# Project 3: Visual Odometry

ENPM673 Perception for Autonomous Robots - Spring 2017


University Of Maryland

Project Report *by*

Neel Parikh
Ravi Bhadeshiya
Kavit Patel

# Contents

**List of Figures** **1**

**1 Brief Pipeline** **2**
1.1 Preprocessing . . . . . 2
1.2 Feature detection and Matching . . . . . 2
1.3 Odometry . . . . . 3
1.4 Output . . . . . 6

# List of Figures

1.1 Final Trajectory . . . . . 6

# Chapter 1

# Brief Pipeline

## 1.1 Preprocessing

The given image is a mosaic image, so the first step is to demosaic image in order to further process it. After demosaicing of the image, the next step is to undistort it. Undistorting the image gives us a better image which is then used for feature detection and to compute visual odometry.

## 1.2 Feature detection and Matching

This is one of the most important step in the algorithm. SURF features are detected and tracked across all the frames. Once the tracker detects the corresponding points in the current frame, this information can be used to compute fundamental matrix and then translation and rotation between two frames. The below shown snippet of code does what is explained in this paragraph.

```
1 function [points , feature, vPoints] = customFeature(image)
2 if size(image,3) == 3
3 image=rgb2gray(image);
4 end
5 points = detectSURFFeatures(image,'MetricThreshold', 1000);
6 % points = detectFASTFeatures(image);
7 points = selectUniform(points, 500, size(image));
8 % points = selectStrongest(points, 200);%, size(image));
9 [feature, vPoints] = extractFeatures(image, points, 'Upright'
    , true);
10 end
```

**Listing 1.1:** Starter code

```
function [points , feature ,idxPair , vPoints]=
    customMatchFeature(image ,pFeature ,pImage ,pvPoints)
if size(image,3) == 3
image=rgb2gray(image);
end
[points , feature , vPoints]=customFeature(image);
idxPair = matchFeatures(pFeature , feature , 'Unique', true);
matchedPoints1 = pvPoints(idxPair(:, 1));
matchedPoints2 = vPoints(idxPair(:, 2));
end
```

**Listing 1.2:** Starter code

## 1.3 Odometry

### Estimate Fundamental Matrix

Estimating fundamental matrix is the key step since all further calculations are based on it. To estimate fundamental matrix, RANSAC algorithm is used and the corresponding matched pairs are given to the function. It also returns the inliers based on the computed fundamental matrix.

### Essential Matrix

The fundamental matrix computed in the previous step can be used to estimate Essential matrix. Essential matrix is computed from the fundamental matrix and camera parameters as shown in the snippet below.

```
function E = FtoEmatrix(F,K)
E=K'*F*K;
[u d v]=svd(E);
i=eye(3);
i(3,3)=0;
E=u*i*v';
% E=E/norm(E);
end
```

**Listing 1.3:** Starter code

### Extract Camera Pose

Extracting the camera pose is the step which gives possible rotations and translations between the two frames. To extract camera pose, essential matrix computed

in the previous step is give as an input. The following snippet of the code shows the mathematical computation behind it.

```
1
2 function [Cset,Rset] = ExtractCameraPose(E)
3
4 W = [0 -1 0;1 0 0;0 0 1];
5 [U,~,V] = svd(E);
6 R = U*W*V';
7 C = U(:,3);%./max(U(:,3));
8 if det(R) < 0
9     R = -R;
10    C = -C;
11 end
12 Rset(:,:,1) = R;
13 Cset(:,:,1) = (-R'*C);
14
15 R = U*W*V';
16 C = -U(:,3);%./max(U(:,3));
17 if det(R) < 0
18    R = -R;
19    C = -C;
20 end
21 Rset(:,:,2) = R;
22 Cset(:,:,2) = (-R'*C);
23
24 R = U*W'*V';
25 C = U(:,3);%./max(U(:,3));
26 if det(R) < 0
27    R = -R;
28    C = -C;
29 end
30 Rset(:,:,3) = R;
31 Cset(:,:,3) =( -R'*C);
32
33 R = U*W'*V';
34 C = -U(:,3);%./max(U(:,3));
35 if det(R) < 0
36    R = -R;
37    C = -C;
38 end
```

```
39 Rset(:,:,4) = R;
40 Cset(:,:,4) = (-R'*C);
41 end
```

**Listing 1.4:** Starter code

### Linear Triangulation

Extract camera pose gives four possible values of rotations and translations. Triangulation is the process of determining a point in 3D space given its projections onto two, or more, images. This step gives points in 3D space for the rotations and translations calculated in extract camera pose.

### Select Correct Essential Camera Matrix

This step selects the correct rotation and translation out of the computed possible rotations and translations. This step also uses Linear triangulation mentioned above to calculate 3D point. For selecting the valid point sum of the depth is calculated and based on that value the correct point is found out.

```
1 function [cRot,cT,correct] =
    SelectCorrectEssentialCameraMatrix(rot,t,x1,x2,k)
2 % Computes 3D points
3 nx1 = NormalizedCoordinates(x1,k);
4 nx2 = NormalizedCoordinates(x2,k);
5 for i=1:4
6     x3D(:,:,i) = LinearTriangulation(nx1, nx2, rot(:,:,i), t
    (:,:,i));
7     x3D(:,:,i) = HomogeneousCoordinates(x3D(:,:,i),'3D');
8 end
9
10 correct = 0;
11 for i=1:4
12     % compute the depth & sum the sign
13     depth(i,1) = sum(sign(DepthOfPoints(x3D(:,:,i),eye(3),
    zeros(3,1)))); %using canonical camera
14     depth(i,2) = sum(sign(DepthOfPoints(x3D(:,:,i),rot(:,:,i)
    ,t(:,:,i)))); % using the recovered camera
15 end
16
17 if(depth(1,1)<0 && depth(1,2)<0)
18     correct = 1;
```

```
elseif(depth(2,1)<0 && depth(2,2)<0)
    correct = 2;
elseif(depth(3,1)<0 && depth(3,2)<0)
    correct = 3;
elseif(depth(4,1)<0 && depth(4,2)<0)
    correct = 4;
end
%correct

% return the selected solution
cRot = rot(:,:,correct);
cT = t(:,correct);
```

**Listing 1.5:** Starter code

## 1.4 Output

The trajectory computed after the completion of whole video is as shown in the figure given below. The average time taken to process each frame is approximately 0.6 to 0.7 seconds.

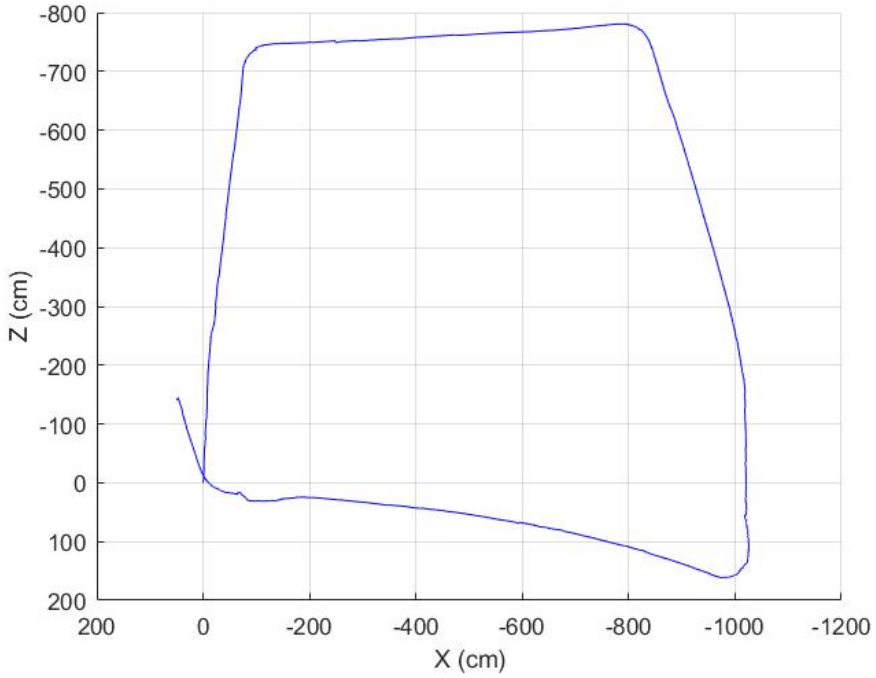


**Figure 1.1:** Final Trajectory