PROJECT-2: VISUAL ODOMETRY

ENPM 673 – PERCEPTION FOR AUTONOMOUS ROBOTS

PROF. CORNELIA FURMULLER

UNIVERSITY OF MARYLAND COLLEGE PARK

PROJECT REPORT BY:

ABHISHEK KATHPAL

DIRECTORY_NAME: AKATHPAL

CONTENTS:

BRIEF PIPELINE

In this project, the position of the camera centre between two successive frames is computed and plotted. The brief pipeline for computing Visual Odometry is described as follows. The first step is computing intrinsic camera parameters with help of **ReadCameraModel** function. The next step is extracting features from each frame and getting the matched features. After that, the fundamental matrix is computed using **8-point Normalized Algorithm and RANSAC** is further used to compute best fundamental matrix. Further the camera poses are computed using Essential matrix obtained from fundamental matrix. Then, selecting correct camera pose using triangulation and constraints because the car is moving in positive Z-direction. At last, update the global pose of the camera.

IMAGE PREPROCESSING

The images given in the oxford dataset are in Bayer's format. To recover the color images, **demosaic** function with grgb alignment is used. Then use the **Undistort** function to undistort the image. After that Gaussian filter is used to smooth the images and then the image is then converted into grayscale image. This image is used for further extracting features. I have saved the matched features in a cell to save the running time.

```
images_folder = fullfile('D:/Robotics-Perception/Visual_Odometry_Project2/input/Oxford_dataset/stereo/centre');
images = imageDatastore(images_folder,'IncludeSubfolders',true);
model=fullfile('D:/Robotics-Perception/Visual_Odometry_Project2/input/Oxford_dataset/model');
[fx, fy, cx, cy, G_camera_image, LUT]=ReadCameraModel(images_folder,model);
K=[fx 0 cx;0 fy cy;0 0 1];

image1 = readimage(images, 1);
image1 = demosaic(image1,'gbrg');
image1 = UndistortImage(image1,LUT);
image1 = imgaussfilt(image1,0.5);
image1 = rgb2gray(image1);
image1_points = detectSURFFeatures(image1,'MetricThreshold', 1000);
image1_points = selectUniform(image1_points, 500, size(image1));
[image1_features, image1_points] = extractFeatures(image1, image1_points, 'Upright', true);
MatchedSURFPoints = cell(numel(images.Files)-1);
```

EXTRACTING AND MATCHING FEATURES

In this project, SURF features are extracted using MATLAB **detectSURFFeatures** and **extractFeatures** function. Features are extracted for two consecutive frames and then using **matchFeatures** function, 500 unique matching features are extracted.

```
for i=2:numel(images.Files)

    image2 = readimage(images, i);
    image2 = demosaic(image2,'gbrg');
    image2 = UndistortImage(image2,LUT);
    image2 = imgaussfilt(image2,0.5);
    image2 = rgb2gray(image2);
    image2_points = detectSURFFeatures(image2,'MetricThreshold', 1000);
    image2_points = selectUniform(image2_points, 500, size(image2));
    [image2_features, image2_points] = extractFeatures(image2, image2_points, 'Upright', true);

    %% Matching features
    index = matchFeatures(image1_features, image2_features, 'Unique', true);
    matchedPoints1 = image1_points(index(:, 1));
    matchedPoints2 = image2_points(index(:, 2));
    MatchedSURFPoints{i-1} = {matchedPoints1,matchedPoints2};
    image1 = image2;
    image1_points = image2_points;
    image1_features = image2_features;

end
save MatchedSURFPoints
save K
```

ESTIMATING FUNDAMENTAL MATRIX

Fundamental matrix is implemented using two methods in this project. The first method used the MATLAB functions **estimateFundamentalMatrix** inbuilt function with RANSAC approach. In the second method, I have written my own code for estimation of fundamental Matrix using 8-point Normalized algorithm. For selecting the best fundamental matrix, Random Sampling and consensus method is used, which calculates the error using the epipolar constraints and then find the fundamental matrix with minimum error.

I also tried computing the fundamental matrix using all matched points but the output was not satisfactory.

The inliers are also computed by checking which point lead to the error less than a particular threshold. The inlier points are further used in Linear Triangulation.

```
%% Fundamental Matrix
[F,inlier_points1,inlier_points2] = get_fundamental_8point_RANSAC(matchedPoints1,matchedPoints2,N,1);
E=K'*F*K;
[u,d,v]=svd(E);
E=u*diag([1,1,0])*v';
E = E/norm(E);

[Hset,Rset,Cset] = getting_poses(E);
[Xset,Xset_new] = linear_triangulation(K,Cset,Rset,inlier_points1.Location,inlier_points2.Location);
[R, t] = disambiguate_points(Cset,Rset,Xset,Xset_new);

t_t = t_t + R_t * t;
R_t = R_t * R;
location = [location;[t_t(1),t_t(3)]];
```

```
%% MATLAB Implementation

 [F_matlab,inlierIdx] = estimateFundamentalMatrix(matchedPoints1,matchedPoints2,...
     'Method','RANSAC','NumTrials',1000,'DistanceThreshold',1e-4);
 [relativeOrientation,relativeLocation] = relativeCameraPose(F_matlab,cameraParams,...
     matchedPoints1.Location(inlierIdx,:),matchedPoints2.Location(inlierIdx,:));
 R1 = relativeOrientation;
 t1 = relativeLocation';
 t_mat = t_mat + R_mat * t1;
 R_mat = R_mat * R1;
 location_mat = [location_mat;[t_mat(1),t_mat(3)]];
```

COMPUTING CAMERA POSE

After finding the best fundamental matrix, the next step is to compute the essential matrix (E) from fundamental process. After that, Rotation and Translation is computed with the help of the below function. There are total four possible combinations of these rotation and translations. The next step is to compute the correct camera pose by keeping the constraints in mind related to the car moving in X-Z plane and in positive Z-direction.

```
function [Hset,Rset,Cset] = getting_poses(E)

W = [0 -1 0;1 0 0;0 0 1];
[U,~,V] = svd(E);
R1 = U*W*V';
C = U(:,3);
R2 = U*W'*V';
Hset{1} =[R1 C; 0 0 0 1];
Hset{2} =[R1 -C; 0 0 0 1];
Hset{3} =[R2 C; 0 0 0 1];
Hset{4} =[R2 -C; 0 0 0 1];
for i = 1:4
    if det(Hset{i}(1:3,1:3))<0
        Hset{i}(1:3,1:4) = -Hset{i}(1:3,1:4);
    end
    Rset{i} = Hset{i}(1:3,1:3);
    Cset{i} = Hset{i}(1:3,4);
end
```

```
function [R, t] = disambiguate_points(Cset,Rset,Xset,Xset2)

    C = [0 0 0 0];

    for i = 1: 4
        C(i) = sum(Xset2(3,:,i)>0) + sum(Xset(3,:,i)>0);
    end

    if C(1) == 0 && C(2) == 0 && C(3)==0 && C(4)==0
        R = eye(3);
        t = [0;0;0];
    else

        [~, index] = max(C);
        R= Rset{index};
        t = Cset{index};
        if t(3) < 0
            t = -t;
        end
    end
```

LINEAR TRIANGULATION

The next step is computing the 3d points from the given 2d points. This is done using the method called Linear Triangulation. The function for that is shown below-

```
function [Xset,Xset_new] = linear_triangulation(K,Cset,Rset,x1,x2)

    n = size(x1,1);
    x= [x1 ones(n,1)]';
    y = [x2 ones(n,1)]';
    x1 = (inv(K)*x);
    x2 = (inv(K)*y);
    Xset = zeros(4,8,4);
    for j=1:4
        rot = Rset{j};
        t = Cset{j};
        p1 = eye(3,4);
        p2 = [rot t];
        H = [p2; 0 0 0 1];
        for i=1:size(x1,2)
            a = [ x1(1,i)*p1(3,:) - p1(1,:); ...
                  x1(2,i)*p1(3,:) - p1(2,:); ...
                  x2(1,i)*p2(3,:) - p2(1,:); ...
                  x2(2,i)*p2(3,:) - p2(2,:) ];

            [~,~,v] = svd(a);
            X = v(:,end);
            Xset(:,i,j) =X./X(4);
            Xset_new(:,i,j) = H*Xset(:,i,j);
        end
```

After getting the 3d points, we find the 3d points in the corresponding next frame. The condition to select the correct pose Is that both the set of 3d points should have their z-value greater than 0.

COMPUTING CAMERA TRAJECTORY

Now after the previous step, we have the relative camera pose and orientation. To compute the position of the camera with respect to world coordinate frames, I have used the following equations. After computing the trajectories from my own function and MATLAB functions, I computed the **accumulated drift**. **The output varies a little bit when you run the code because of random selection of points.** To get a more reliable output, you can increase the no. of trials in RANSAC. The output below are shown for N-20 (Fig. 2.) and N-200(Fig. 3). **The accumulated drift when N=200 is 1463262**.

$$R_{K+1} = R_K \, X \, {}^{K}R_{K+1}$$

$$T_{K+1} = T_K + R_K \, X \, {}^{K}T_{K+1}$$

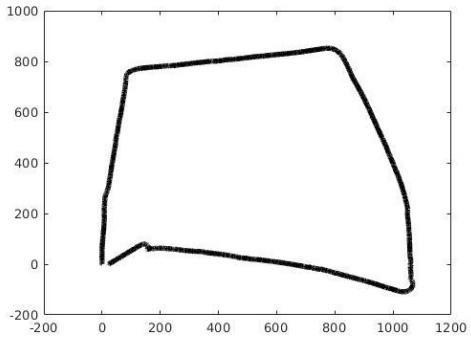

Fig. 1 MATLAB Output

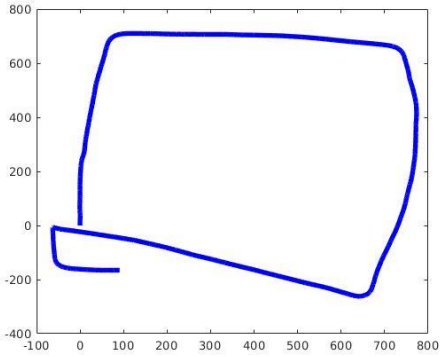

Fig. 2. My Output

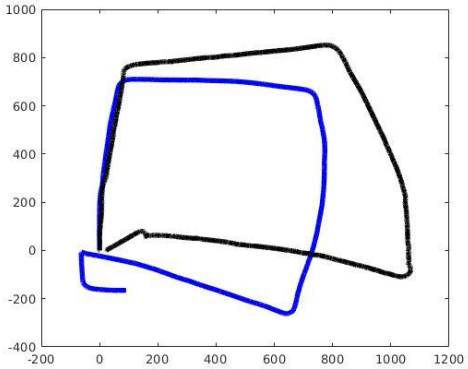

Fig. 4. N=20

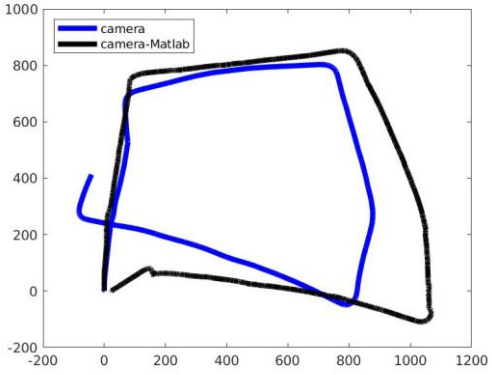

Fig. 3. N=200

REFERENCES

1) Multi-View Geometry in Computer vision by Richard Hartley
2) Coursera, Robotics Perception Course Lectures, UPenn, Prof. Kostas
3) The references provided in the instructions for this project.
4) Fundamental Matrix lecture by Dr. Mubarak Shah : https://www.youtube.com/watch?v=K-j704F6F7Q&t=3122s