**Coursework 3 Report**

**Usman Ali**

* 1. **Calculate Focal Length**

The focal length was calculated using the width of the camera sensor size and dividing this by width of the resolution. The resulting number was then multiplied by the focal length in pixels.

Fmm = Fpx x (Sw / Iw)

Fmm = 5806.559 x (22.2 / 3088)

Fmm = 41.74 mm (2dp)

**1.2 Disparity Map A screenshot of a computer

Description automatically generated**

Figure 1 Disparity Image with blocksize of 5 and numOfDisparities at 64

A screenshot of a computer

Description automatically generated

Figure 2 Disparity Image with blocksize of 15 and numOfDisparities at 64

A screenshot of a computer

Description automatically generated

Figure 3 Disparity Image with blocksize of 5 and numOfDisparities at 16

A screenshot of a computer

Description automatically generated

Figure 4 Disparity Image with blocksize of 5 and numOfDisparities at 128

The **blockSize** defines the size of the comparison window used to find the corresponding points between the left and right images. The robustness of disparity calculations can be enhanced by increasing the **blockSize** because more pixels are needed to establish correspondence. The result of this change can be seen when comparing Figure 2 with Figure 1. Figure 2 has a larger **blockSize**, and as a result the details in the image, represented by the thicker lines, are a lot smoother and there is less noise present. Although a smaller **blockSize** can retain more detail and identify edges and small objects more accurately, it may struggle in low-textured areas and be more vulnerable to noise.

The number of disparities represents the number of horizontal shifts that the algorithm will test to find correspondences between the left and right images. The depth range that the system is able to detect is directly affected by this parameter. A comparison between Figure 3 (disparity parameter set at 16) and Figure 4 (disparity parameter set at 128) allows us to see the effect of increasing the number of disparities. This increase results in a disparity map that is more detailed and captures a wider range of depth. In contrast, the disparity map seems less detailed and only the most significant shifts are depicted when the number of disparities is set to a smaller value, such as 16. These larger or closer objects typically have more pronounced depth variations from the background.

A screenshot of a graph

Description automatically generated

A screenshot of a computer

Description automatically generated

Figure 5

A screenshot of a computer

Description automatically generated

Figure 6

A screenshot of a computer

Description automatically generated

Figure 7

A person in a dress

Description automatically generatedA person in a dress

Description automatically generated

Figure 8

depth = 1/ (disparity + k)

In the depth calculation formula, the variable “k" serves as a control to avoid division by zero and to adjust how disparity changes affect the depth values that are obtained. A smaller k value increases the depth estimation's responsiveness to disparity alterations, a greater k value ensures a smoother depth map and mitigates noise.

**Appendix**

import numpy as np

import cv2

import sys

from mpl\_toolkits import mplot3d

from matplotlib import pyplot as plt

# ================================================

def getDisparityMap(imL, imR, numDisparities, blockSize):

    stereo = cv2.StereoBM\_create(numDisparities=numDisparities, blockSize=blockSize)

    disparity = stereo.compute(imL, imR)

    disparity = disparity - disparity.min() + 1 # Add 1 so we don't get a zero depth, later

    disparity = disparity.astype(np.float32) / 16.0 # Map is fixed point int with 4 fractional bits

    return disparity # floating point image

# ================================================

# ================================================

def plot(disparity, f, cx, cy, baseline, doffs):

    h, w = disparity.shape

    x, y = np.meshgrid(np.arange(w), np.arange(h))

    # Calculate Z, X, and Y for each pixel

    Z = (baseline \* f) / (disparity.astype(np.float32) + doffs)

    X = (x - cx) \* Z / f

    Y = (y - cy) \* Z / f

    X = X.flatten()

    Y = Y.flatten()

    Z = Z.flatten()

    # Calculate the maximum value of Z and set the threshold to 98% of this value

    max\_Z\_value = np.max(Z)

    Z\_threshold = 0.98 \* max\_Z\_value

    # Filter out points where Z is above the threshold

    mask = Z < Z\_threshold

    X\_filtered = X[mask]

    Y\_filtered = Y[mask]

    Z\_filtered = Z[mask]

    fig = plt.figure(figsize=(15, 5))

    # 3D plot

    ax1 = fig.add\_subplot(131, projection='3d')

    ax1.scatter(X\_filtered, Y\_filtered, Z\_filtered, c='green', marker='.')

    ax1.set\_xlabel('X')

    ax1.set\_ylabel('Y')

    ax1.set\_zlabel('Z')

    ax1.set\_xlim([-2000, 2000])

    ax1.set\_ylim([-2000, 2000])

    ax1.set\_zlim([8000, Z\_threshold])

    ax1.title.set\_text('3D View')

    # Top view (x,z)

    ax2 = fig.add\_subplot(132)

    ax2.scatter(X\_filtered, Z\_filtered, c='red', marker='.')

    ax2.set\_xlabel('X')

    ax2.set\_ylabel('Z')

    ax2.set\_xlim([-2000, 2000])

    ax2.set\_ylim([8500, Z\_threshold])

    ax2.title.set\_text('Top View (X,Z)')

    # Side view (y,z)

    ax3 = fig.add\_subplot(133)

    ax3.scatter(Y\_filtered, Z\_filtered, c='blue', marker='.')

    ax3.set\_xlabel('Y')

    ax3.set\_ylabel('Z')

    ax3.set\_xlim([-2000, 2000])

    ax3.set\_ylim([8500, Z\_threshold])

    ax3.title.set\_text('Side View (Y,Z)')

    plt.tight\_layout()

    plt.show()

def calc\_focal\_length(f\_px, sw ,iw):

    f\_mm = f\_px \* (sw / iw)

    return f\_mm

focal\_length = calc\_focal\_length(5806.559, 22.2, 3088)

print(focal\_length)

# Global variables to store the trackbar values

numDisparities = 64

blockSize = 5

def on\_trackbar\_disp(val):

    # global numDisparities

    if val % 16 == 0 and val != 0:

        numDisparities = val

    disparity = getDisparityMap(edgesL, edgesR, numDisparities, blockSize)

    # disparity = getDisparityMap(imgL, imgR, numDisparities, blockSize)

    disparityImg = np.interp(disparity, (disparity.min(), disparity.max()), (0.0, 1.0))

    cv2.imshow('Disparity', disparityImg)

def on\_trackbar\_block(val):

    #global blockSize

    blockSize = val

    blockSize = max(5, blockSize + (blockSize % 2 == 0))

    disparity = getDisparityMap(edgesL, edgesR, numDisparities, blockSize)

    # disparity = getDisparityMap(imgL, imgR, numDisparities, blockSize)

    disparityImg = np.interp(disparity, (disparity.min(), disparity.max()), (0.0, 1.0))

    cv2.imshow('Disparity', disparityImg)

f\_original = 5806.559  # Focal length in pixels for the original image size

cx\_original = 1429.219  # The x-coordinate of the principal point for the original image size

cy\_original = 993.403  # The y-coordinate of the principal point for the original image size

doffs\_original = 114.291  # The x-difference of the principal points

baseline = 174.019  # The camera baseline in millimeters

# Original and resized image dimensions

width\_original, height\_original = 2960, 2016

width\_resized, height\_resized = 740, 505

scale\_factor = width\_resized / width\_original

f = scale\_factor \* f\_original

cx0 = scale\_factor \* cx\_original

cy = scale\_factor \* cy\_original

doffs = scale\_factor \* doffs\_original

if \_\_name\_\_ == '\_\_main\_\_':

    # Load left image

    filename = 'umbrellaL.png'

    imgL = cv2.imread(filename, cv2.IMREAD\_GRAYSCALE)

    if imgL is None:

        print('\nError: failed to open {}.\n'.format(filename))

        sys.exit()

    edgesL = cv2.Canny(imgL, 50, 150)

    # Load right image

    filename = 'umbrellaR.png'

    imgR = cv2.imread(filename, cv2.IMREAD\_GRAYSCALE)

    if imgR is None:

        print('\nError: failed to open {}.\n'.format(filename))

        sys.exit()

    edgesR = cv2.Canny(imgR, 50, 150)

    # Initialize the disparity settings

    numDisparities = 64

    blockSize = 5

    cv2.namedWindow('Disparity', cv2.WINDOW\_NORMAL)

    cv2.createTrackbar('numDisparities', 'Disparity', numDisparities, 256, on\_trackbar\_disp)

    cv2.createTrackbar('blockSize', 'Disparity', blockSize, 50, on\_trackbar\_block)

    # Recalculate the disparity map with the new parameters

    disparity = getDisparityMap(edgesL, edgesR, numDisparities, blockSize)

    disparityImg = np.interp(disparity, (disparity.min(), disparity.max()), (0.0, 1.0))

    cv2.imshow('Disparity', disparityImg)

    plot(disparityImg, f, cx0, cy, baseline, doffs)

    while True:

        key = cv2.waitKey(1)

        if key == ord(' ') or key == 27:

            break

    cv2.destroyAllWindows()

File 2:

def on\_trackbar\_k(val):

    # global k

    k = val/100

    # Recalculate the disparity map with the new parameters

    disparity = getDisparityMap(imgL, imgR, numDisparities, blockSize)

    depth\_map = cv2.divide(1.0, cv2.add(disparity, k))

    disparityImg = np.interp(depth\_map, (disparity.min(), disparity.max()), (0.0, 1.0))

    cv2.imshow('Disparity', disparityImg)

def process\_background(image, depth\_map, k, blur=False, grayscale=False):

    # Threshold the depth map to create a mask for the foreground

    \_, foreground\_mask = cv2.threshold(depth\_map, k, 255, cv2.THRESH\_BINARY)

    foreground\_mask = foreground\_mask.astype(np.uint8)

    foreground\_mask\_3c = cv2.cvtColor(foreground\_mask, cv2.COLOR\_GRAY2BGR)

    foreground = cv2.bitwise\_and(image, foreground\_mask\_3c)

    background\_mask = cv2.bitwise\_not(foreground\_mask)

    background\_mask\_3c = cv2.cvtColor(background\_mask, cv2.COLOR\_GRAY2BGR)

    background = cv2.bitwise\_and(image, background\_mask\_3c)

    if blur:

        foreground = cv2.GaussianBlur(foreground, (21, 21), 0)

    if grayscale:

        foreground = cv2.cvtColor(foreground, cv2.COLOR\_BGR2GRAY)

        foreground = cv2.cvtColor(foreground, cv2.COLOR\_GRAY2BGR)

    combined\_image = cv2.add(foreground, background)

    return combined\_image

if \_\_name\_\_ == '\_\_main\_\_':

    # Load left image

    filename = 'girlL.png'

    imgL = cv2.imread(filename, cv2.IMREAD\_GRAYSCALE)

    if imgL is None:

        print('\nError: failed to open {}.\n'.format(filename))

        sys.exit()

    # Load right image

    filename = 'girlR.png'

    image = cv2.imread(filename, cv2.IMREAD\_COLOR)

    imgR = cv2.imread(filename, cv2.IMREAD\_GRAYSCALE)

    if imgR is None:

        print('\nError: failed to open {}.\n'.format(filename))

        sys.exit()

    numDisparities = 64

    blockSize = 45

    k2 = 10

    k = 0.99

    # Create a window to display the image in

    cv2.namedWindow('Disparity', cv2.WINDOW\_NORMAL)

    cv2.createTrackbar('numDisparities', 'Disparity', numDisparities, 256, on\_trackbar\_disp)

    cv2.createTrackbar('blockSize', 'Disparity', blockSize, 50, on\_trackbar\_block)

    cv2.createTrackbar('k', 'Disparity', k2, 100, on\_trackbar\_k)

    disparity = getDisparityMap(imgL, imgR, numDisparities=numDisparities, blockSize=blockSize)

    disparityImg = np.interp(disparity, (disparity.min(), disparity.max()), (0.0, 1.0))

    depth\_map = cv2.divide(1.0, cv2.add(disparityImg, k))

    new\_image = process\_background(image, depth\_map, k, blur=False, grayscale=True)

    cv2.imshow('Image', new\_image)

    while True:

        key = cv2.waitKey(1)

        if key == ord(' ') or key == 27:

            break