CS-532-A

Homework 1

Janmejay Mohanty

Part 2: Design a sequence of projection matrices corresponding to each frame of a "dolly zoom" capture sequence and effect.

The Dolly Zoom effect is obtained by adjusting the zoom angle of the camera in the opposite direction while moving the camera position forward or backwards. Our starting point, or origin in the image is around 4 meters from the foreground object, which displays it in a 250\*400 box. To display the foreground object in a 400\*640 box instead, we have to move the camera position forward by 1.44 meters obtained from the ratio 8/5 \* the length of projection 9 meters. As for the zoom adjustment in each step to maintain the size of the foreground object while moving forward or backwards, we have to change the values of rx and ry in the internal calibration matrix K using the formula y = f\*Y/Z where Y, Z are the initial values of [rx or ry] and [distance] respectively and y, f are the current values. The starting position of the image is now 1.44m away from the origin, so for the first video we zoom in while moving backwards from the object to return to the origin, with each step being 0.0192m obtained from 1.44 distance / 75 frames. The second video simply performs the zoom in the other direction.

Source Code:

PointCloud2Image.m

function img =PointCloud2Image (M, Sets3DRGB, viewport, filter\_size)

%% setting up output image

disp (' Initializing 2D image...');

top = viewport (1);

left = viewport (2);

h = viewport (3);

w = viewport (4);

bot = top + h +1;

right = left + w +1;

output\_image = zeros (h+1, w+1,3);

for counter = 1:numel (Sets3DRGB)

disp(' Projecting point cloud into image plane...');

% clear drawing area of current layer

canvas = zeros (bot, right,3);

% segregate 3D points from color

dataset = Sets3DRGB {counter};

P3D = dataset (1:3, :);

color = dataset (4:6, :)';

% form homogeneous 3D points (4xN)

X = [P3D; ones (1, size(P3D,2))];

% apply (3x4) projection matrix

x = M\*X;

% normalize by 3rd homogeneous coordinate

x = x ./ [x(3,:);x(3,:);x(3,:)];

% truncate image coordinates

x(1:2,:) = floor(x(1:2,:));

% determine indices to image points within crop area

i1 = x(2,:)>top;

i2 = x(1,:)>left;

i3 = x(2,:)<bot;

i4 = x(1,:)<right;

ix = i1 & i2 & i3 & i4;

% make reduced copies of image points and corresponding color

rx = x(:,ix);

rcolor = color(ix,:);

% fill canvas with corresponding color

for i=1:size(rx,2)

canvas(rx(2,i), rx(1,i), : ) = rcolor(i,:);

end

%crop canvas to desired output size

cropped\_canvas = canvas(top:top+h,left:left+w,:);

%filter individual color channel

disp(' Running 2D filters...');

for i=1:3

%median filter

%img(:,:,i)=medfilt2(img2(:,:,i),filter\_size);

%max filter

filtered\_cropped\_canvas(:,:,i)=ordfilt2(cropped\_canvas(:,:,i),25,true(5));

%no filter

%img(:,:,i)=img2(:,:,i);

end

%get indices of pixel drawn in the current canvas

drawn\_pixels = sum(filtered\_cropped\_canvas,3);

idx = drawn\_pixels~=0;

%make a 3-chanel copy of the indices

idxx(:,:,1) = idx;

idxx(:,:,2) = idx;

idxx(:,:,3) = idx;

%erase canvas drawn pixels from the output image

output\_image(idxx) = 0;

%sum current canvas on top of output image

output\_image = output\_image + filtered\_cropped\_canvas ;

end

img = output\_image;

disp('Done');

end

SampleCameraPath.m

% Sample use of PointCloud2Image(...)

%

% The following variables are contained in the provided data file:

% BackgroundPointCloudRGB,ForegroundPointCloudRGB,K,crop\_region,filter\_size

% None of these variables needs to be modified

clc

clear all

% load variables: BackgroundPointCloudRGB,ForegroundPointCloudRGB,K,crop\_region,filter\_size)

load data.mat

data3DC = {BackgroundPointCloudRGB,ForegroundPointCloudRGB};

R = eye(3);

fmove = [0 0 -0.25]';

z = ForegroundPointCloudRGB(3, 1);

st = [0,0,0];

dz= 0.5;

% Creating video file

v = VideoWriter('newfile.wmv');

% Setting Video frame rate to 15Hz

v.FrameRate = 15;

open(v)

for step=0:15

tic

fname = sprintf('SampleOutput%03d.jpg',step);

fprintf('\nGenerating %s\n',fname);

ft = step \* fmove;

scale = (z + step\*dz) / z;

st = [0;0;step\*dz];

new\_K = K \* [scale, 0, 0; 0, scale, 0; 0, 0, 1];

M = new\_K \* [R, st];

im = PointCloud2Image(M,data3DC,crop\_region,filter\_size);

imwrite(im,fname);

toc

end

for itr = 0:8

tic

img = imread(sprintf('SampleOutput%03d.jpg',itr));

writeVideo(v,img);

toc

end

close(v)

Part 1: Implementing normalising DLT algorithm for homography estimation and use bilinear interpolation for rendering output image.

For starters, we had taken four points on the original baskeball court image to select only the playing court itself, which we know is a rectangle shape.

1. [23,193] -> [0,0]

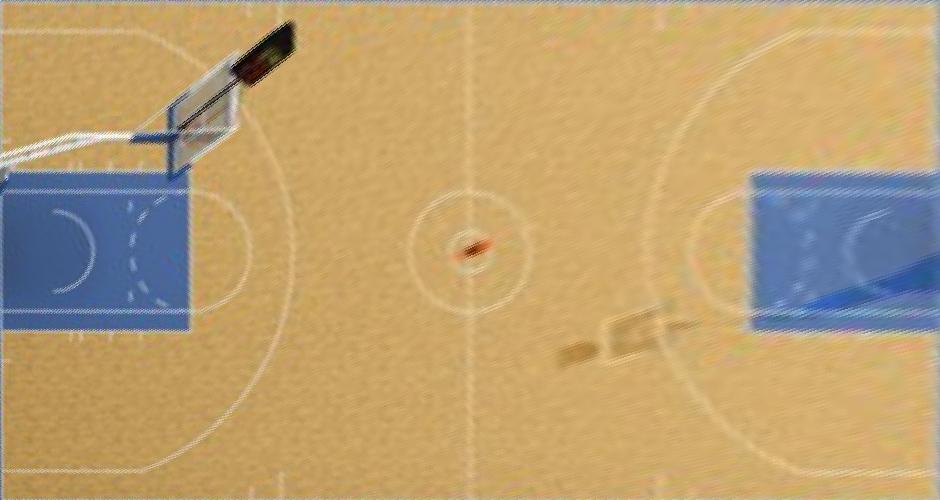
2. [247,51] -> [939,0]

3. [402,74] -> [939,499]

4. [279,279] -> [0,499]

We want to map the court to a 940\*500 canvas to portray a topdown view of the basketball court, and to do that we use the DLT algorithm to obtain the homography matrix which we then inversed as we will be using reverse warping. For each pixel in the new canvas, the corresponding pixel location is obtained using the homography matrix and if the location is between pixels, we use bilinear interpolation to sample the color of each surrounding pixel. The resulting image will give us a topdown view of the basketball court area that we selected from the original image.

Output Image:



Source Code:

import numpy as np

from PIL import Image

# Defining the image warping function

def Warping\_image(original, new, H):

(h,b,\_) = new.shape

for i in range(0,b):

for j in range(0,h):

co\_ordinates = [i,j,1]

result = np.matmul(H, co\_ordinates)

# Changing the homogeneous co\_ordinates back to regular inhomogeneous co\_ordinates

x = result[0]/result[2]

y = result[1]/result[2]

split\_x = str(x).split('.')

split\_y = str(y).split('.')

get\_i = int(split\_y[0])

get\_j = int(split\_x[0])

float\_i = float('0.'+split\_y[1])

float\_j = float('0.'+split\_x[1])

if(float\_i+float\_j <= 0.01):

new[j][i] = original[get\_i][get\_j]

else:

rgb\_values = [0,0,0]

for k in range(0,3):

result = (1-float\_i)\*(1-float\_j)\*original[get\_i][get\_j][k] + (float\_i)\*(1-float\_j)\*original[get\_i+1][get\_j][k] \

+ (float\_i)\*(float\_j)\*original[get\_i+1][get\_j-1][k] + (1-float\_i)\*(float\_j)\*original[get\_i][get\_j-1][k]

rgb\_values[k] = int(result)

new[j][i] = rgb\_values

new\_img = Image.fromarray(new)

return new\_img

# Defining The Normalize Function

def normalize(breadth,height):

Normalize = np.array([[breadth+height,0,breadth/2],[0,breadth+height,height/2],[0,0,1]])

La = np.linalg.inv(Normalize)

return La

# Defining the matrix computation function

def compute\_matrix(origin, destination):

[a,b,c] = origin

[x,y,z] = destination

matrix\_1 = np.array([[0,0,0,-a,-b,-c,y\*a,y\*b,y\*c],[a,b,c,0,0,0,-x\*a,-x\*b,-x\*c]])

return matrix\_1

image = Image.open("basketball-court.ppm")

new\_img = Image.new(mode = "RGB", size = (940, 500))

array\_1 = [[23,193,1],[247,51,1],[402,74,1],[279,279,1]]

array\_2 = [[0,0,1],[939,0,1],[939,499,1],[0,499,1]]

Top1 = normalize(488,366)

Top2 = normalize(940,500)

# Normalizing both arrays, array\_1 and array\_2

for i in range(0,4):

array\_1[i] = np.matmul(Top1,array\_1[i])

array\_2[i] = np.matmul(Top2,array\_2[i])

# Computing and generating the matrix\_1

m1 = np.zeros((8,9))

for i in range(0,4):

cm = compute\_matrix(array\_1[i],array\_2[i])

iterate = 2\*i

m1[iterate:iterate+2] = cm

++i

# Calculating and evaluating the svd and taking least eigenvector

Up, Down, Eigenvector = np.linalg.svd(m1)

Hm = Eigenvector[8]

# Reshaping into 3\*3 homography matrix

Hr = np.reshape(Hm,(3,3))

# Denormalizing the Hr component

Hd = np.matmul(np.matmul(np.linalg.pinv(Top2),Hr),Top1)

# Now taking the homography matrix inverse because all the implementation are in reverse warping

Hla = np.linalg.inv(Hd)

new\_img = Warping\_image(np.array(image),np.array(new\_img),Hla)

new\_img.save("New\_Basketball\_Court\_Image.jpg")

new\_img.show()