컴퓨터비전

Fundamental Matrix

2021 - 1학기

Fundamental Matrix

Overview

1. Gaussian Blur (to reduce noise)
2. Keypoint & Descriptor (SIFT)
3. Match Keypoints (knnMatch)
4. Find Good Matches (nndrRatio: 0.4)
5. Compute Fundamental Matrix
6. Compute Epilines

Fundamental Matrix

Main

```cpp
int main() {
	Mat img1 = imread("epip1.jpg", IMREAD_GRAYSCALE);
	Mat img2 = imread("epip2.jpg", IMREAD_GRAYSCALE);

	if (img1.empty() || img2.empty())
		return -1;


	GaussianBlur(img1, img1, Size(5, 5), 0.0);
	GaussianBlur(img2, img2, Size(5, 5), 0.0);

	vector<KeyPoint> keypoints1;
	Mat descriptors1;

	vector<KeyPoint> keypoints2;
	Mat descriptors2;

	SIFT siftF(500, 3);
	siftF.detect(img1, keypoints1);
	siftF.detect(img2, keypoints2);

	siftF.compute(img1, keypoints1, descriptors1);
	siftF.compute(img2, keypoints2, descriptors2);

	vector<vector<DMatch>> matches;
	FlannBasedMatcher matcher;
	int k = 2;
	matcher.knnMatch(descriptors1, descriptors2, matches, k);

	vector<DMatch> goodMatches;
	float nndrRatio = 0.4f;
	for (int i = 0; i < matches.size(); i++) {
		if (matches.at(i).size() == 2 &&
			matches.at(i).at(0).distance <= nndrRatio * matches.at(i).at(1).distance)
			goodMatches.push_back(matches[i][0]);
	}

	cout << "goodMatch size: " << goodMatches.size() << endl;

	if (goodMatches.size() < 8)
		return 0;

	Mat imgMatches;
	drawMatches(img1, keypoints1, img2, keypoints2, goodMatches, imgMatches);
	imshow("goodMAtches", imgMatches);
```

```cpp
	vector<Point2f> kp1;
	vector<Point2f> kp2;
	for (int i = 0; i < goodMatches.size(); i++) {
		kp1.push_back(keypoints1[goodMatches[i].queryIdx].pt);
		kp2.push_back(keypoints2[goodMatches[i].trainIdx].pt);
	}
	Mat l;
	l = FMat(img1, kp1, kp2);
	cout << "my F matrix" << endl;
	cout << l << endl;

	Mat lines1, lines2;
	lines1 = computeEpilines(fundamental_matrix, kp2, 2);
	lines2 = computeEpilines(fundamental_matrix, kp1, 1);

	Mat img3 = drawlines(img1, img2, lines1, kp1, kp2);
	Mat img4 = drawlines(img2, img1, lines2, kp2, kp1);

	imshow("1st", img3);
	imshow("2nd", img4);
	waitKey(0);

	return 0;
}
```

https://github.com/ParkSomin23/21.03.14/blob/master/epip3.cpp

Fundamental Matrix

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Main 1

```
int main() {
	Mat img1 = imread("epip1.jpg", IMREAD_GRAYSCALE);
	Mat img2 = imread("epip2.jpg", IMREAD_GRAYSCALE);

	if (img1.empty() || img2.empty())
		return -1;

	GaussianBlur(img1, img1, Size(5, 5), 0.0);
	GaussianBlur(img2, img2, Size(5, 5), 0.0);

	vector<KeyPoint> keypoints1;
	Mat descriptors1;

	vector<KeyPoint> keypoints2;
	Mat descriptors2;

	SIFT siftF(500, 3);
	siftF.detect(img1, keypoints1);
	siftF.detect(img2, keypoints2);

	siftF.compute(img1, keypoints1, descriptors1);
	siftF.compute(img2, keypoints2, descriptors2);

	vector<vector<DMatch>> matches;
	FlannBasedMatcher matcher;
	int k = 2;
	matcher.knnMatch(descriptors1, descriptors2, matches, k);

	vector<DMatch> goodMatches;
	float nndrRatio = 0.4f;
	for (int i = 0; i < matches.size(); i++) {
		if (matches.at(i).size() == 2 &&
			matches.at(i).at(0).distance <= nndrRatio * matches.at(i).at(1).distance)
			goodMatches.push_back(matches[i][0]);
	}

	cout << "goodMatch size: " << goodMatches.size() << endl;

	if (goodMatches.size() < 8)
		return 0;

	Mat imgMatches;
	drawMatches(img1, keypoints1, img2, keypoints2, goodMatches, imgMatches);
	imshow("goodMAtches", imgMatches);
```

1. Gaussian Blur (to reduce noise)

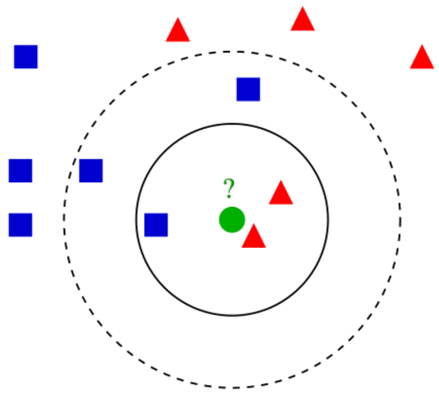
2. Keypoint & Descriptor (SIFT)

보유할 최적 특징의 개수 : 500

한 옥타브의 층 수 : 3 (opencv : 3+3 = 6)

3. Match Keypoints (knnMatch)

4. Find Good Matches (nndrRatio: 0.4)



가장 비슷한 k(=2)개 matching

matching된 k개의 keypoint의 NNDR이 0.4보다 작은 matching만 고른다

$$NNDR = \frac{d_1}{d_2}$$

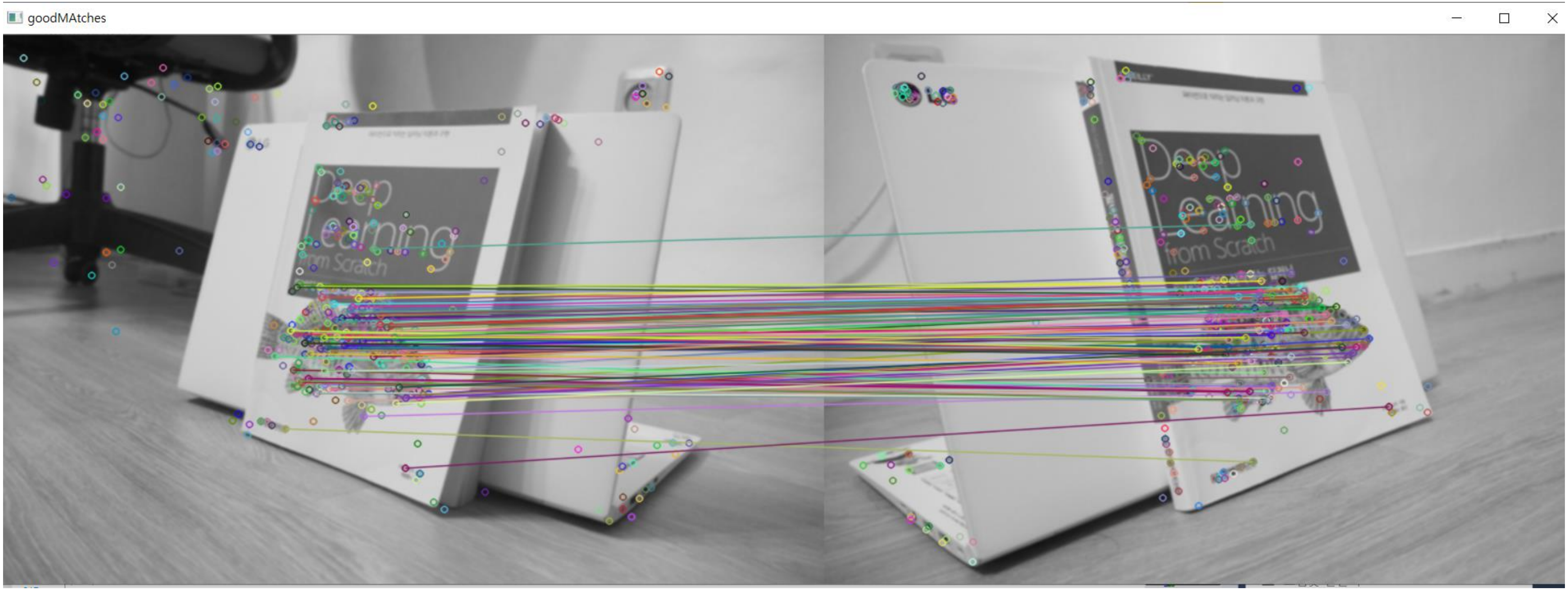
d_1 : 가장 가까운 이웃까지의 거리

d_2 : 두번째로 가까운 이웃까지의 거리

NNDR이 작을수록 좋음

Fundamental Matrix

Main 1 :
Best Match



Fundamental Matrix

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Main 2

```
vector<Point2f> kp1;
vector<Point2f> kp2;
for (int i = 0; i < goodMatches.size(); i++) {
        kp1.push_back(keypoints1[goodMatches[i].queryIdx].pt);
        kp2.push_back(keypoints2[goodMatches[i].trainIdx].pt);
}
Mat l;
l = FMat(img1, kp1, kp2);
cout << "my F matrix" << endl;
cout << l << endl;
Mat lines1, lines2;
lines1 = computeEpilines(fundamental_matrix, kp2, 2);
lines2 = computeEpilines(fundamental_matrix, kp1, 1);

Mat img3 = drawlines(img1, img2, lines1, kp1, kp2);
Mat img4 = drawlines(img2, img1, lines2, kp2, kp1);

imshow("1st", img3);
imshow("2nd", img4);
waitKey(0);

return 0;
}
```

5. Compute Fundamental Matrix

- normalize points
- 8-point algorithm
- enforce rank2 constraint
- denormalize

6. Compute Epilines

Fundamental Matrix

- normalize points
- 8-point algorithm
- enforce rank2 constraint
- denormalize

Compute Fundamental Matrix

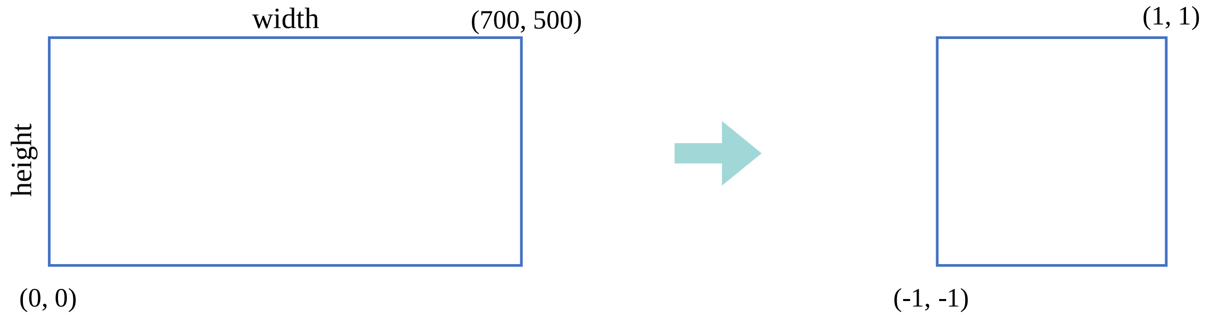
```
Mat FMat(Mat img, vector<Point2f> keypoint1, vector<Point2f> keypoint2) {

    Mat T1 = Mat::zeros(3, 3, CV_32F);
    Mat T2 = Mat::zeros(3, 3, CV_32F);

    vector<Point2f> kp1 = NormPoints(keypoint1, T1);
    vector<Point2f> kp2 = NormPoints(keypoint2, T2);

    int size = kp1.size();
    Mat A(size, 9, CV_32F);
    for (int i = 0; i < kp1.size(); i++) {
            A.at<float>(i, 0) = kp1[i].x * kp2[i].x;
            A.at<float>(i, 1) = kp1[i].y * kp2[i].x;
            A.at<float>(i, 2) = kp2[i].x;

            A.at<float>(i, 3) = kp1[i].x * kp2[i].y;
            A.at<float>(i, 4) = kp1[i].y * kp2[i].y;
            A.at<float>(i, 5) = kp2[i].y;

            A.at<float>(i, 6) = kp1[i].x;
            A.at<float>(i, 7) = kp1[i].y;
            A.at<float>(i, 8) = 1;
    }

    SVD svd(A, SVD::FULL_UV);

    Mat tmp;
    transpose(svd.vt, tmp);

    Mat F;
    F = tmp.col(8);
```

```
    Mat F2(3,3, CV_32F);
    F2.at<float>(0, 0) = F.at<float>(0, 0);
    F2.at<float>(0, 1) = F.at<float>(1, 0);
    F2.at<float>(0, 2) = F.at<float>(2, 0);
    F2.at<float>(1, 0) = F.at<float>(3, 0);
    F2.at<float>(1, 1) = F.at<float>(4, 0);
    F2.at<float>(1, 2) = F.at<float>(5, 0);
    F2.at<float>(2, 0) = F.at<float>(6, 0);
    F2.at<float>(2, 1) = F.at<float>(7, 0);
    F2.at<float>(2, 2) = F.at<float>(8, 0);

    SVD svdF(F2, SVD::FULL_UV);

    Mat d = Mat::zeros(3, 3, CV_32F);
    d.at<float>(0, 0) = svdF.w.at<float>(0);
    d.at<float>(1, 1) = svdF.w.at<float>(1);
    d.at<float>(2, 2) = 0.0;

    Mat fin = svdF.u * d * svdF.vt;

    transpose(T2, T2);
    fin = T2 * fin * T1;

    float F33 = fin.at<float>(2, 2);

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            fin.at<float>(i, j) = fin.at<float>(i, j) / F33;

    return fin;
```

Fundamental Matrix

Compute Fundamental Matrix 1 : normalize points

```
vector<Point2f> NormPoints(vector<Point2f> kp, Mat T) {
        T.at<float>(0, 0) = 2.0f / float(img1.cols);
        T.at<float>(0, 2) = -1.0f;

        T.at<float>(1, 1) = 2.0f / float(img1.rows);
        T.at<float>(1, 2) = -1.0f;

        T.at<float>(2, 2) = 1.0f;

        int size = kp.size();

        Mat newKp(3, size, CV_32F);
        for (int i = 0; i < kp.size(); i++) {
                newKp.at<float>(0, i) = kp[i].x;
                newKp.at<float>(1, i) = kp[i].y;
                newKp.at<float>(2, i) = 1;
        }

        vector<Point2f> newpt;
        for (int i = 0; i < size; i++) {

                Mat pts = T * newKp.col(i);
                newpt.push_back(Point2f(pts.at<float>(0), pts.at<float>(1)));
        }

        return newpt;
}
```



$$\mathbf{T} = \begin{bmatrix} \frac{2}{widht} & 0 & -1 \\ 0 & \frac{2}{height} & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T * kp = T * \begin{bmatrix} kp.x \\ kp.y \\ 1 \end{bmatrix}$$

https://slideplayer.com/slide/3275895/

Compute Fundamental Matrix 2 : 8-point algorithm

$$[x' \quad y' \quad 1] \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

$$\Rightarrow \quad x\,x'e_{11} + y\,x'e_{12} + x'e_{13} + x\,y'e_{21} + y\,y'e_{22} + y'e_{23} + x\,e_{31} + ye_{32} + e_{33} = 0$$

$$\Rightarrow \quad A\,F = 0 \qquad A = [x\,x' \quad y\,x' \quad x' \quad x\,y' \quad y\,y' \quad y' \quad x \quad y \quad 1] \qquad F = \begin{bmatrix} e_{11} \\ \vdots \\ e_{33} \end{bmatrix}$$

seek F to minimize ||AF|| (= least eigenvector of $A^T A$)

Compute Fundamental Matrix 3 : enforce rank2 constraint

- To enforce that F is of rank 2, F is replaced by F' that minimizes $\|\mathbf{F}-\mathbf{F}'\|$ subject to $\det \mathbf{F}'=0$.

- It is achieved by SVD. Let $\mathbf{F}=\mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}}$, where

$$\Sigma=\begin{bmatrix}\sigma_1 & 0 & 0\\ 0 & \sigma_2 & 0\\ 0 & 0 & \sigma_3\end{bmatrix}, \text{ let } \Sigma'=\begin{bmatrix}\sigma_1 & 0 & 0\\ 0 & \sigma_2 & 0\\ 0 & 0 & 0\end{bmatrix}$$

then $\mathbf{F}'=\mathbf{U}\Sigma'\mathbf{V}^{\mathrm{T}}$ is the solution.

Fundamental Matrix

Compute Epilines

mode == 1

$$l_i^{(2)} = Fp_i$$

mode == 2

$$l_i^{(1)} = F^T p_i'$$

(i = i^{th} key point)

line2

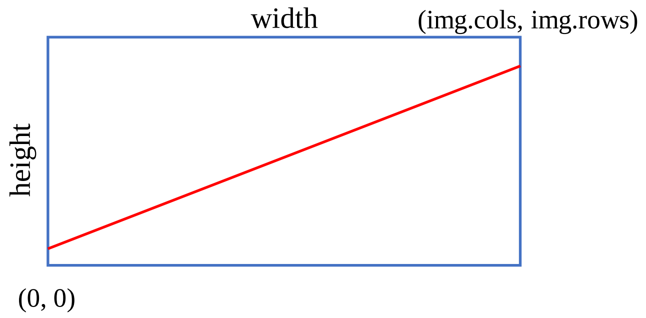
$$ax + by + c = 0 \qquad (a^2 + b^2 = 1)$$

```
Mat computeEpilines(Mat F, vector<Point2f>kp, int mode) {
    F.convertTo(F, CV_32F);

    int size = kp.size();
    Mat line2(size, 3, CV_32F);
    for (int i = 0; i < size; i++) {

        Mat pt(3, 1, CV_32F);
        pt.at<float>(0, 0) = kp[i].x;
        pt.at<float>(1, 0) = kp[i].y;
        pt.at<float>(2, 0) = 1;

        Mat tmp;
        Mat Ft;

        if(mode == 1)
            tmp = F * pt;
        else {
            transpose(F, Ft);
            tmp = Ft * pt;
        }

        float a = tmp.at<float>(0, 0);
        float b = tmp.at<float>(1, 0);

        float t = sqrt(pow(a, 2) + pow(b, 2));

        line2.at<float>(i, 0) = a / t;
        line2.at<float>(i, 1) = b / t;
        line2.at<float>(i, 2) = tmp.at<float>(2, 0) / t;
    }

    return line2;
}
```

Fundamental Matrix

Draw Epilines

```
Mat drawlines(Mat img1, Mat img2, Mat lines, vector<Point2f>pts1, vector<Point2f> pts2) {

        Mat newImg(img1.cols, img1.rows, CV_32F);
        newImg = img1.clone();

        for (int i = 0; i < lines.rows; i++) {
                int x0 = 0;
                int y0 = int(-lines.at<float>(i, 2) / lines.at<float>(i, 1));
                int x1 = int(img1.cols);
                int y1 = int(-(lines.at<float>(i, 2) + lines.at<float>(i, 0) * img1.cols) / lines.at<float>(i, 1));

                line(newImg, Point(x0, y0), Point(x1, y1), (0.255, 3), 1);
                circle(newImg, pts1[i], 5, (0.255, 3), 1);
        }

        return newImg;
}
```



Fundamental Matrix

SIFT(500,3)

```
goodMatch size: 108
Fundamental_matrix
[9.940921314897228e-07, 6.616005632106819e-06, -0.00078797427345379;
  6.726356910076537e-06, -2.874566394109793e-08, 0.002800102069032093;
  -0.001445438846417059, -0.008817054632507782, 1]
my F matrix
[1.1893447e-06, 5.9180834e-06, -0.00069764222;
  6.8886247e-06, 2.0618218e-07, 0.0024703436;
  -0.0016305951, -0.0083414968, 1]
```

Fundamental Matrix

SIFT(2500,3)

```
goodMatch size: 170
Fundamental_matrix
[9.919401916301457e-07, 6.188836945119396e-06, -0.0007520048386492377;
  7.138306547511392e-06, 1.523274063980804e-07, 0.002555598842733464;
  -0.001492461870013029, -0.008579320379886412, 0.9999999999999999]
my F matrix
[1.2887452e-06, 5.1804113e-06, -0.00062092772;
  7.3017509e-06, 4.8389296e-07, 0.0020777809;
  -0.0017647097, -0.0078661731, 1]
```