Computer Vision

LAB 4

Submitted By:

Gaurav Agarwal	BT17CSE001
Abhibha Gupta	BT17CSE020
Thogaru Himabindu	BT17CSE056

Submitted to:

Dr. Mayur Parate

Task:

GENERATE 3D POINT CLOUD USING BINOCULAR VISION I.E., USING 2 CAMERAS

भारतीय सूचना प्रौद्योगिकी संस्थान, नागपूर

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, NAGPUR

(An Institution of National Importance by Act of Parliament)

BSNL RTTC, Near TV Tower, Besides Balaji Temple, Seminary Hills, Nagpur-440006

GENERATE 3D POINT CLOUD USING BINOCULAR VISION I.E., USING 2 CAMERAS

```
import cv2
import numpy as np
```

Function to Get the output by clicking on the depth map

```
def click_event(event, x, y, flags, param):
     if event == cv2.EVENT_LBUTTONDOWN:
          print ((x, y), float(235.298/displ[y,x]))
```

The below image pair is rectified by cv2.remap and read below

```
imgR = cv2.imread(
'/home/bindu/Sem8/Cv/Lab3/3dPoint/3DReconstruction-master/Calibration/RightRect
.jpeg')
imgL = cv2.imread(
'/home/bindu/Sem8/Cv/Lab3/3dPoint/3DReconstruction-master/Calibration/LeftRect.
jpeg')

imgL=cv2.resize(imgL,(640,480))
imgR=cv2.resize(imgR,(640,480))

imgL=cv2.cvtColor(imgL, cv2.COLOR_BGR2GRAY)
imgR = cv2.cvtColor(imgR, cv2.COLOR_BGR2GRAY)
```

To calculate the disparity map, a stereoSGBM object is generated with the function cv2.StereoSGBM_create (). This class uses a semi-global box matching algorithm to obtain a stereo match between the images from the right camera and the left.

```
window_size = 3
min_disp = 0
num_disp = 128-min_disp
```

```
matcher_left = cv2.StereoSGBM_create(
   blockSize = 5,
   numDisparities = 5*16,
   minDisparity = min_disp ,
   P1 = 8*3*window_size,
   P2 = 32*3*window_size,
   # disp12MaxDiff = 12,
   # uniquenessRatio = 10,
   # speckleWindowSize = 50,
   # speckleRange = 32, preFilterCap = 63,
   mode=cv2.STEREO_SGBM_MODE_SGBM_3WAY)
```

Parameters:

minDisparity :

Minimum possible disparity value. Normally, it is zero but sometimes rectification algorithms can shift images, so this parameter needs to be adjusted accordingly.

numDisparities :
Maximum disparity minus minimum disparity. The value is always greater than zero. In the current implementation, this parameter must be divisible by 16.
blockSize Matched block size. It must be an odd number >=1 . Normally, it should be somewhere in the 3..11 range.

P1 :
The first parameter controlling the disparity smoothness.This parameter is used for the case of slanted surfaces (not fronto parallel).

P2 :
The second parameter controlling the disparity smoothness.This parameter is used for "solving" the depth discontinuities problem. The larger the values are, the smoother the disparity is.

P1 is the penalty on the disparity change by plus or minus 1 between neighbor pixels.
P2 is the penalty on the disparity change by more than 1 between neighbor pixels

preFilterCap :
Truncation value for the prefiltered image pixels. The algorithm first computes x-derivative at each pixel and clips its value by [-preFilterCap, preFilterCap] interval. The result values are passed to the Birchfield-Tomasi pixel cost function.

uniquenessRatio :
Margin in percentage by which the best (minimum) computed cost function value should "win" the second best value to consider the found match correct.

speckleWindowSize :
Maximum size of smooth disparity regions to consider their noise speckles and invalidate. Set it to 0 to disable speckle filtering. Otherwise, set it somewhere in the 50-200 range.

speckleRange :
Maximum disparity variation within each connected component. If you do speckle filtering, set the parameter to a positive value, it will be implicitly multiplied by 16. Normally, 1 or 2 is good enough.

mode :
Set it to StereoSGBM::MODE_HH to run the full-scale two-pass dynamic programming algorithm. It will consume O(W*H*numDisparities) bytes, which is large for 640x480 stereo and huge for HD-size pictures. By default, it is set to false.

```
matcher_right = cv2.ximgproc.createRightMatcher(matcher_left)

cv2.namedWindow("Disparity")
```

Parameters:

lambda :
parameter defining the amount of regularization
sigma_color :
parameter, that is similar to color space sigma in bilateralFilter.

```
lmbda = 80000
```

```
sigma = 1.3
                        Callback function for mouse events

visual_multiplier = cv2.setMouseCallback("Disparity", click_event)

create instance of DisparityWLSFilter and execute basic initialization routines

wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=matcher_left)
wls_filter.setLambda(lmbda)

wls_filter.setSigmaColor(sigma)

displ = matcher_left.compute(imgL, imgR) .astype(np.float32)
displ = np.int16(displ)
dispr = matcher_right.compute(imgR, imgL) .astype(np.float32)
dispr = np.int16(dispr)

                                    WLS FILTER
It is still very difficult to recognize the edges of objects because of the
noise, so a WLS filter is used

filteredImg = wls_filter.filter(displ, imgL, None, dispr)
filteredImg = cv2.normalize( src=filteredImg,
dst=filteredImg,beta=1,alpha=255,norm_type=cv2.NORM_MINMAX,dtype=cv2.CV_8U)
filteredImg = np.uint8(filteredImg)
displ= ((displ.astype(np.float32)/ 16)-min_disp)/num_disp
#Calculation allowing us to have 0 for the most distant object able to detect


                                    Removing Noise:
      Apply an morphological filter for closing little "black" holes in the
                                    picture

k= np.ones((3,3),np.uint8)
closing= cv2.morphologyEx(displ,cv2.MORPH_CLOSE, k)

                                    Colors map

dispc= (closing-closing.min())*255
```

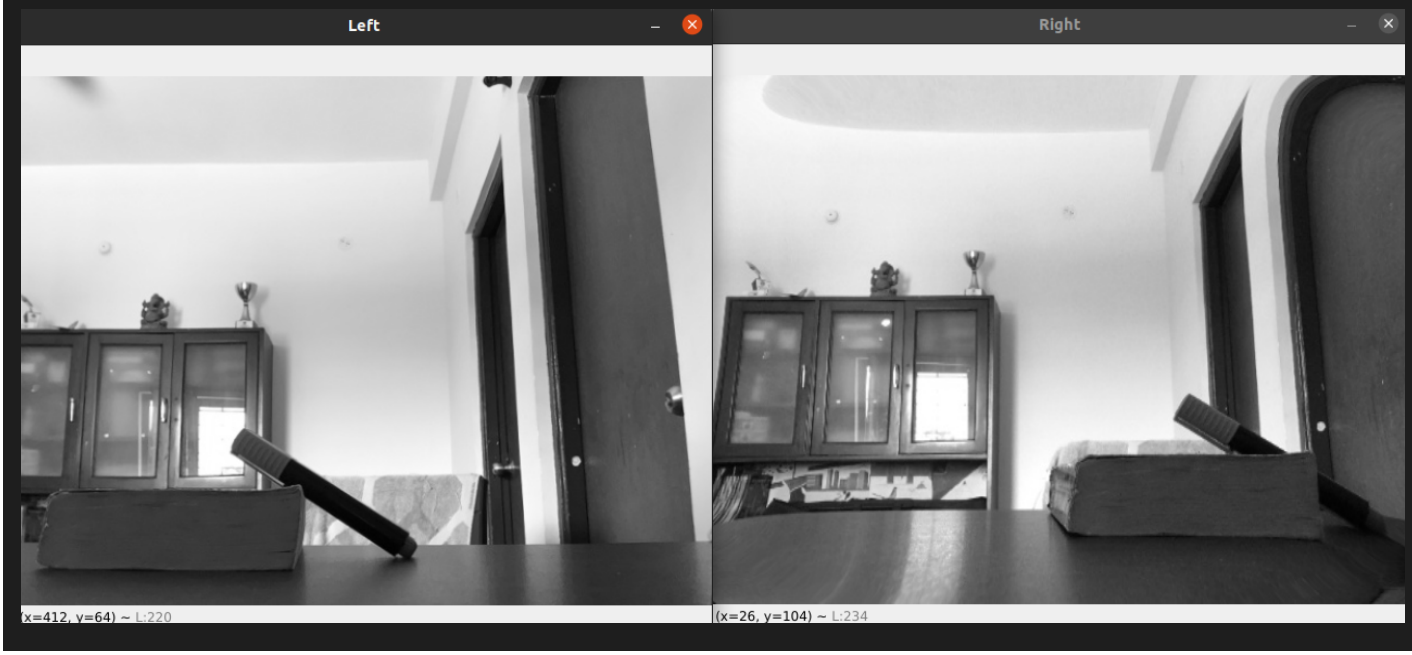
Convert the type of the matrix from float32 to uint8, this way you can show the results with the function cv2.imshow()

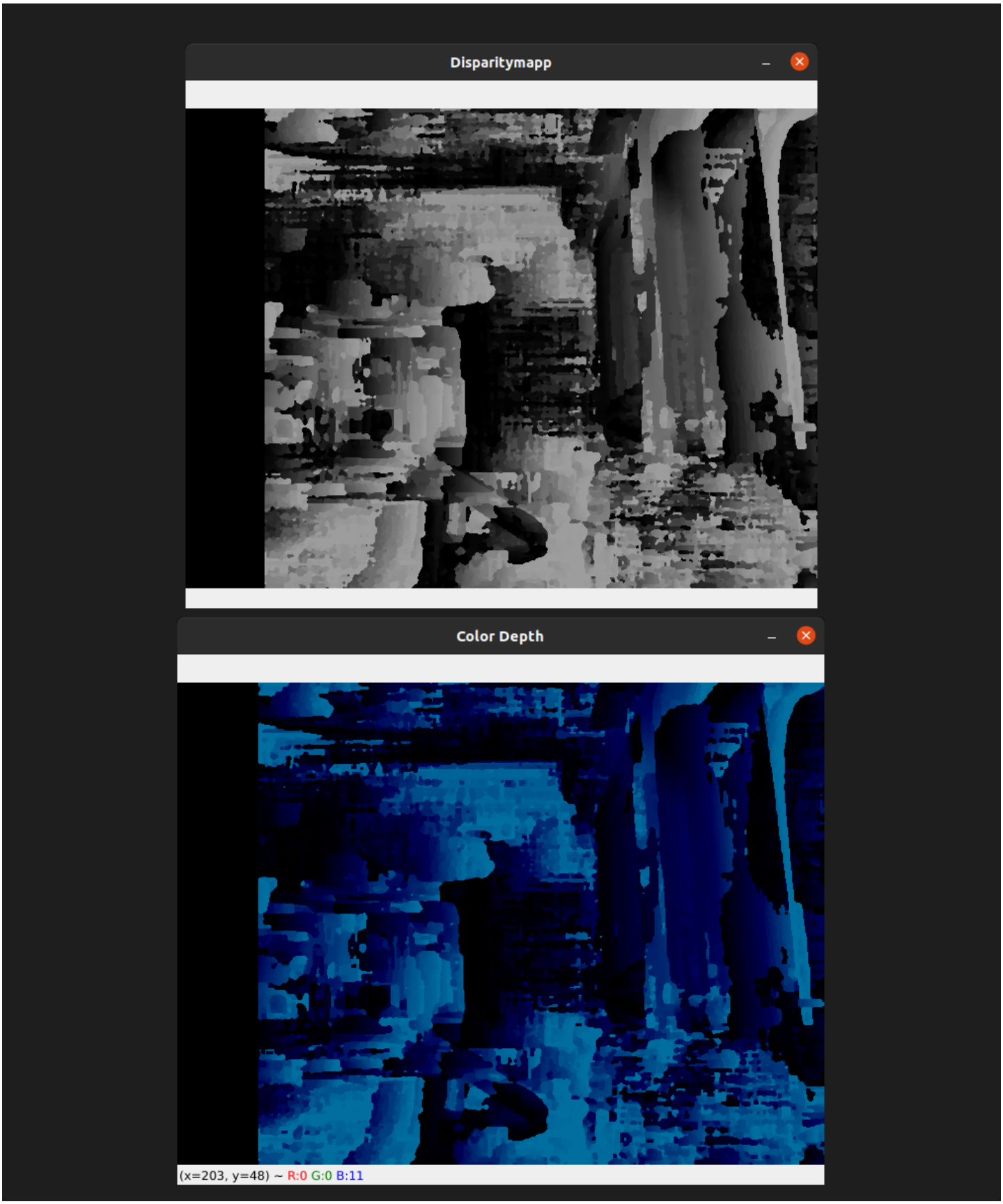
```
dispC= dispc.astype(np.uint8)
```

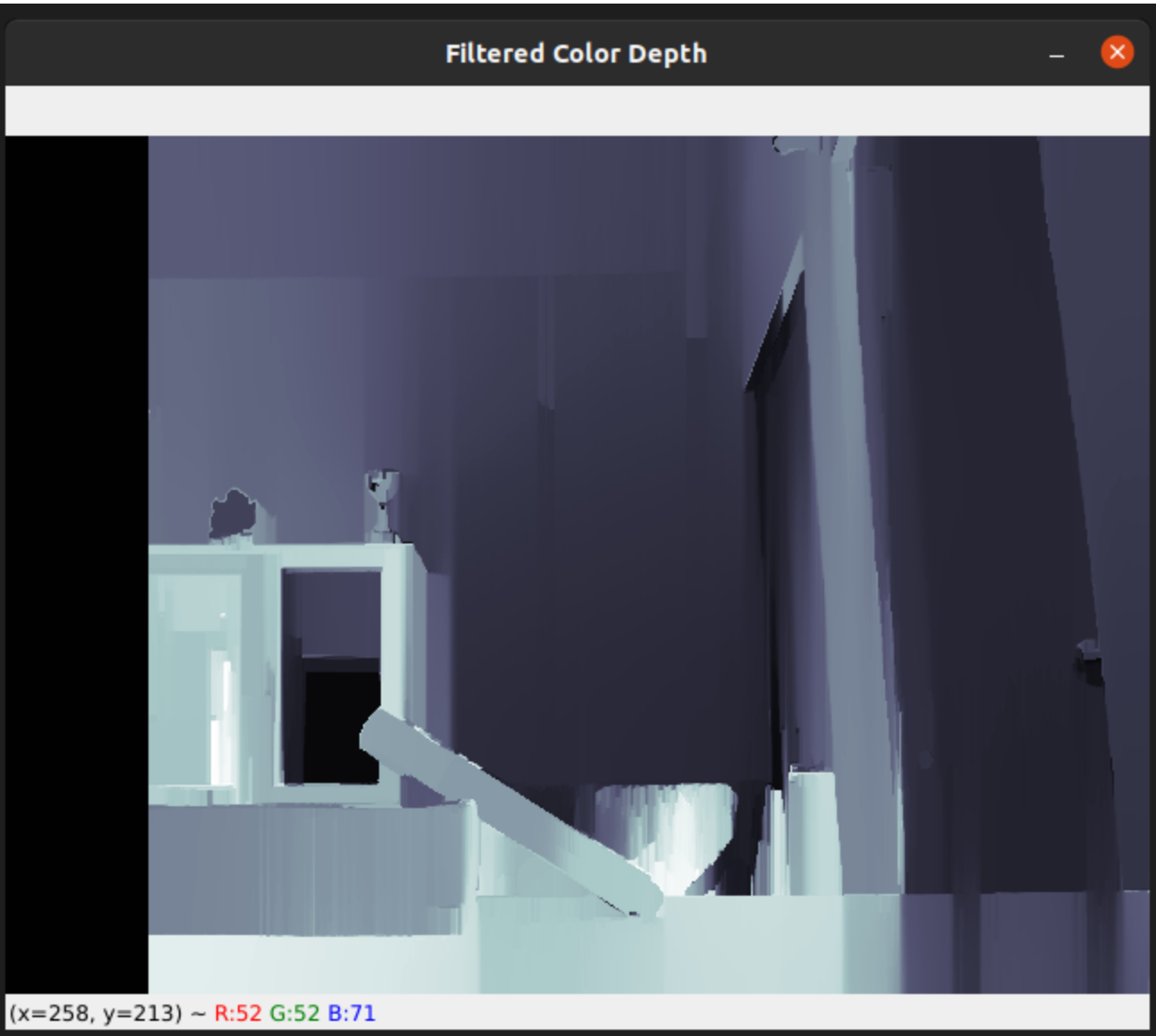
Change the Color of the Picture with cv2.COLORMAP_*

```
disp_Color= cv2.applyColorMap(dispC,cv2.COLORMAP_OCEAN)
filt_Color= cv2.applyColorMap(filteredImg,cv2.COLORMAP_BONE)

cv2.imshow('Disparitymapp', displ)
cv2.imshow('Left', imgL)
cv2.imshow('Right', imgR)
cv2.imshow('Closing',closing)
cv2.imshow('Color Depth',disp_Color)
cv2.imshow('Filtered Color Depth',filt_Color)
cv2.imshow('Disparity',filt_Color)
cv2.setMouseCallback("Disparity", click_event)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Gray Scale Left and Right Rectified Images:




Disparitymapp
Color Depth
(x=203, y=48) ~ R:0 G:0 B:11



When clicked on filtered depth map, we obtain depth from the cameras to corresponding onclick coordinate.Some of them are shown below:

To measure the distance to objects, the coordinates x and y of the WLS filtered image are taken with a double click. These (x, y) coordinates are then used to get the disparity value from the disparity map and then to measure the distance. The returned value is the mean value of the disparity of a 9x9 pixel matrix.

(X,Y) Coordinate and Corresponding depth from Cameras in mm
(300, 383) 771.0244864
(209, 336) 386.74984269662923

```
(290, 380) 1068.4929135254988
(307, 389) 470.596
(439, 183) 1882.384
(428, 142) 2769.4845057471266
(412, 403) 501.446726326743
(386, 417) 381.24232911392403
(278, 465) 778.4980678513732
(306, 472) 520.3998963282937
(359, 475) 429.8753826940232
(435, 457) 397.5992607260726
(121, 465) 402.91831438127093
(121, 465) 402.91831438127093
(97, 437) 485.77651612903225
(183, 422) 381.24232911392403
(89, 471) 456.76806066350713
(114, 475) 416.14015889464594
(139, 476) 454.1850179076343
(160, 474) 778.4980678513732
(168, 473) 787.4024575163398
(183, 472) 560.9898766006985
(211, 469) 418.30755555555555
```

Extra Work:

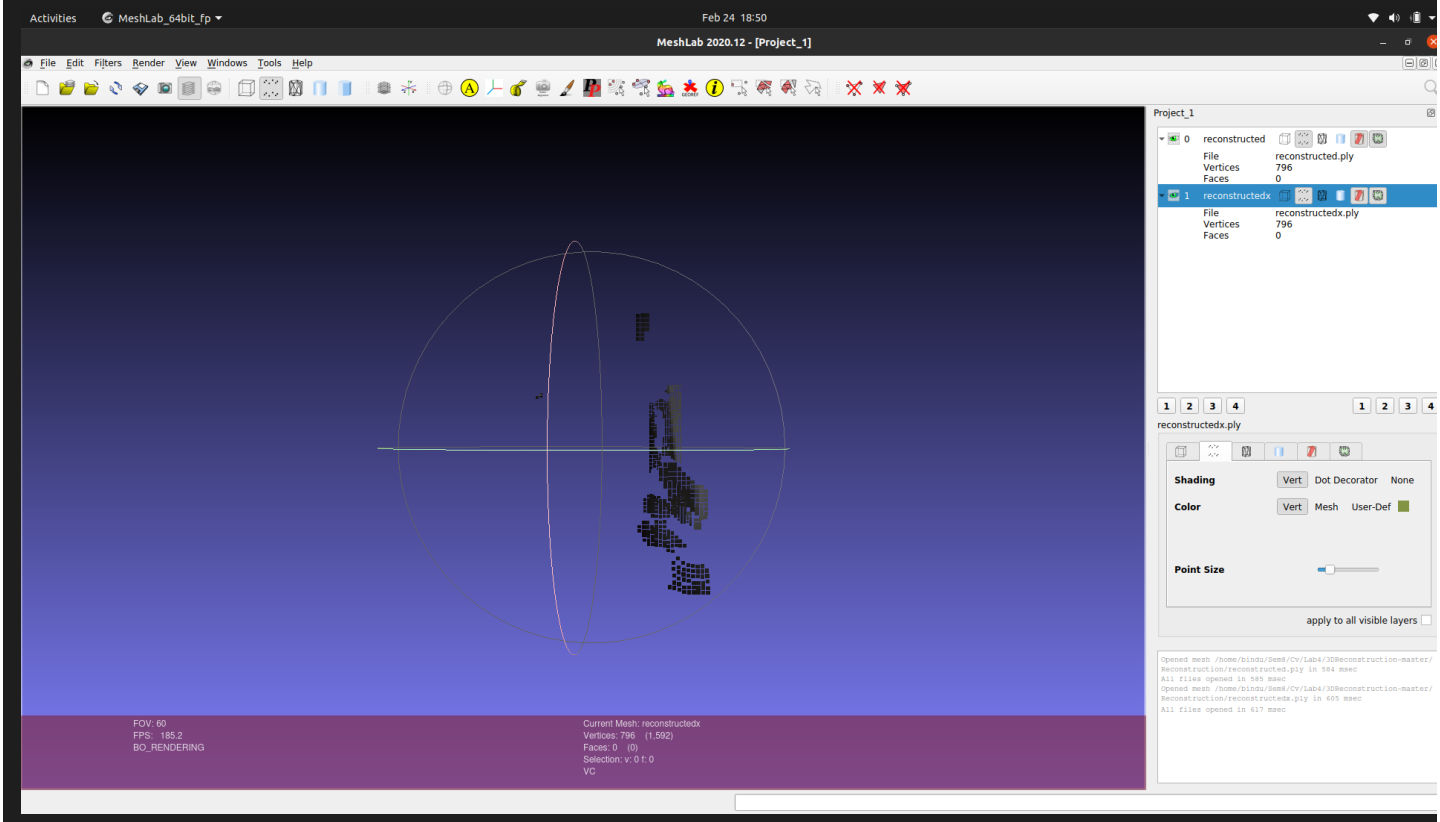
We can measure the disparity values experimentally at many points to determine the regression curve. And the distance can be determined easily.I have not included this part of the code.

Create 3d point Cloud:

```
#Function to create point cloud file
def create_output(vertices, colors, filename):
      colors = colors.reshape(-1,3)
      vertices = np.hstack([vertices.reshape(-1,3),colors])

      ply_header = '''ply
            format ascii 1.0
            element vertex %(vert_num)d
            property float x
            property float y
            property float z
            property uchar red
```

```
            property uchar green
            property uchar blue
            end_header
            '''
      with open(filename, 'w') as f:
            f.write(ply_header %dict(vert_num=len(vertices)))
            np.savetxt(f,vertices,'%f %f %f %d %d %d')

#Function that Downsamples image x number (reduce_factor) of times.
def downsample_image(image, reduce_factor):
      for i in range(0,reduce_factor):
            #Check if image is color or grayscale
            if len(image.shape) > 2:
                  row,col = image.shape[:2]
            else:
                  row,col = image.shape

            image = cv2.pyrDown(image, dstsize= (col//2, row // 2))
      return image


#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be
usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")

#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]

#Load focal length.
focal_length =
np.load('/home/bindu/Sem8/Cv/Lab3/3dPoint/3DReconstruction-master/Calibration/FocalLeng
thh.npy',allow_pickle=True)
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation.
Q = np.float32([[1,0,0,-w/2.0],
```

```
                        [0,-1,0,h/2.0],
                        [0,0,0,-focal_length],
                        [0,0,1,0]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()

#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]

#Define name for output file
output_file = 'reconstructed.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)
```



Reconstructed.ply can be visualised by MeshLab
The 3d point cloud is shown above. It should be refined.