Project 1: Augmented Reality

ENPM673 Perception for Autonomous Robots - Spring 2017

University Of Maryland

Project Report *by*

Neel Parikh
Ravi Bhadeshiya
Kavit Patel

Contents

Abstract

This project deals with an interesting idea and gives an insight to some of the fundamental concepts in the field of Robotic perception. The project deals with AprilTag detection and read the data encrypted in the tag. It includes the concept of warping the given image using projective transform and writing the image to the previously detected AprilTag and also with elementary level idea of augmented reality.

This report circumscribes the steps followed in order to solve the presented problem. It gives the higher understanding of the underlying concepts which are needed to solve this problem. Excerpts of the code are provided for key steps.

Chapter 1

Implementation

1.1 Corner Detection

The most fundamental and key step of the project is corner detection. It is an approach used in computer vision systems to extract corner features. It is frequently used in motion detection, image registration, video tracking, image mosaicing, panorama stitching, 3D modeling and object recognition. Corner detection overlaps with the topic of interest point detection.

There are ample amount of methods available to detect corners of quadrilateral. Some of the famous approaches are Harris corner detector,Plessey detector or using Hough transform to find houglines and then find intersection point, etc. However, Harris corner detection method is very efficient and promising compare to other.So, this method was used for the project.

1.2 Blob detection

To detect the corner, one firsts needs to extract the blob of ArTag quadrilateral. The frame from the video is converted to grayscale and then binarized. Once the binzarized image is obtained, it becomes very easy to extract the blob using simple bitwise operations on binary images.

```
1 function blob=findBlob(image)
2
3 firstFrame=rgb2gray(image);
4 [r c]=size(firstFrame);
5 firstFrame=imbinarize(firstFrame);
6
7 CC=bwconncomp(~firstFrame);
```

```
numPixels = cellfun(@numel,CC.PixelIdxList);
[biggest,idx] = max(numPixels);
firstFrame(CC.PixelIdxList{idx}) = 1;

CC=bwconncomp(firstFrame);
numPixels = cellfun(@numel,CC.PixelIdxList);
[smallest,idx] = min(numPixels);
firstFrame(CC.PixelIdxList{idx}) = 0;

blob=firstFrame;

end
```

Listing 1.1: Code for blob detection

This excerpt shows the approach taken in this project to extract the blob of ArTag. All connected components in the binarized image are found and then the biggest blob is set to *True*. Similarly in the second iteration again connected components are found and the smallest blob is set to *False*.

1.2.1 Harris corner detection

After extracting the blob, Harris corner detection method was used to detect four strongest corners in the extracted blob. *Harris* function by Peter Kovesi is used in order to detect the corners.

```
function [corner_pts]=findCorner(image)

firstFrame=findBlob(image);

[rows cols]=size(firstFrame);

[~,r,c]=harris(firstFrame,2,0.04,'N',4);% By Peter Kovesi

corner_pts=[c r];

end
```

Listing 1.2: Code for extracting corners

1.3 April Tag detection

1.3.1 Homography and Image wrap

In order to read the ArTag, the second step after extracting corners is to use inverse homography and unwarp the image in the video. In order to detect the correct orientation of the tag one needs to check different possibilities of the corners detected. In this project, probabilistic detection method which is discussed in the next section.

```
function [id,pts] = aprilTagDetect(img,pts,refImage)
[rY, rX, ~] = size(refImage);
ref_pts = [0 0; rX 0; rX rY; 0 rY];
id=0;

[~,I] = sort(angle(complex(pts(:,1)-mean(pts(:,1)),pts(:,2)-

mean(pts(:,2)))));
pts=pts(I,:);

for i=1:4

    transRef=fitgeotrans(pts,ref_pts,'projective');

    warpedTag=imwarp(img,transRef,'OutputView',imref2d(size(
   refImage)));

    ID=imbinarize(rgb2gray(imresize(warpedTag,[8,8])));

    if(probDetect(ID,80))
        id=8*ID(5,4)+4*ID(5,5)+2*ID(4,5)+ID(4,4);
    else
        pts=circshift(pts,1);
    end
end
end
```

Listing 1.3: Code for homography and unwarping image

1.3.2 Probabilistic detection

After number of iterations, it was detected that a simpler approach like reading the data directly from the unwarped image doesn't work well and thus probabilistic

method was implemented for reading the tag ID. The snippet of the code shown below, shows how the probability was used to detect the correct orientation of the tag. The border cells are given negative weights and thus if there is border data it means that the tag is not in the correct orientation. The highest weight is given to the notch which determines whether or not the tag is in correct orientation. Using this technique promising results were obtained as compared to trivial approach.

```
function bool = probDetect(matrix,likelyhood)
border=-2.083;
noch=33.32;
side=8.33;

prob=[border,border,border,border,border,border,border,border
    ;
     border,border,border,border,border,border,border,border;
     border,border,-side,side,side,-side,border,border;
     border,border,side,   0   ,  0    ,side,border,border;
     border,border,side,   0   ,  0    ,side,border,border;
     border,border,-side,side,side,noch,border,border;
     border,border,border,border,border,border,border,border;
     border,border,border,border,border,border,border,border
    ;];

detection=sum(sum(prob.*matrix));
if detection > likelyhood
    bool=true;
else
    bool=false;
end

end
```

Listing 1.4: Code for correct orientation of tag

1.4 Lena Projection

After reading the tag, the next task presented was to project the *Lena* image on to the tag. The simple approach taken to do this part is shown in the code below. Here inbuilt function of MATLAB, *fitgeotrans* was used to obtain the homography between Lena and tag in the video. Once the homography is calculated, bitwise operations are used to put it in the video.

```
function projected_imgs=lenaProject(logo_pts,corner_pts,
    logo_img,frame)
            trans=fitgeotrans(logo_pts,corner_pts,'projective
    ');
            warped=imwarp(logo_img,trans,'OutputView',imref2d
    (size(frame)));
            blob=findBlob(frame);
            blob3D=im2uint8(cat(3,blob,blob,blob));
            frame=bitand(frame,blob3D);
            projected_imgs=frame+warped;
end
```

Listing 1.5: Code for projecting *Lena*

1.5 Cube Projection

Unembellished idea of the augmented reality is the third part of the project in which the cube is projected into video frame on the cube.

1.5.1 Projective Transformation

Width, height, and depth of the cube are calculated and then they are used to calculate vertices of the base of the cube which is to be projected on the cube. Those points are then multiplied with camera calibration matrix.

```
function cubeProject(frame,corner_pts)

K =[1406.08415449821,0,0;
    2.20679787308599, 1417.99930662800,0;
    1014.13643417416, 566.347754321696,1]';

pr = [ 1 0 0;
    0 1 0 ];

tag_width = 0.10;
tag_height = 0.10;
cube_depth = -0.10;

corner_pt = [  tag_width/2,  tag_height/2;
    -tag_width/2,  tag_height/2;
    -tag_width/2, -tag_height/2;
    tag_width/2, -tag_height/2 ];
```

```
18
19 render_points = [ corner_pt, zeros(4,1);
20     corner_pt, (cube_depth)*ones(4,1) ];
21
22 p = (pr*(K \ [corner_pts'; ones(1,4)]))';
23 H = est_homography(corner_pt,p);
24
25 [proj_pts, ~, ~] = ar_cube(H,render_points,K);
26 inds = [ 1,2, 1,4, 1,5, 2,3, 2,6, 3,4, 3,7, 4,8, 5,6, 5,8,
     6,7, 7,8 ];
27
28 X = proj_pts(inds,:);
29
30 for j = 1:2:length(X)
31     Xj = X([j,j+1],:);
32     line(round(Xj(:,1)), round(Xj(:,2)),'LineWidth', 3,'color
     ','green');
33 end
34 pause(1/10);
35 end
```

Listing 1.6: Code for computing projective transform

1.5.2 Cube point and Homography

The next step after calculating the vertices of the base face is to obtain the coordinates of the pixels where the vertices will lie in the image. The projected points are then multiplied with camera calibration matrix and homography to project the cube.

```
1 function [proj_points, t, R] = ar_cube(H,render_points,K)
2 h1 = H(:,1);
3 h2 = H(:,2);
4 h3 = cross(h1, h2);
5 [U,S,V] = svd([h1 h2 h3]);
6
7 R = U * [1 0 0; 0 1 0; 0 0 det(U*V')]*V';
8 t = H(:,3)/norm(h1);
9
10 [n,m] = size(render_points);
11 proj_points = zeros(n,2);
12 for i = 1:n
```

```
13      result_vector = K *(R * [render_points(i,1);render_points
        (i,2);render_points(i,3)]+t);
14      proj_points(i,1) = result_vector(1)/result_vector(3);
15      proj_points(i,2) = result_vector(2)/result_vector(3);
16  end
17  end
```

Listing 1.7: Code for calculating homography

1.5.3 Draw cube

The final step is to draw the edges of the cube to generate cube on the ArTag. For this, *plot* function of MATLAB is used which will plot all the edges of the cube between the vertices.

```
1  function cubeProject(frame,corner_pts)
2
3  K =[1406.08415449821,0,0;
4      2.20679787308599, 1417.99930662800,0;
5      1014.13643417416, 566.347754321696,1]';
6
7  pr = [ 1 0 0;
8      0 1 0 ];
9
10 tag_width = 0.10;
11 tag_height = 0.10;
12 cube_depth = -0.10;
13
14 corner_pt = [  tag_width/2,  tag_height/2;
15     -tag_width/2,  tag_height/2;
16     -tag_width/2, -tag_height/2;
17      tag_width/2, -tag_height/2 ];
18
19 render_points = [ corner_pt, zeros(4,1);
20     corner_pt, (cube_depth)*ones(4,1) ];
21
22 p = (pr*(K \ [corner_pts'; ones(1,4)]))';
23 H = est_homography(corner_pt,p);
24
25 [proj_pts, ~, ~] = ar_cube(H,render_points,K);
26 inds = [ 1,2, 1,4, 1,5, 2,3, 2,6, 3,4, 3,7, 4,8, 5,6, 5,8,
       6,7, 7,8 ];
27
```

```
28 X = proj_pts(inds,:);
29
30 for j = 1:2:length(X)
31     Xj = X([j,j+1],:);
32     line(round(Xj(:,1)), round(Xj(:,2)),'LineWidth', 3,'color
    ','green');
33 end
34 pause(1/10);
35 end
```

Listing 1.8: Code for plotting cube