Project 2: Lane Detection

ENPM673 Perception for Autonomous Robots - Spring 2017

University Of Maryland

Project Report *by*

Neel Parikh
Ravi Bhadeshiya
Kavit Patel

Contents

List of Figures

Chapter 1

Brief Pipeline - Lane detection

1.1 Pre-process Image

Preprocessing is heavily dependent on feature extraction method and input image type.The step below were followed to pre-process image.
The first step is to undistorted image and then denoise.

Figure 1.1: Main Image

```
clear all;close all;clc

video=VideoReader('DataSet/project_video.mp4');
numberToExtract=2;
```

```
5 cd scripts;
6 rectI=zeros(720,720);
7
8 trap=[178 720; 552 450; 728 450; 1280 720];
9 rect=[475 720; 475 0; 800 0; 800 720];
10
11 c = [552 178 1280 728];
12 r = [450 720 720 450];
13
14 BW = poly2mask(c,r,720,1280);
15 i=1;
16 while hasFrame(video)
17
18     mainframe = readFrame(video);
19
20     frame=mainframe;
21
22     frame=undistortimage(frame, 1.6281e+03, 6.71627794e+02,
       3.86046312e+02, -2.42565104e-01,-4.77893070e
       -02,-1.31388084e-03,-8.79107779e-05,2.20573263e-02);
23
24     frame=imgaussfilt(frame,2);
25
26     gray=rgb2gray(frame);
27
28     gray=uint8(BW).*gray;
29
30     gray=im2bw(gray,0.65);
31
32     gray=imdilate(gray,strel('disk',6));
```

Listing 1.1: Code for blob detection

Figure 1.2: Image mask

To correct image for lens distortion.

Figure 1.3: UndistortedDiff

Figure 1.4: Undistorted

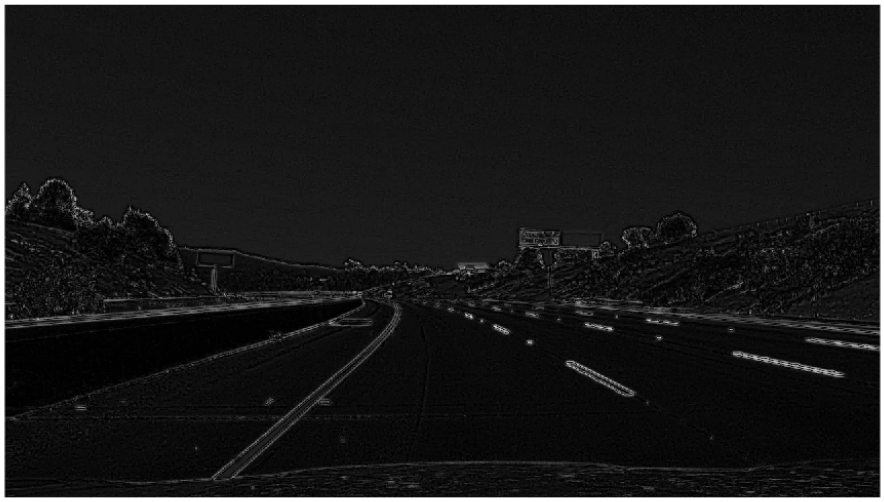

Figure 1.5: Blur image diff

1.1.1 Undistored Image

```
1 % UNDISTORTIMAGE - Removes lens distortion from an image
2 %
3 % Usage:   nim = undistortimage(im, f, ppx, ppy, k1, k2, k3,
    p1, p2)
```

```
4 %
5 % Arguments:
6 %          im - Image to be corrected.
7 %           f - Focal length in terms of pixel units
8 %               (focal_length_mm/pixel_size_mm)
9 %     ppx, ppy - Principal point location in pixels.
10 %   k1, k2, k3 - Radial lens distortion parameters.
11 %       p1, p2 - Tangential lens distortion parameters.
12 %
13 % Returns:
14 %          nim - Corrected image.
15 %
16 %  It is assumed that radial and tangential distortion
    parameters are
17 %  computed/defined with respect to normalised image
    coordinates corresponding
18 %  to an image plane 1 unit from the projection centre.  This
     is why the
19 %  focal length is required.
20
21 % Copyright (c) 2010 Peter Kovesi
22 % Centre for Exploration Targeting
23 % The University of Western Australia
24 % peter.kovesi at uwa edu au
25 %
26 % Permission is hereby granted, free of charge, to any person
     obtaining a copy
27 % of this software and associated documentation files (the "
    Software"), to deal
28 % in the Software without restriction, subject to the
    following conditions:
29 %
30 % The above copyright notice and this permission notice shall
     be included in
31 % all copies or substantial portions of the Software.
32 %
33 % The Software is provided "as is", without warranty of any
    kind.
34
35 % October   2010  Original version
36 % November  2010  Bilinear interpolation + corrections
```

```
37 % April      2015  Cleaned up and speeded up via use of
       interp2
38 % September 2015  Incorporated k3 + tangential distortion
       parameters
39
40 function nim = undistortimage(im, f, ppx, ppy, k1, k2, k3, p1
       , p2)
41
42     % Strategy: Generate a grid of coordinate values
       corresponding to an ideal
43     % undistorted image.  We then apply the imaging process
       to these
44     % coordinates, including lens distortion, to obtain the
       actual distorted
45     % image locations.  In this process these distorted image
        coordinates end up
46     % being stored in a matrix that is indexed via the
       original ideal,
47     % undistorted coords.  Thus for every undistorted pixel
       location we can
48     % determine the location in the distorted image that we
       should map the grey
49     % value from.
50
51     % Start off generating a grid of ideal values in the
       undistorted image.
52     [rows,cols,chan] = size(im);
53     [xu,yu] = meshgrid(1:cols, 1:rows);
54
55     % Convert grid values to normalised values with the
       origin at the principal
56     % point.  Dividing pixel coordinates by the focal length
       (defined in pixels)
57     % gives us normalised coords corresponding to z = 1
58     x = (xu-ppx)/f;
59     y = (yu-ppy)/f;
60
61     % Radial lens distortion component
62     r2 = x.^2+y.^2;                    % Squared normalized
       radius.
63     dr = k1*r2 + k2*r2.^2 + k3*r2.^3;  % Distortion scaling
```

```
    factor.
64
65      % Tangential distortion component (Beware of different p1
        ,p2
66      % orderings used in the literature)
67      dtx =    2*p1*x.*y       +  p2*(r2 + 2*x.^2);
68      dty = p1*(r2 + 2*y.^2)   +    2*p2*x.*y;
69
70      % Apply the radial and tangential distortion components
        to x and y
71      x = x + dr.*x + dtx;
72      y = y + dr.*y + dty;
73
74      % Now rescale by f and add the principal point back to
        get distorted x
75      % and y coordinates
76      xd = x*f + ppx;
77      yd = y*f + ppy;
78
79      % Interpolate values from distorted image to their ideal
        locations
80      if ndims(im) == 2   % Greyscale
81          nim = interp2(xu,yu,double(im),xd,yd);
82      else % Colour
83          nim = zeros(size(im));
84          for n = 1:chan
85              nim(:,:,n) = interp2(xu,yu,double(im(:,:,n)),xd,
    yd);
86          end
87      end
88
89      if isa(im, 'uint8')      % Cast back to uint8 if needed
90          nim = uint8(nim);
91      end
```

Listing 1.2: Code for blob detection

1.2 RegionProp

Next step is to measure properties of image regions like region extrema.

```
1   [labeledImage, numberOfBlobs] = bwlabel(gray);
2   blobMeasurements = regionprops(labeledImage, 'area');
```

```
3     allAreas = [blobMeasurements.Area];
4     [sortedAreas, sortIndexes] = sort(allAreas, 'descend');
5     biggestBlob1 = ismember(labeledImage, sortIndexes(1));
6     if size(sortIndexes,2)>2
7         biggestBlob2 = ismember(labeledImage, sortIndexes
      (2:3));
8     else
9         biggestBlob2 = ismember(labeledImage, sortIndexes(2))
      ;
10    end
11    biggestBlob2=imerode(biggestBlob2,strel('disk',4));
12    stats1=regionprops(biggestBlob1,'Extrema');
13    stats2=regionprops(biggestBlob2,'Extrema');
14    stats2E=[];
15    stx=[];sty=[];
16    for j=1:size(stats2,1)
17        stats2E=[stats2E;stats2(j).Extrema];
18        stx=[stx,stats2(j).Extrema(:,1)];
19        sty=[sty,stats2(j).Extrema(:,2)];
20    end
21    stx=median(stx)';
22    sty=median(sty)';
```

Listing 1.3: Code for blob detection

Figure 1.6: grayMask

Figure 1.7: BinaryImage

1.3 Polyfit and Polyval

After getting extrema of region, the next line of code will fit polynomial curve on region.

```
1
2       fit1=polyfit(stats1.Extrema(:,1),stats1.Extrema(:,2),1);
3       m1=fit1(1);
4       c1=fit1(2);
5       xlt=(500-c1)/m1;
6       xlb=(700-c1)/m1;
7       fitV1=polyval(fit1,[xlt;xlb]);
8
9       if(mod(i+2,3)==0)
10          fit2=polyfit(stx,sty,1);
11          fitV23=polyval(fit2,stx);
12          m2=fit2(1);
13          c2=fit2(2);
14          xrt=(500-c2)/m2;
15          xrb=(700-c2)/m2;
16          fitV2=polyval(fit2,[xrt;xrb]);
17      end
```

Listing 1.4: Code for blob detection

1.4 Turn Predict

The turn predict is the same as given pipeline.

```
function [dir xi yi]=turnPridict(fit1,fit2)

xi=fzero(@(x) polyval(fit1-fit2,x),3);
yi = polyval(fit1,xi);

diff = xi-640;

if diff <= -15
    dir='left';
elseif diff > -15 && diff<15
    dir='stright';
else
    dir='right';
end

end
```

Listing 1.5: Code for blob detection

1.5 Output format

```
    red=poly2mask([xlt,xlb,xrb,xrt
],[500,700,700,500],720,1280); mainframe(:,:,1)=imadd(
double(mainframe(:,:,1)),double(100*red));
    figure(1);
    imshow(mainframe);hold on;
    title(i);
    [dir xi yi]=turnPridict(fit1,fit2);
    plot(xi,420,'r*');
    dir
    plot([xlt;xlb],fitV1,'LineWidth',5,'Color',[0.9 0.8
0.25]);
    plot([xrt;xrb],fitV2,'LineWidth',5,'Color',[0.9 0.8
0.25]);
    i=i+1;
    pause(1/10);
```

Listing 1.6: Code for blob detection

Figure 1.8: redMaskImage

Figure 1.9: Final Output Image