0 11 -873.5 653.762939453 1 11 -1523.5 660.787231445 2 3 -491.5 769.16233516 3 3 -67.5 757.120727539 4 3 170.5 685.874084473 5 3 225.5 844.4
-0.789055314014 0.642768773313 5.33384472492 130 135 138 9 0 16 -572.5 369.779968262 1 16 -1205.5 372.790374756 2 8 -182.5 474.141204834 3 8 217.5 464.10647583 4 8 427.5 401.891113281 5 8 452.5 537
-0.509140226103 0.642138038186 5.32480310479 124 129 133 9 0 17 -420.5 364.76260376 1 17 -1047.5 369.779968262 2 9 -26.5 473.13772583 3 9 371.5 462.099517822 4 9 568.5 398.8067627 5 9 581.5 529.33
-0.509445992179 0.921169261915 5.27232545567 138 145 152 9 0 18 -421.5 521.304443359 1 18 -1055.5 530.335693359 2 10 -25.5 635.700439453 3 10 380.5 622.655273438 4 10 582.5 552.412109375 5 10 600.5
-0.789917059927 0.919542008113 5.27598604461 138 147 150 9 0 19 -575.5 525.318359375 1 19 -1215.5 533.346130371 2 11 -184.5 636.7039918457 3 11 224.5 626.662231445 4 11 439.5 525.445246387 5 11 471
10.2587586763 0.686288214489 3.79517174412 127 138 156 4 14 16 1641.5 332.651428223 15 16 1062.5 335.661865234 16 20 882.5 303.550720215 17 20 660.5 226.28326416
0.709999412275 -0.990733349413 4.39404219665 121 108 98 2 5 49 1450.5 -364.76260376 6 45 1855.5 -396.873748779
1.55238559699 0.7455613242 4.54148459676 171 170 177 2 4 19 1830.5 481.165496826 5 23 1718.5 575.492004395
10.5422448818 0.688732719437 3.7751019537 110 120 132 4 14 17 1890.5 334.65838623 15 17 1277.5 337.668823242 16 21 1088.5 303.550720215 17 21 858.5 227.286727905
0.103670931941 0.4995954527326 5.3287663744 132 136 137 10 0 24 -92.5 279.467346191 2 14 -699.5 281.474304199 2 16 308.5 391.856384277 3 16 697.5 379.814697266 4 12 863.5 319.606292725 5 12 839.5 4
0.385460749354 0.501638175439 5.32122481078 129 133 135 10 0 25 57.5 277.460388184 1 25 -542.5 279.467346191 2 17 461.5 393.863311768 3 17 848.5 379.814697266 4 13 997.5 319.606292725 5 13 954.5 4
0.383055023368 0.780508433754 5.27096893395 139 144 148 10 0 26 60.5 430.991851807 1 26 -547.5 437.012664795 2 18 467.5 553.415588379 3 18 862.5 537.360046387 4 14 1013.5 466.11340332 5 14 974.5 5
0.100836128463 0.777633671503 5.27904016084 127 132 137 10 0 27 -90.5 432.988779297 1 27 -705.5 439.019622803 2 19 311.5 550.405151367 3 19 709.5 537.360046387 4 15 877.5 468.120361328 5 15 858.5
6.49202827528 0.49813349486 4.97270415167 136 140 144 8 10 29 1032.5 97.8386611938 11 25 722.5 84.7935028076 12 25 138.5 127.942863464 13 25 -416.5 236.317993164 14 29 -539.5 169.085266113 15 29 -
-1.49106264589 0.425753261698 5.38856612195 118 123 125 8 0 29 -955.5 260.401367188 1 29 -1594.5 256.387451172 2 21 -577.5 352.720916748 3 21 -172.5 345.696594238 4 21 57.5 287.495147705 5 17 98.5
```

→(상): sfm 에 사용된 사진들 이름, 각 사진의 카메라 파라미터

(하): 복원된 총 개수, 각 특징점의 정보들(위치, 색깔, Scalar 등등)

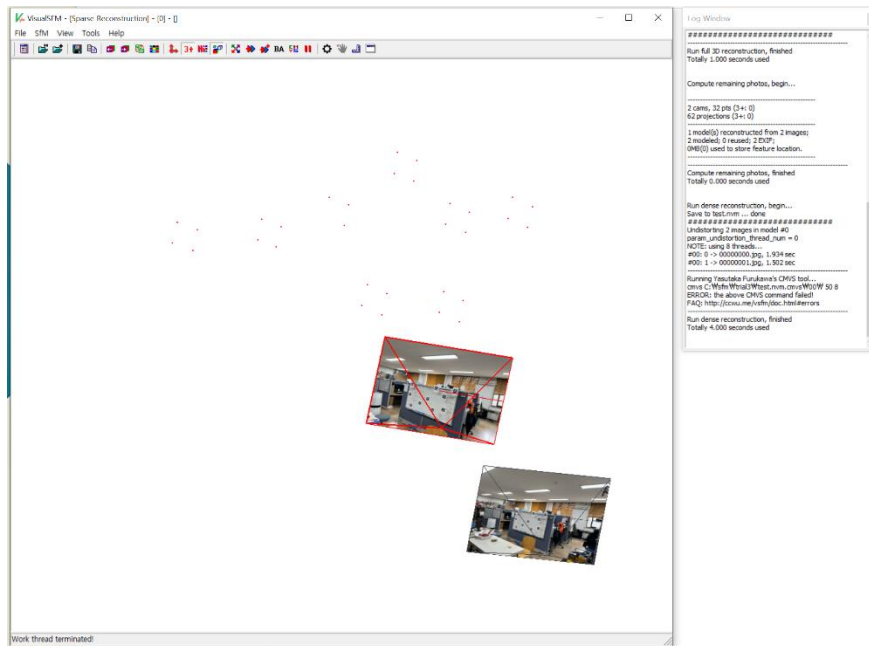
: 각 줄은 복원한 각 특징점들에 대한 정보를 가지고 있고, 특징점 위치( $x,y,z$ )는 앞에서부터 3 개의 숫자들을 순서대로 지칭한다.

```
179
8.98694051733 0.966124191807 4.61831649538 152 163 183 5 13 8 1017.5 511.269714355 14 8 691.5 424.971008301 15 8 321.5 420.957092285 16 8 283.5 395.870269775 17 4 258.5 323.620178223
10.537851596 0.969875443792 3.72900878234 153 166 188 4 14 18 1906.5 557.429504395 15 18 1285.5 543.380859375 16 22 1097.5 509.262756348 17 22 862.5 459.089080811
5.82974855694 1.44864636775 4.85203898499 172 183 202 5 13 2 -760.5 761.13458252 14 2 -824.5 630.683044434 15 2 -1012.5 612.620544434 16 2 -859.5 583.519775391 17 2 -660.5 484.175933838
7.57341774734 0.709072644407 4.82146518344 145 152 161 7 11 17 1376.5 209.224212646 12 17 783.5 250.366622925 13 17 152.5 347.703552246 14 21 -61.5 271.439575195 15 21 -332.5 279.467346191 16 25 -
-1.30591365823 0.851193415548 5.30012028799 133 137 140 9 0 8 -862.5 495.214141846 1 8 -1507.5 498.22454834 2 0 -479.5 599.575378418 3 0 -64.5 509.540649414 4 0 167.5 524.314880371 5 0 215.5 675.835
-1.02581351606 0.859603319084 5.29504794548 137 145 151 9 0 9 -705.5 495.214141846 1 9 -1347.5 500.231506348 2 1 -318.5 602.58581543 3 1 91.5 591.547607422 4 1 314.5 525.318359375 5 1 353.5 668.815
-1.03490611075 1.13865151545 5.24259577889 139 148 156 9 0 10 -715.5 654.766418457 1 10 -1362.5 662.794189453 2 2 -327.5 771.169311523 3 3 92.5 758.124145508 4 2 320.5 684.870605469 5 2 366.5 834.31
-1.31677188585 1.12995799862 5.24584851679 132 141 144 8 0 11 -873.5 653.762939453 1 11 -1523.5 660.787231445 2 3 -491.5 769.16233516 3 3 -67.5 757.120727539 4 3 170.5 685.874084473 5 3 225.5 844.4
-0.789055314014 0.642768773313 5.33384472492 130 135 138 9 0 16 -572.5 369.779968262 1 16 -1205.5 372.790374756 2 8 -182.5 474.141204834 3 8 217.5 464.10647583 4 8 427.5 401.891113281 5 8 452.5 537
-0.509140226103 0.642138038186 5.32480310479 124 129 133 9 0 17 -420.5 364.76260376 1 17 -1047.5 369.779968262 2 9 -26.5 473.13772583 3 9 371.5 462.099517822 4 9 568.5 398.8067627 5 9 581.5 529.33
-0.509445992179 0.921169261915 5.27232545567 138 145 152 9 0 18 -421.5 521.304443359 1 18 -1055.5 530.335693359 2 10 -25.5 635.700439453 3 10 380.5 622.655273438 4 10 582.5 552.412109375 5 10 600.5
-0.789917059927 0.919542008113 5.27598604461 138 147 150 9 0 19 -575.5 525.318359375 1 19 -1215.5 533.346130371 2 11 -184.5 636.7039918457 3 11 224.5 626.662231445 4 11 439.5 525.445246387 5 11 471
10.2587586763 0.686288214489 3.79517174412 127 138 156 4 14 16 1641.5 332.651428223 15 16 1062.5 335.661865234 16 20 882.5 303.550720215 17 20 660.5 226.28326416
0.709999412275 -0.990733349413 4.39404219665 121 108 98 2 5 49 1450.5 -364.76260376 6 45 1855.5 -396.873748779
1.55238559699 0.7455613242 4.54148459676 171 170 177 2 4 19 1830.5 481.165496826 5 23 1718.5 575.492004395
10.5422448818 0.688732719437 3.7751019537 110 120 132 4 14 17 1890.5 334.65838623 15 17 1277.5 337.668823242 16 21 1088.5 303.550720215 17 21 858.5 227.286727905
0.103670931941 0.4995954527326 5.3287663744 132 136 137 10 0 24 -92.5 279.467346191 2 14 -699.5 281.474304199 2 16 308.5 391.856384277 3 16 697.5 379.814697266 4 12 863.5 319.606292725 5 12 839.5 4
0.385460749354 0.501638175439 5.32122481078 129 133 135 10 0 25 57.5 277.460388184 1 25 -542.5 279.467346191 2 17 461.5 393.863311768 3 17 848.5 379.814697266 4 13 997.5 319.606292725 5 13 954.5 4
0.383055023368 0.780508433754 5.27096893395 139 144 148 10 0 26 60.5 430.991851807 1 26 -547.5 437.012664795 2 18 467.5 553.415588379 3 18 862.5 537.360046387 4 14 1013.5 466.11340332 5 14 974.5 5
0.100836128463 0.777633671503 5.27904016084 127 132 137 10 0 27 -90.5 432.988779297 1 27 -705.5 439.019622803 2 19 311.5 550.405151367 3 19 709.5 537.360046387 4 15 877.5 468.120361328 5 15 858.5
6.49202827528 0.49813349486 4.97270415167 136 140 144 8 10 29 1032.5 97.8386611938 11 25 722.5 84.7935028076 12 25 138.5 127.942863464 13 25 -416.5 236.317993164 14 29 -539.5 169.085266113 15 29 -
-1.49106264589 0.425753261698 5.38856612195 118 123 125 8 0 29 -955.5 260.401367188 1 29 -1594.5 256.387451172 2 21 -577.5 352.720916748 3 21 -172.5 345.696594238 4 21 57.5 287.495147705 5 17 98.5
```

[Coordinate X,Y,Z]



## 6. 카메라 포즈 계산(회전행렬, 이동행렬)



-카메라의 R/T 를 알려면 dense reconstruction 을 해야함.

( 메뉴→SfM→Reconstruct Dense )

-새 창이 생성되는데, 5 번 과정에서 저장한 test.nvm 을 넣으면 진행이 된다.  
단, Dense reconstruction 과정이 완료되지 않고 Log Window 에 따르면 Fail 함.

-해당 파일 경로에 <파일이름.nvm>.cmvs 폴더 생성됨을 확인. 생성된 폴더  
내 cameras\_v2.txt 에 각 카메라에 대한 정보를 가지고 있음.

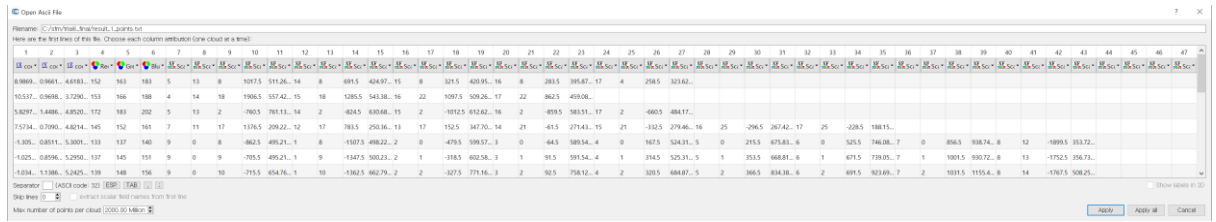
-카메라 포즈는 'Camera Pose Visualization Code'를 통해 시각화할 수  
있음.(관련 코드 및 사용법 첨부함)

-자세한 내용은 'Camera Pose Visualization Code 사용법' 문서 참고하여  
카메라 포즈 시각화한다.

## 7. CloudCompare 로 시각화하여 결과 확인해보기

-5 번 결과의 nvm 파일에서 3 차원 특징점의 위치 X,Y,Z 부분만 발췌해서  
다른 파일(txt 파일)로 생성 → text 파일을 'CloudCompare'에서 시각화하기  
위함

-3 차원 특징점 text 파일을 CloudCompare 에 넣을 때, 세로줄 1~6 줄을  
제외하고 IGNORE 해야, 전체 특징점을 볼 수 있음.

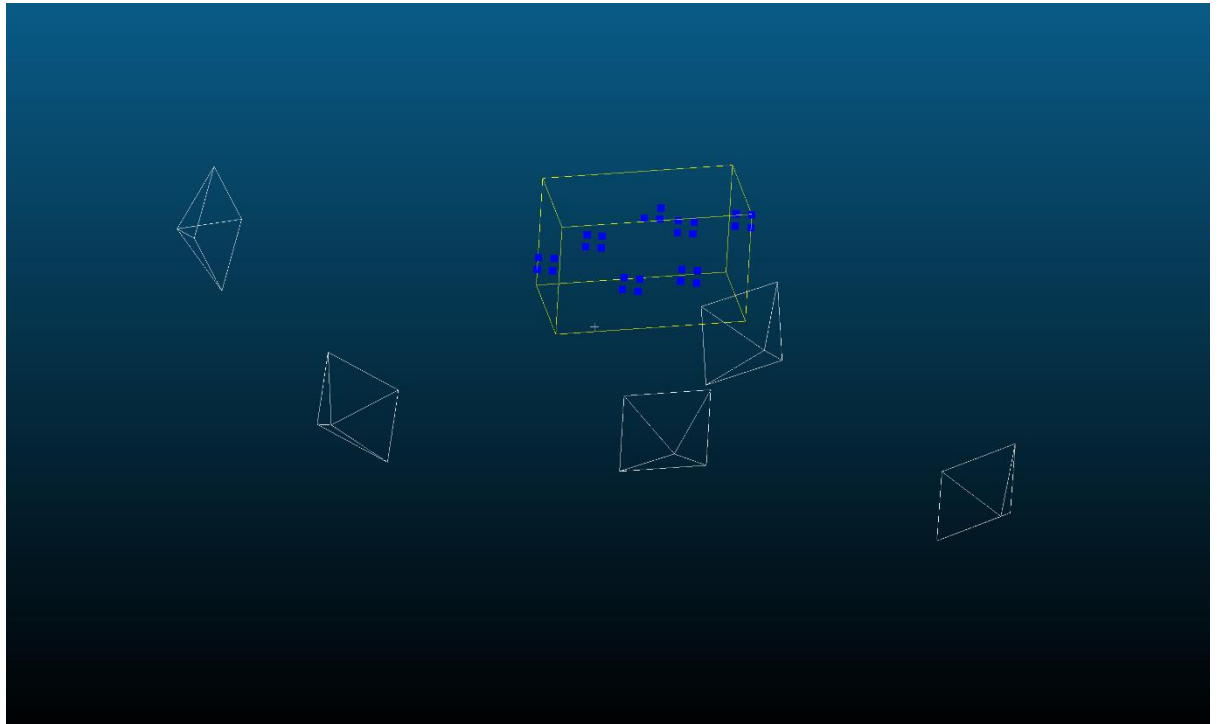


22	23	24	25	26	27
Scα	Scα	Scα	Scα	Scα	Scα
283.5	395.87...	Ignore	4	258.5	323.62...
		c...X			
862.5	459.08...	c...Y			
		c...Z			
-859.5	583.51...	Nx	2	-660.5	484.17...
		Ny			
-61.5	271.43...	Nz	21	-332.5	279.46...
		R...)			
-64.5	589.54...	G...)	0	167.5	524.31...
		B...)			
91.5	591.54...	R...)	1	314.5	525.31...
		G...)			
92.5	758.12...	B...)	2	320.5	684.87...
		G...y			
		R...i			
		R...f			
		L...l			
		SF c			

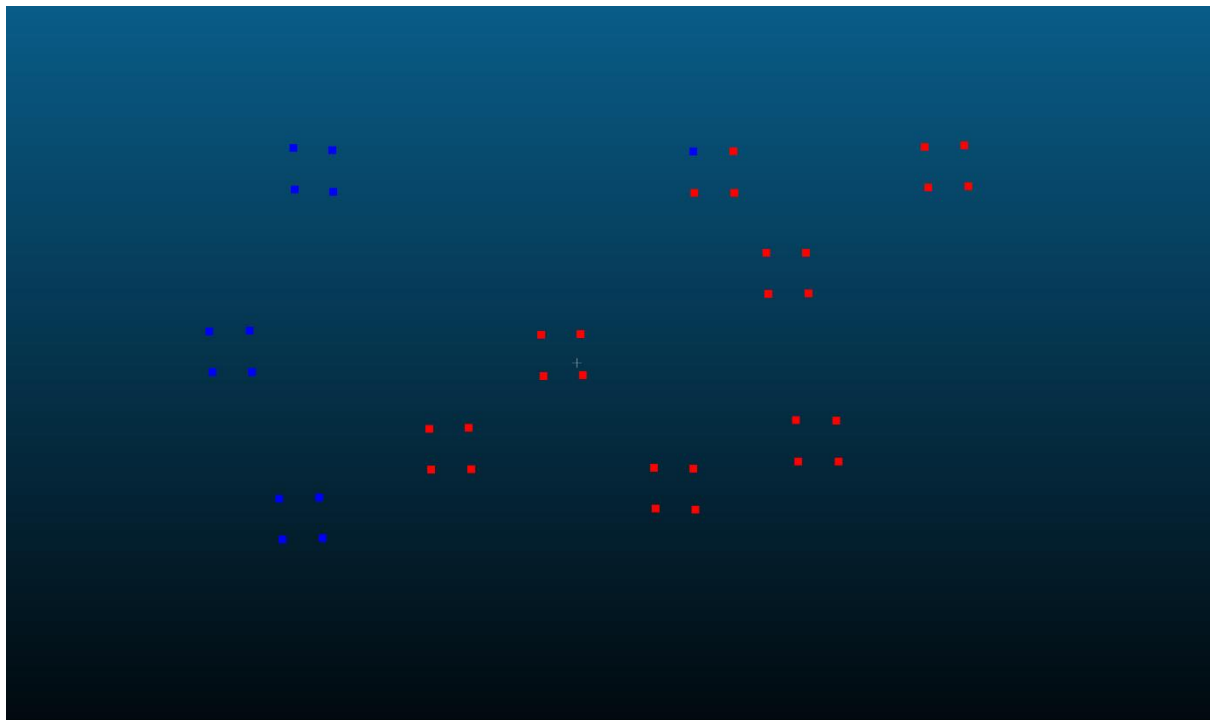
- 6 번의 결과로 생성한 카메라 포즈 'Camera.obj' 또한 CloudCompare 에  
입력으로 드래그해서 넣음으로써 시각화 할 수 있다.



-결과 예시



<카메라 정보 시각화>



<특징점 시각화>

# PnP 알고리즘 (과제 #5.3)

- 목표
- 3 차원 특징점 정답값과 입력영상을 이용한 PnP 알고리즘의 이해
- ArUco 마커의 3 차원 특징점 정답값과 PnP 알고리즘을 이용한 카메라 외부 파라미터 계산

1. ArUco 마커의 정답값에 대한 Text 파일(result.txt), 임의의 사진을 읽는다
  1. 정답값 파일(result.txt) : 각 ArUco 마커의 ID 와 World 좌표계 기준 마커의 코너점(x,y,z) 4 개
  2. 정답값 파일(result.txt)의 Format

```
<ArUco 마커의 ID>
<Corner 1 = x,y,z>
<Corner 2 = x,y,z>
<Corner 3 = x,y,z>
<Corner 4 = x,y,z>
... ..
```

3. 파일 예시

```
1
-1.7780474013548275 -0.5911724465900646 2.538114159560967
-1.6167995200194363 -0.5876025476704967 2.525722531509747
-1.6053059325272965 -0.5597621240987823 2.6867047324758473
-1.7675475951523465 -0.5622238510981535 2.6967577924941404
6
-0.7226252380666552 0.1373691807122849 3.0892130791345105
-0.5631206110358429 0.1405330320573944 3.086604420825473
-0.5659680213599725 0.300476203006917 3.0640311453438516
-0.7261388641456582 0.29693207389012743 3.0662599876236207
7
-0.7524670195106614 -0.6195602434516664 2.297376415423312
-0.5919010400274396 -0.6187448844051906 2.2918126337334934
-0.5843321542216886 -0.5886409587720803 2.4535007712913735
-0.745430237334934 -0.589250295386726 2.457878928920139
8
-0.4510691785365574 0.36744341926084717 3.0491309237523576
-0.29105369353627164 0.36708285496255644 3.0439622161950495
-0.29122848692562736 0.5265930726090436 3.013962988435374
-0.4515618018218716 0.5256628412010519 3.016055590584805
10
-0.3968960171405705 -0.5808066931458298 2.487796400680272
-0.2351234462287772 -0.5800962314640141 2.4813908431372544
-0.22817182822900586 -0.5504007906848454 2.6427155714800206
-0.38968479982507287 -0.5502610991533756 2.6485516747384668
```

#### 4. 관련 코드

```
def read_text_file(file_path):
    data = {}
    with open(file_path, 'r') as file:
        #파일의 각 줄을 읽음
        lines = file.readlines()
        current_id = None
        for line in lines:
            line=line.strip()
            if line:
                #ArUco 마커의 ID를 읽고,
                if line.isdigit():
                    current_id = line.strip()
                    data[current_id] = []
                    #해당 ID의 data에 points(x,y,z)를 입력함.
                else:
                    points = [float(val) for val in line.split()[:3]]
                    data[current_id].append(points)
        #결과값 data는 각 ArUco ID와 그에 해당되는 4개의 코너점들을 가지고 있음.
    return data

text_data = read_text_file(text_file_path)
```

#### 2. 입력 영상으로부터 ArUco 마커의 ID와 corner 점들을 계산

```
def detect_aruco(image_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #'DICT_6X6_250' : ArUco 생성 시 설정하는 파라미터, 사용자의 환경에 맞게 수정가능함.
    aruco_dict = cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_250)
    parameters = cv2.aruco.DetectorParameters_create()
    #흑백영상으로부터 ArUco 마커 검출, 결과값은 ID와 각 corner 4개 점들
    corners, ids, _ = cv2.aruco.detectMarkers(gray, aruco_dict, parameters=parameters)
    return ids, corners

detected_ids, detected_corners = detect_aruco(image_path)
```

3. 1 번의 결과와 2 번의 결과 간의 ID 비교하여, 동일 ID 여부 확인 및 corner 점들 저장

```
def compare_ids(text_data, detected_ids):
    matched_ids = []
    matched_points = []
    detected_ids_list = detected_ids.flatten().tolist()
    for id, points in text_data.items():
        if str(id) in map(str, detected_ids_list):
            matched_ids.append(id)
            matched_points.append(points)
    return matched_ids, matched_points

matched_ids, matched_points = compare_ids(text_data, detected_ids)
```

#### 4. PnP 알고리즘 동작

```
#PnP알고리즘을 돌리기 위해, 데이터 사전정렬
target_updated_corners = []
updated_3dpoint = []

for id_str, corner_points in zip(matched_ids, matched_points):
    # Find the index of the ID in detected_ids
    id_index = np.where(detected_ids == int(id_str))[1][0]
    target_updated_corners.append(detected_corners[id_index][0])
    updated_3dpoint.append(corner_points)

selected_target_updated_corners = [point for sublist in target_updated_corners
                                   for point in sublist]
selected_updated_3dpoint = [point for sublist in updated_3dpoint
                            for point in sublist]

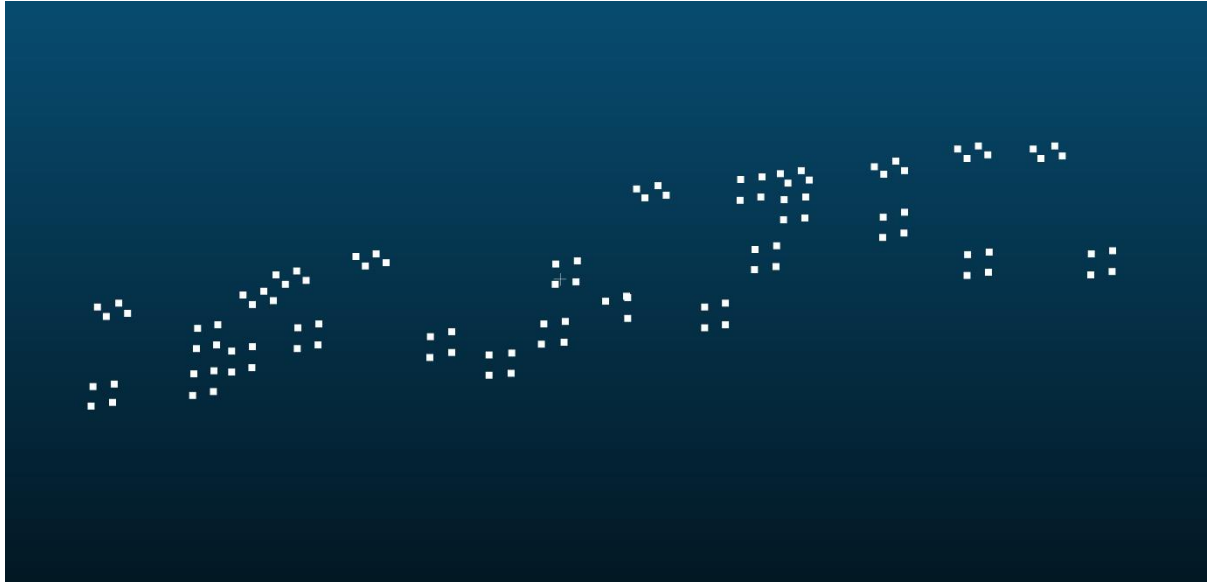
#OpenCV의 PnP 알고리즘
#입력데이터
# - 2차원 포인트들 : selected_target_updated_corners
# - 3차원 포인트들 : selected_updated_3dpoint
# - Camera Parameters : K(카메라 파라미터), D(왜곡함수)
#출력데이터
# - rvec : 회전벡터
# - tvec : 이동벡터
rvec, tvec = solve_pnp(np.array(selected_target_updated_corners),
                      np.array(selected_updated_3dpoint), K, D)

#회전벡터를 회전행렬로 변환
R, _ = cv2.Rodrigues(rvec)

#최종 결과(Transformation Matrix Visualization)
T = tvec
print(np.hstack((R, T)))
```

## 5. 실행결과

### 1. 정답값 시각화 예시



### 2. PnP 알고리즘 실행 결과

-입력영상



## -결과화면

Transformation Matrix

```
[[ 8.75443776e-01  3.08806633e-01 -3.71801907e-01  5.50578160e+00]
 [ 3.15071655e-01  2.18713041e-01  9.23522852e-01  7.19614119e+01]
 [ 3.66507908e-01 -9.25636575e-01  9.41747537e-02  2.09540841e+02]]
```