COMP37212: Computer Vision Coursework 3

Horia Radu - k55592hr

1. Stereo Imagery

1.1. Focal Length Calculation

For the focal length calculations of the cameras, I used the following formula:

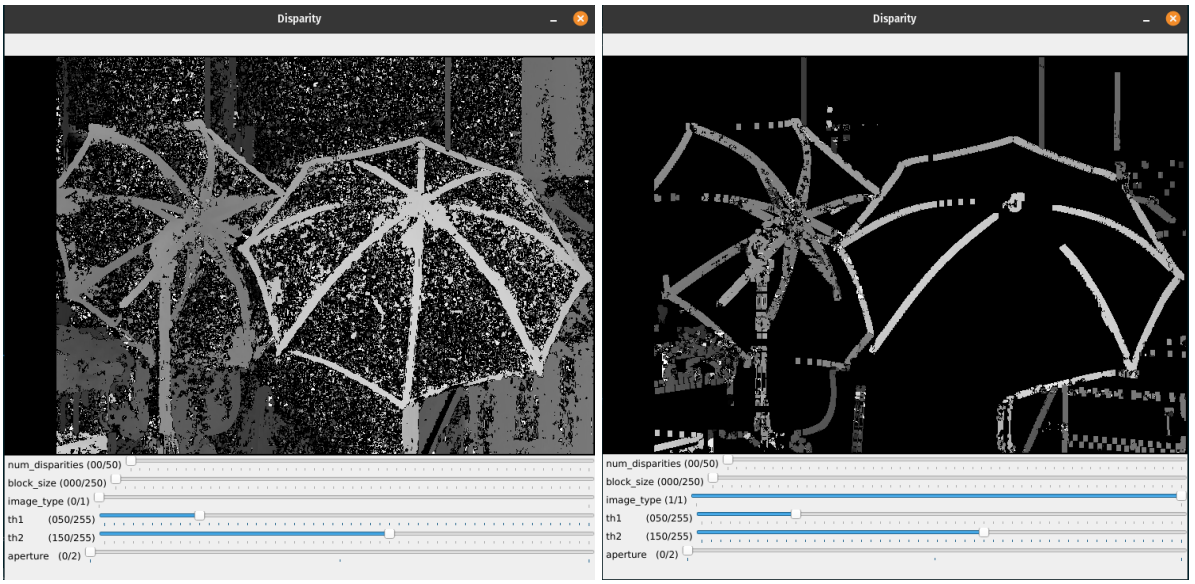
Focal Length (mm) = Focal Length (pixels) * Sensor Size / Camera Resolution

By using the 22.2 mm × 14.8 mm sensor size, 3088 × 2056 resolution, and 5806.559 focal length in pixels I got the results of ~41.75 Focal Length in mm for cam0 and cam1, rounded to **42 mm**, both of them being the same camera.

- cam0 x focal length in mm: 41.744044624352334
- cam0 y focal length in mm: 41.79818735408561
- cam1 x focal length in mm: 41.744044624352334
- cam1 y focal length in mm: 41.79818735408561

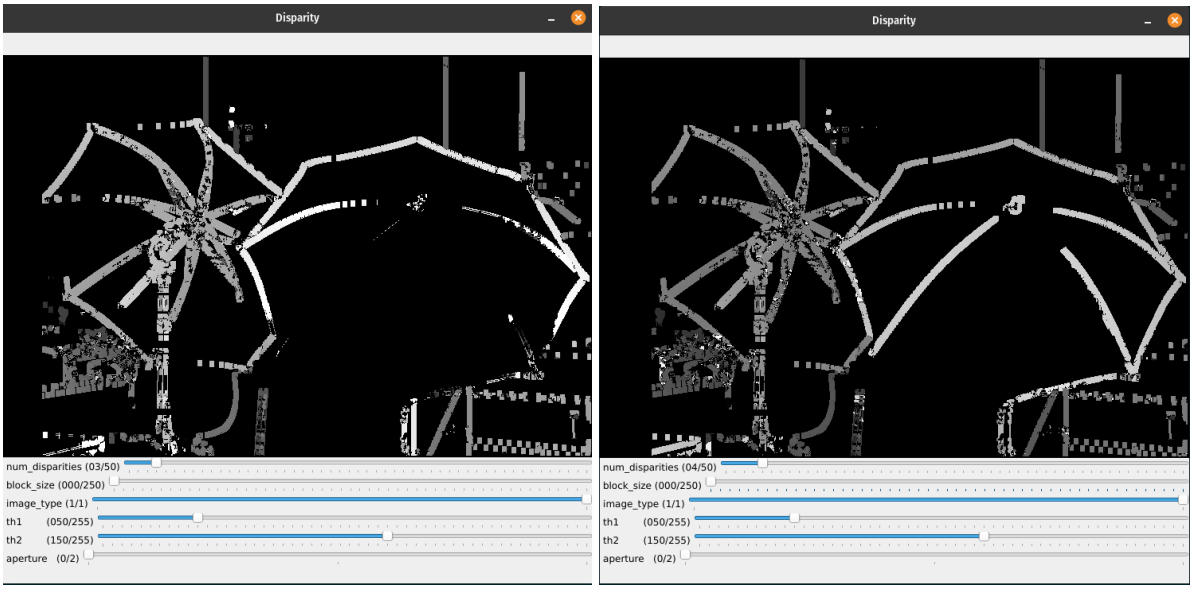
1.2. Disparity Map

I tested the disparity map with both the greyscale images and the edge only images, made using the Canny edge detector. Better results were obtained using the edge detected images.

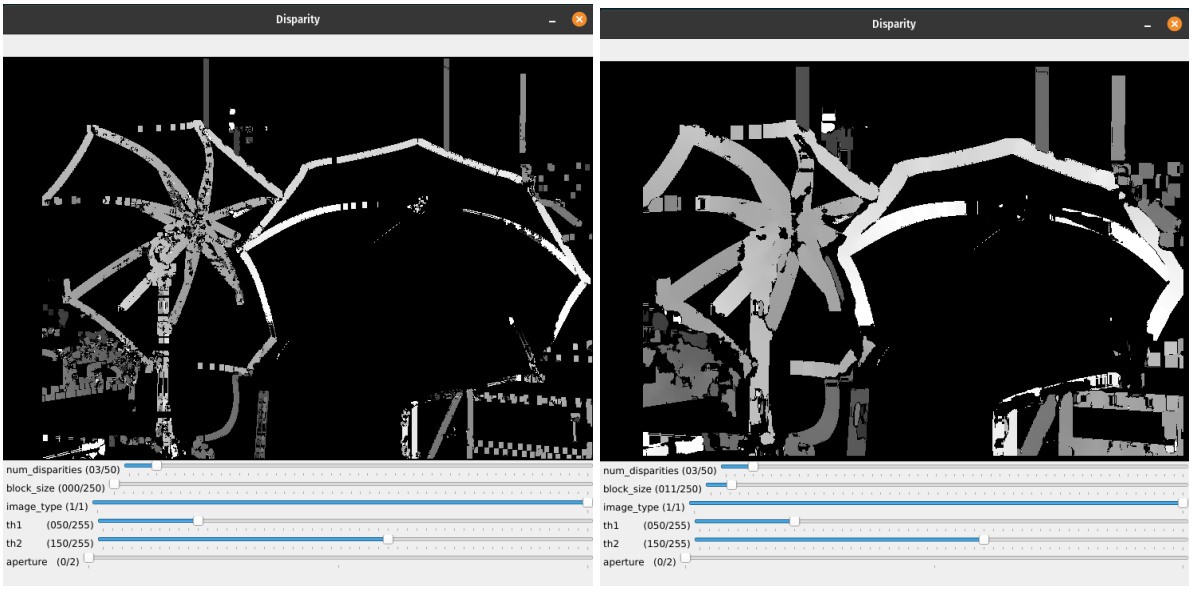


Greyscale image – Edge only image

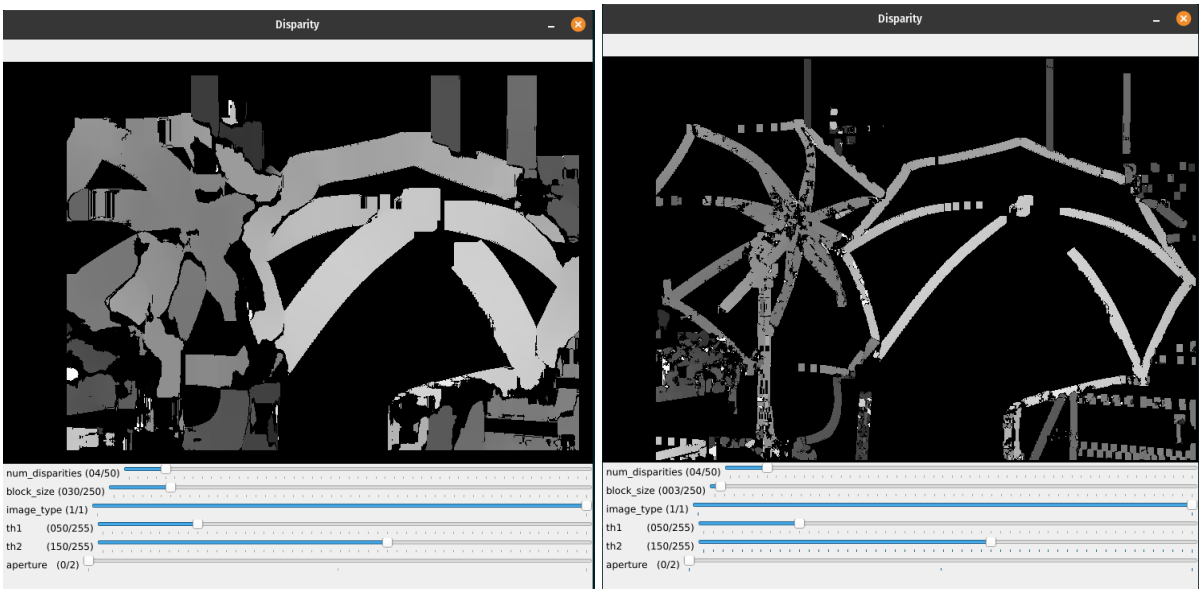
To make sure I used the optimal number of disparities and block size parameters, I created Trackbars for real time variations of those parameters. I have also added parameters for the Canny edge detector to find out the best ones.



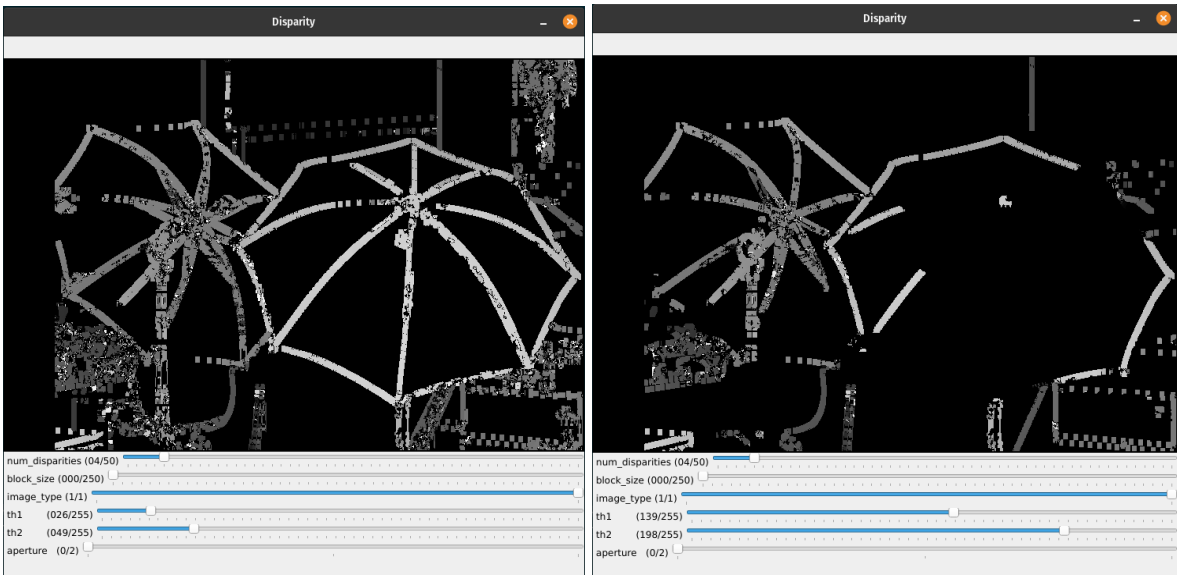
48 Number Disparities (N.D.) and 5 Block Size (B.S.) – 64 N.D. and 5 B.S.



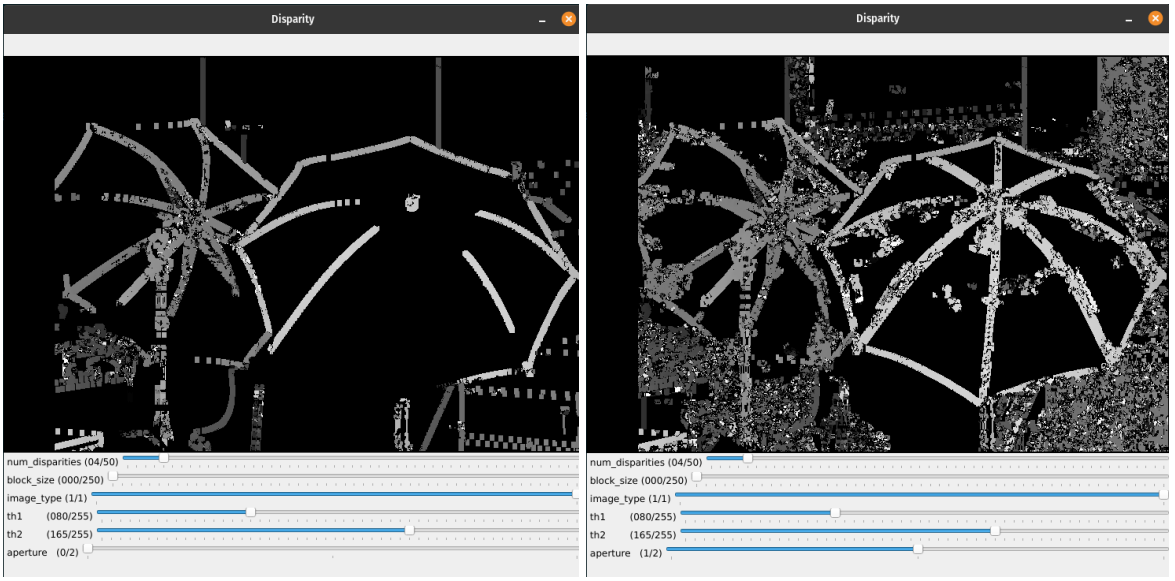
48 N.D. and 5 B.S. – 48 N.D. and 15 B.S.



64 N.D. and 35 B.S. – 64 N.D. and 7 B.S.



64 N.D. and 5 B.S. lower Canny thresholds – 64 N.D. and 5 B.S. higher Canny thresholds

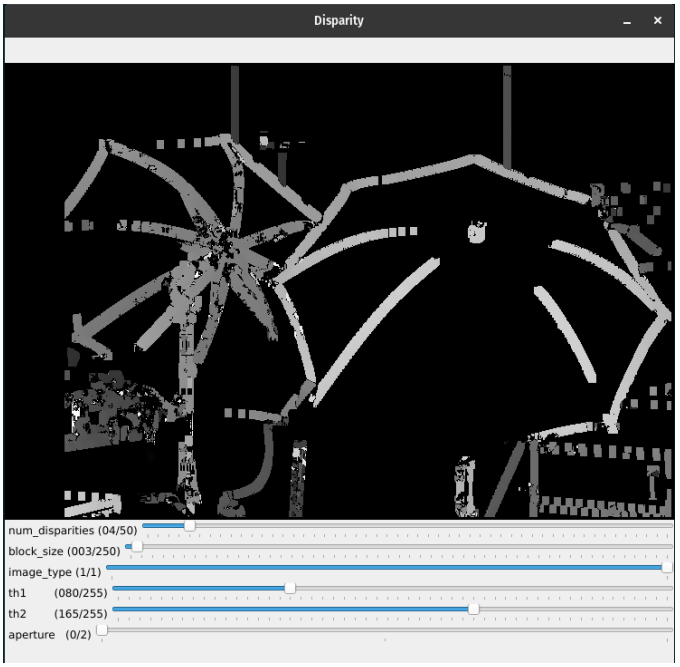


64 N.D. and 5 B.S. aperture size 3 – 64 N.D. and 5 B.S. aperture size 5

The most optimal choice of parameters I have found is:

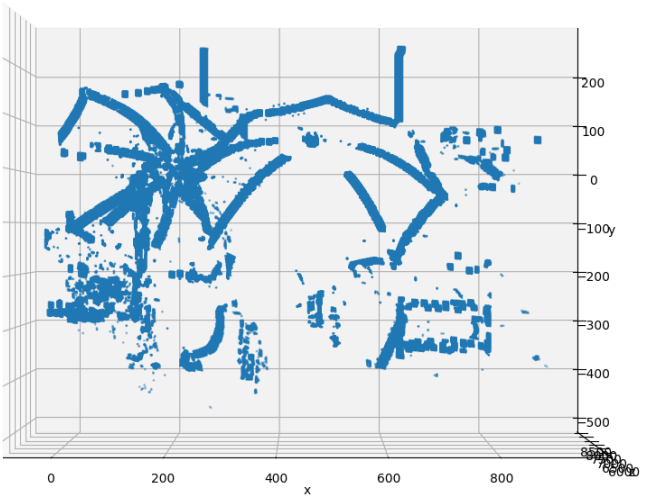
- Number of disparities: 64
- Block size: 7, but is also viable between 5 and 15 (higher block size reduces the noise and fills incomplete edge differences, but looses some accuracy and precision of each disparity). More options are good, it depends on your use case.
- Min threshold: 80
- Max threshold 165
- Aperture size: 3

The image for those parameters is:

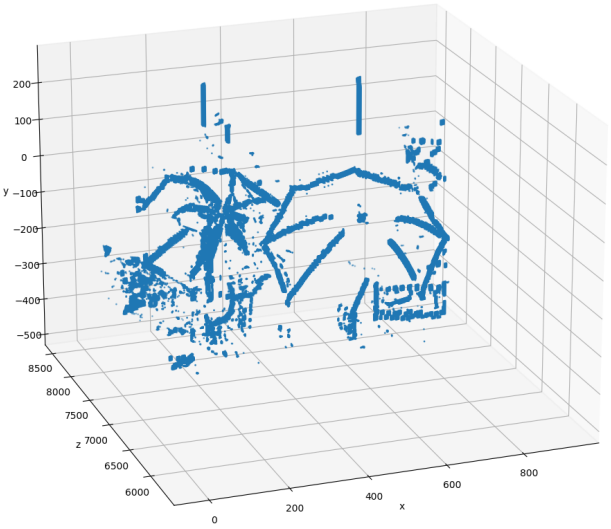


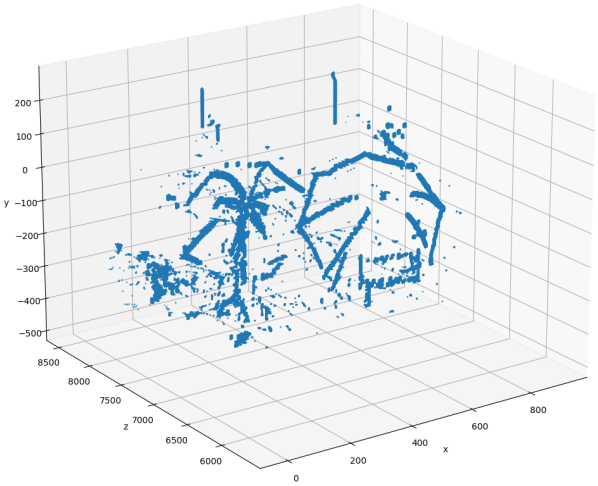
1.3. Views of the Scene

Using the depth calculation function, I have created a 3D plot of the scene in the image. I have also handled the coordinates of the pixels in 3D, and flipped the axes Z and Y. so that the plot if displayed from the perspective of the camera. Thus, the front view would look just like the camera's point of view:

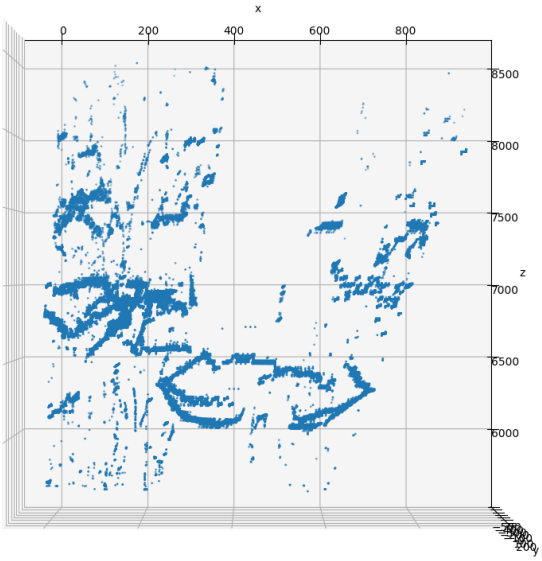


Next are the 3D views:

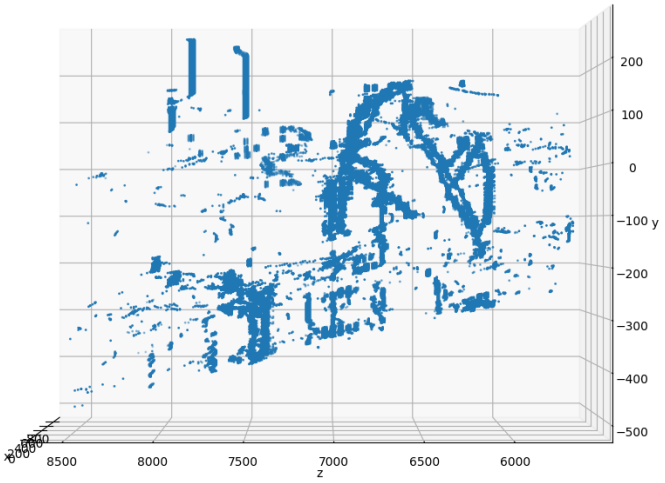




The Top view:



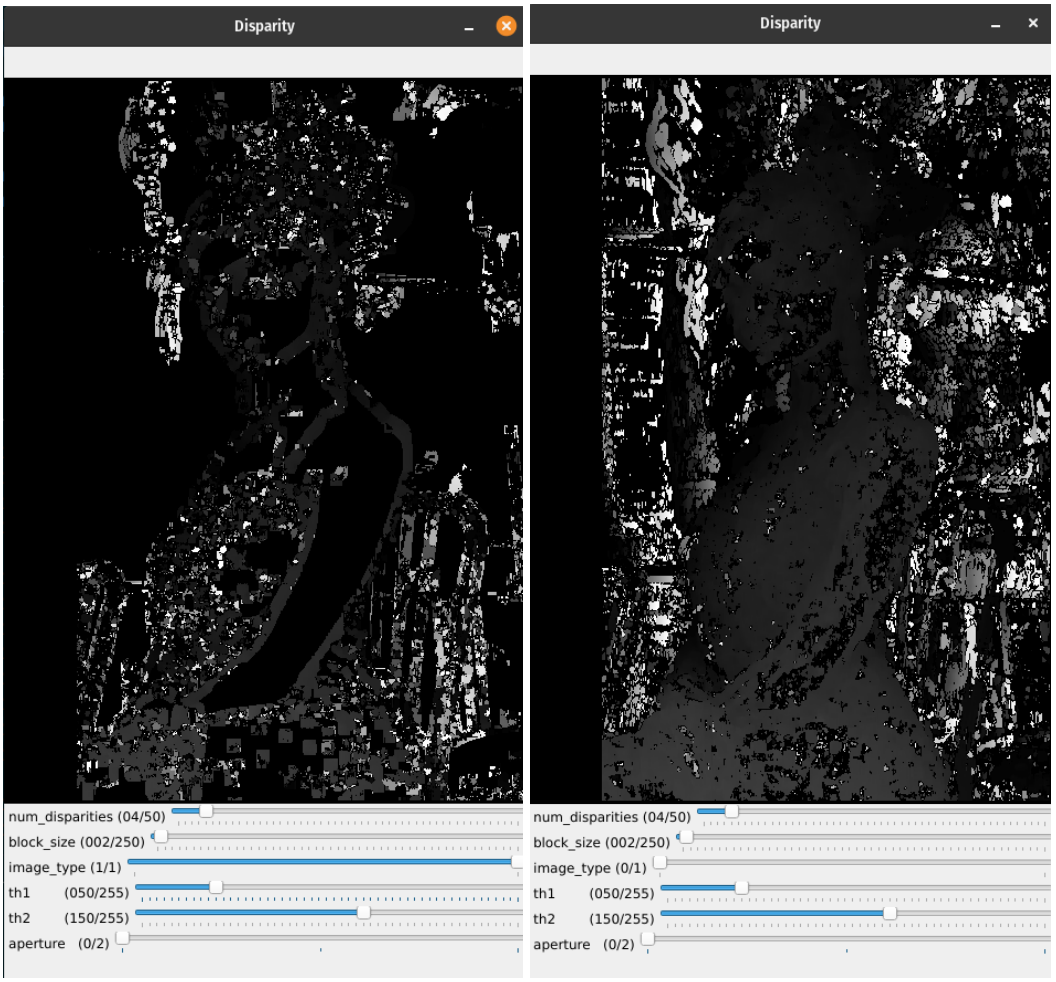
The Side view:



I have removed the pane of pixels in the back of the plot, caused from the values over a certain depth in the scene, as to get a better view of the image. The threshold used by me removed the pixels over 8500.

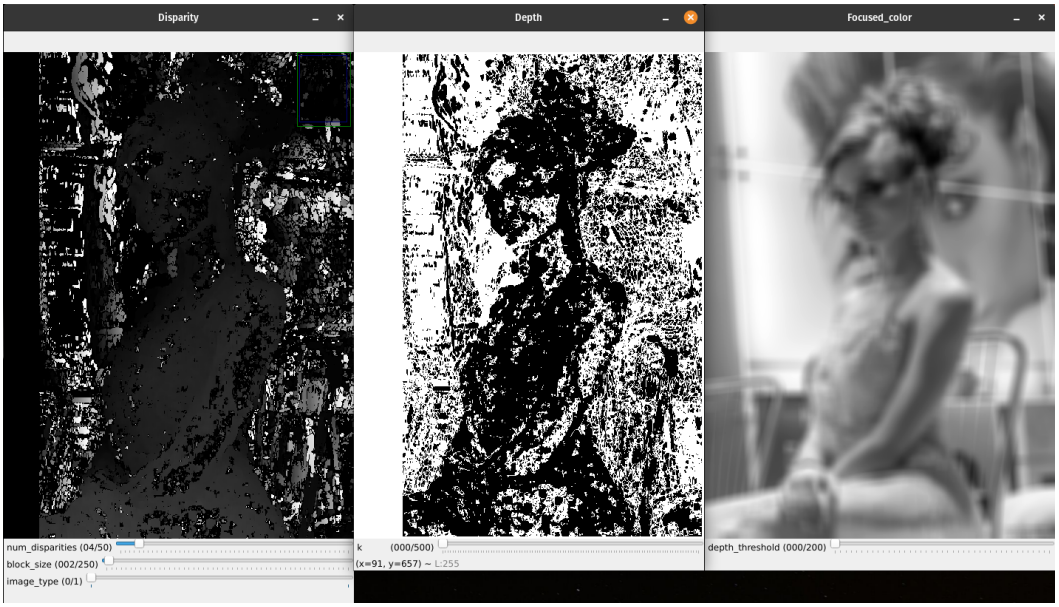
2. Selective Focus

For the selective focus, I have noticed that the greyscale images are better than edge only images, in differentiating the background and foreground of the girl in the given images.

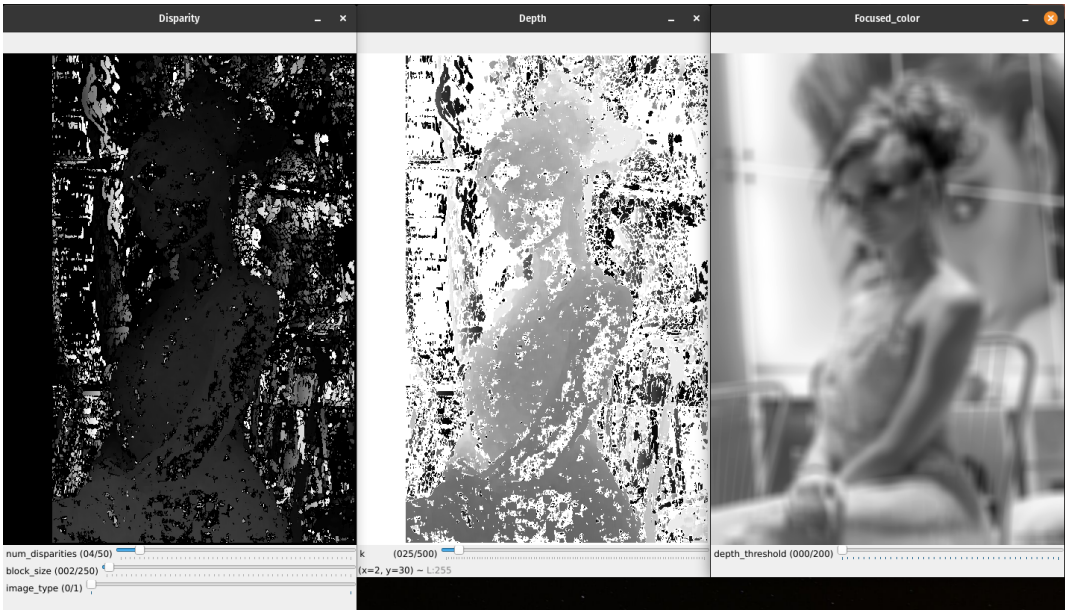


Edge only image – Greyscale image

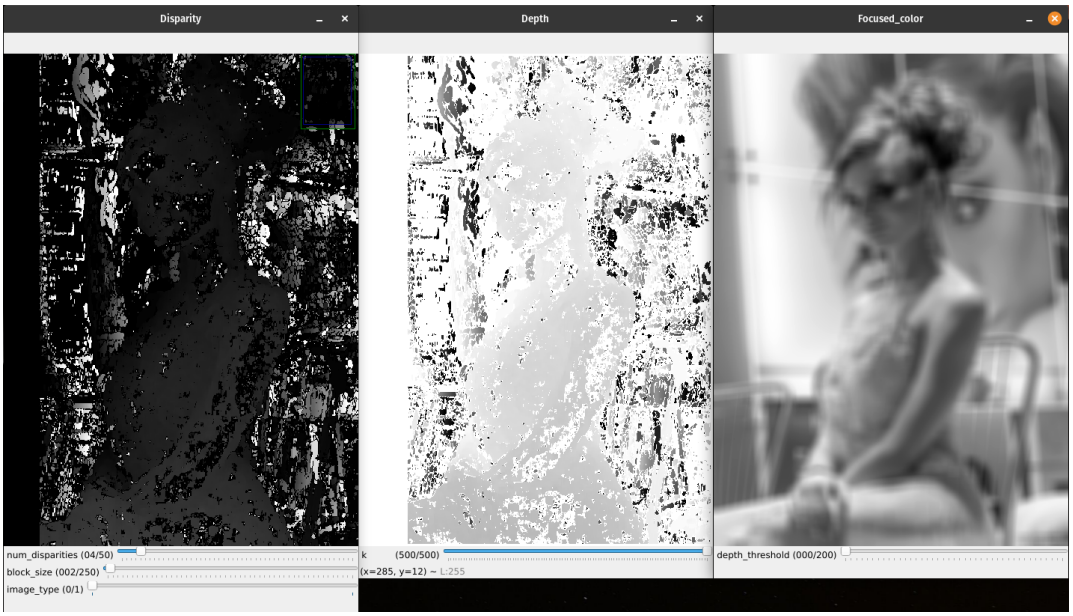
I approximated the depth using the given function and the disparity map, and then implemented a Trackbar to vary the k parameter in the function.



Depth image for k = 0



Depth image for k = 0.25

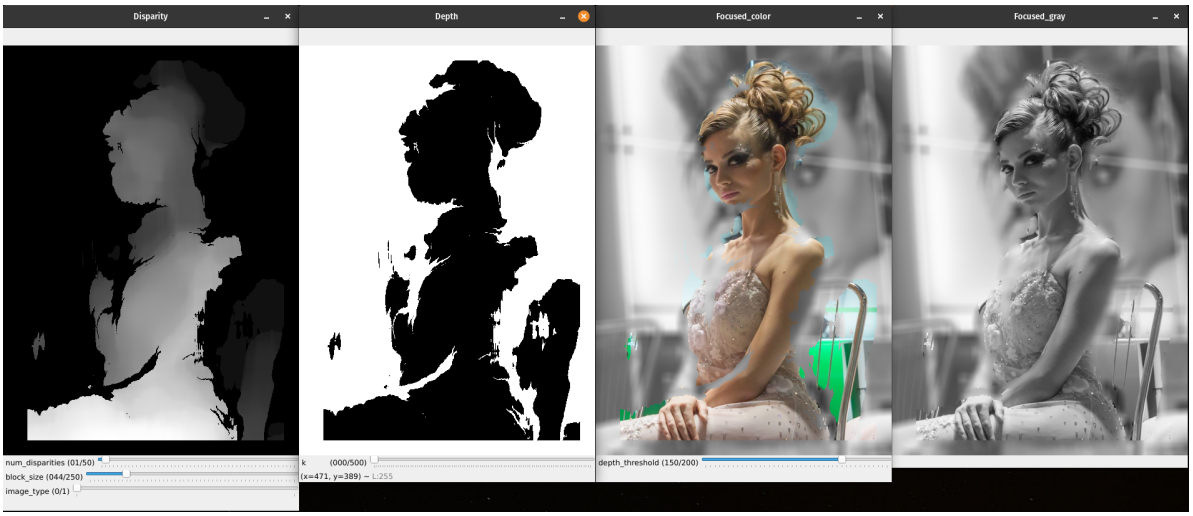


Depth image for k = 5

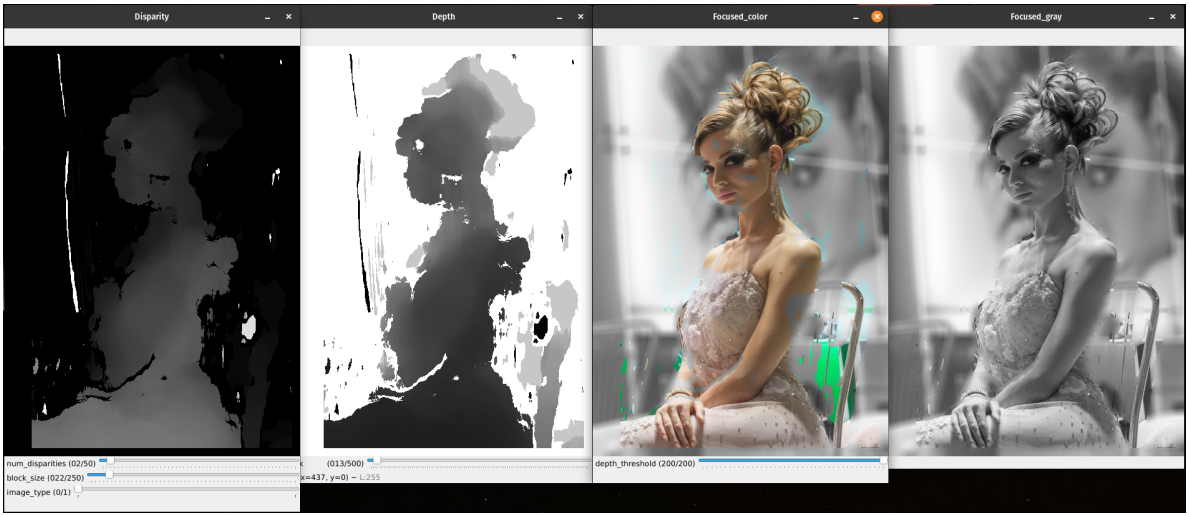
With the depth approximated in the image, I was then able to differentiate the background pixels in the depth image from the object pixels in the foreground.

I decided on implementing both a greyscale image with the background heavily blurred and a colour image with the background greyscale and heavily blurred.

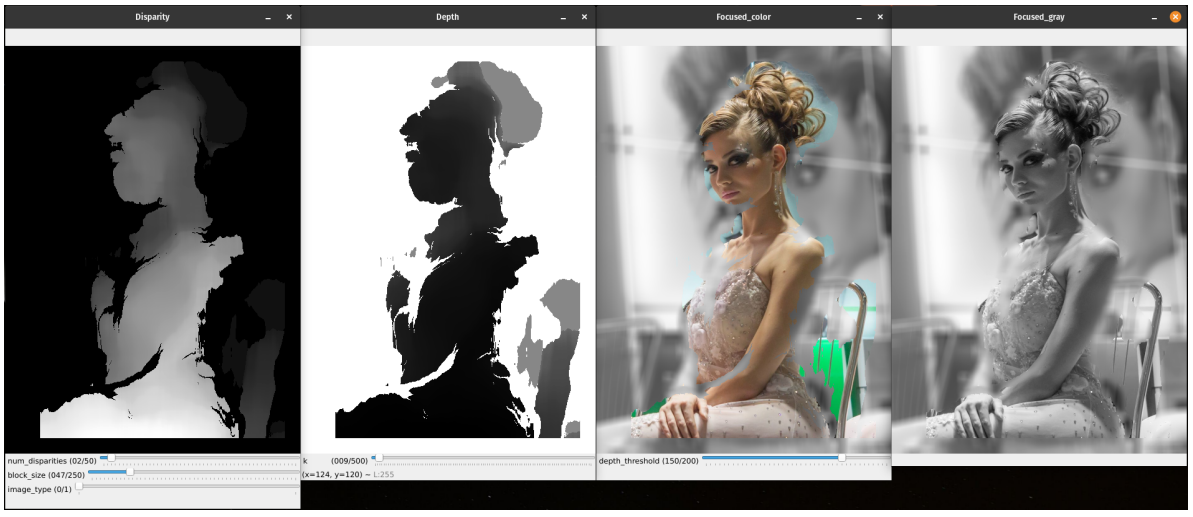
I did the whole process in parallel with all the images, and varied the parameters to get different results. Below are some of the best results received with those variations:



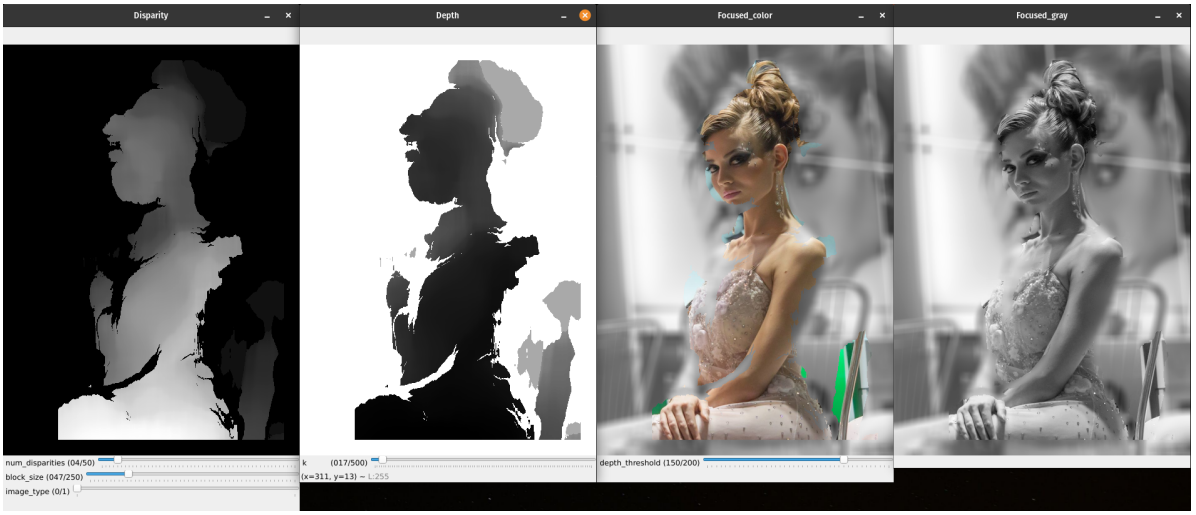
Num. disparities = 16; Block size = 49; k = 0.0



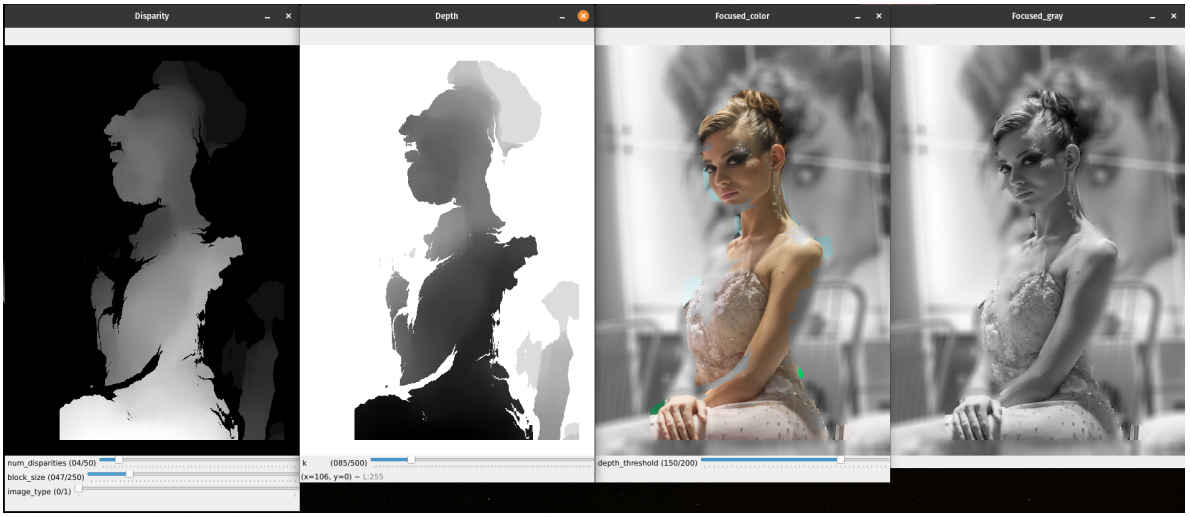
Num. disparities = 32; Block size = 27; k = 0.13



Num. disparities = 32; Block size = 51; k = 0.09



Num. disparities = 64; Block size = 51; k = 0.17



Num. disparities = 64; Block size = 51; k = 0.85

3. Conclusion

1. Stereo Imagery

I have noticed that the number of disparities affect what results in the disparity map, and what connections are made from the two stereo images. It determines the range in which matching pixels are identified from those two images. A good number of disparities, in my use case, varies between 32, 48 and 64.

The disparity map looked much better for the edge only image, when the block size was smaller, but a higher size usually removed some noise and connected discontinued edges.

Overall, the edge only image got a better disparity map and 3D plot, so, the edge detector, Canny in my case, still can to be tuned to get the more optimal result.

In the 3D plot, the main disparities and edges are distinguishable, but there is still some noise in the background.

2. Selective Focus

The number of disparities did not make a huge difference, and the best results were still achieved with 32, 48 and 64.

The block size helps in filling the image and connecting the bulk of the person in the main image, while getting rid of small background noise. This helped a lot in creating a connected block for the main focus object and get rid of unnecessary disparities around the person.

In the resulting focused images, I managed to detect the person in the supplied stereo pair and blur the surrounding background, albeit small parts of the object/background were incorrectly classified. With this you can blur the background and greyscale it to varying levels, based on your parameters, with an already taken image.

All in all, I managed to plot a 3D representation of two already taken stereo images, and managed to distinguish the foreground and background, with greyscale and blur, of another two stereo images fairly well. There is still noise and some miss classification in the results, which can be handled with additional pre-processing of the images.

4. Appendix – Code

1. Stereo Imagery

```
import numpy as np
import cv2
import sys
from matplotlib import pyplot as plt

# Calibration data
cam0=np.array([[5806.559, 0, 1429.219], [0, 5806.559, 993.403], [0, 0, 1]])
cam1=np.array([[5806.559, 0, 1543.51 ], [0, 5806.559, 993.403], [0, 0, 1]])
doffs=114.291
baseline=174.019
width=2960
height=2016
focal_length = 5806.559

# Camera Focal Length Parameters
width_sensor_size = 22.2
height_sensor_size = 14.8
width_camera_resolution = 3088
height_camera_resolution = 2056

# Parameters
global disparity, disparityImg
disparity = None
disparityImg = None
global img, imgL, imgR
imgL = []
imgR = []
global num_disparities, block_size, img_type, th1, th2, aperture
num_disparities = 4
block_size = 3
img_type = 1
th1 = 80
th2 = 165
aperture = 0
global num_disparities_val, block_size_val
num_disparities_val = 0
```

```
block_size_val = 5
# ================================================

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, baseline, doffs, focal):
    # This just plots some sample points.  Change this function to
    # plot the 3D reconstruction from the disparity map and other values
    z = baseline * (focal / (disparity + doffs))
    x = np.zeros(z.shape)
    y = np.zeros(z.shape)
    for i in range(z.shape[0]):
        for j in range(z.shape[1]):
            x[i,j] = z[i,j] * (j / focal) - baseline / 2
            y[i,j] = - z[i,j] * (i / focal) + disparity.shape[0] / 2

    x = x.flatten()
    y = y.flatten()
    z = z.flatten()

    threshold_id = []
    for id, val in enumerate(z):
        if val > 8500:
            threshold_id.append(id)

    x = np.delete(x, threshold_id)
    y = np.delete(y, threshold_id)
    z = np.delete(z, threshold_id)

    # Plt depths
    ax = plt.axes(projection ='3d')
    ax.scatter(x, z, y, s=1)
    ax.view_init(elev=10, azim=-120)

    # Labels
    ax.set_xlabel('x')
    ax.set_ylabel('z')
    ax.set_zlabel('y')
```

```
    # plt.savefig('myplot.pdf', bbox_inches='tight') # Can also specify an
image, e.g. myplot.png
    plt.show()

# ================================================
# Trackbar functions

def print_stats_vals():
    print('-----------------------------------------------------------')
    print("imgtype: {}".format(img_type))
    print("th1: {}".format(th1))
    print("th2: {}".format(th2))
    print("aperture: {}".format(3 + aperture * 2))
    print("num_disparities: {}".format(num_disparities_val))
    print("block_size: {}".format(block_size_val))
    print('-----------------------------------------------------------')

def th1_trackbar(val):
    global th1
    global img
    th1 = val
    imgL[1] = edge_detection(imgL[0], th1, th2, 3 + aperture * 2)
    imgR[1] = edge_detection(imgR[0], th1, th2, 3 + aperture * 2)
    img = [imgL[1], imgR[1]]
    show_disparity()

def th2_trackbar(val):
    global th2
    global img
    th2 = val
    imgL[1] = edge_detection(imgL[0], th1, th2, 3 + aperture * 2)
    imgR[1] = edge_detection(imgR[0], th1, th2, 3 + aperture * 2)
    img = [imgL[1], imgR[1]]
    show_disparity()

def aperture_trackbar(val):
    global aperture
    global img
    aperture = val
    imgL[1] = edge_detection(imgL[0], th1, th2, 3 + aperture * 2)
    imgR[1] = edge_detection(imgR[0], th1, th2, 3 + aperture * 2)
    img = [imgL[1], imgR[1]]
    show_disparity()

def num_disparities_trackbar(val):
    global num_disparities_val
    num_disparities_val = val * 16
    show_disparity()
```

```
def block_size_trackbar(val):
    global block_size_val
    if val % 2 == 0:
        block_size_val = val + 5
    else:
        block_size_val = val + 4
    show_disparity()

def image_type_trackbar(val):
    global img_type
    global img
    img_type = val
    img = [imgL[val], imgR[val]]
    show_disparity()

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

def focal_length_px_to_mm(focal_length_px, sensor_size_mm, image_resolution):
    return focal_length_px * sensor_size_mm / image_resolution

def edge_detection(gray, th1, th2, aperture):
    # Apply Canny edge detection
    edges = cv2.Canny(gray, th1, th2, apertureSize=aperture)
    return edges

def show_disparity():
    global disparity
    global disparityImg
    disparity = getDisparityMap(img[0], img[1], num_disparities_val,
block_size_val)
    disparityImg = np.interp(disparity, (disparity.min(), disparity.max()),
(0.0, 1.0))
    print_stats_vals()
    cv2.imshow('Disparity', disparityImg)

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

if __name__ == '__main__':

    # Load left image
    filename = 'umbrellaL.png'
    imgL.append(cv2.imread(filename, cv2.IMREAD_GRAYSCALE))
    if imgL[0] is None:
        print('\nError: failed to open {}.\n'.format(filename))
        sys.exit()

    # Load right image
    filename = 'umbrellaR.png'
    imgR.append(cv2.imread(filename, cv2.IMREAD_GRAYSCALE))
    if imgR[0] is None:
```

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

    imgL.append(edge_detection(imgL[0], th1, th2, 3 + aperture * 2))
    imgR.append(edge_detection(imgR[0], th1, th2, 3 + aperture * 2))


    cam0_x = focal_length_px_to_mm(cam0[0,0], width_sensor_size,
width_camera_resolution)
    cam0_y = focal_length_px_to_mm(cam0[1,1], height_sensor_size,
height_camera_resolution)

    cam1_x = focal_length_px_to_mm(cam1[0,0], width_sensor_size,
width_camera_resolution)
    cam1_y = focal_length_px_to_mm(cam1[1,1], height_sensor_size,
height_camera_resolution)

    print("cam0 x focal length in mm: ", cam0_x)
    print("cam0 y focal length in mm: ", cam0_y)
    print("cam1 x focal length in mm: ", cam1_x)
    print("cam1 y focal length in mm: ", cam1_y)


################################################################################
####

    # Create a window to display the image in
    cv2.namedWindow('Disparity', cv2.WINDOW_AUTOSIZE)

    cv2.createTrackbar('num_disparities', 'Disparity', num_disparities, 50,
num_disparities_trackbar)
    cv2.createTrackbar('block_size', 'Disparity', block_size, 250,
block_size_trackbar)
    cv2.createTrackbar('image_type', 'Disparity', img_type, 1,
image_type_trackbar)
    cv2.createTrackbar('th1', 'Disparity', th1, 255, th1_trackbar)
    cv2.createTrackbar('th2', 'Disparity', th2, 255, th2_trackbar)
    cv2.createTrackbar('aperture', 'Disparity', aperture, 2,
aperture_trackbar)

    cv2.waitKey(0)

    # Plot the 3D reconstruction
    # disparity = getDisparityMap(img[0], img[1], num_disparities_val,
block_size_val)

    cv2.destroyAllWindows()

    plot(disparity, baseline, doffs, focal_length)
```

2. Selective Focus

```
import numpy as np
import cv2
import sys
from matplotlib import pyplot as plt

# Calibration data
cam0=np.array([[5806.559, 0, 1429.219], [0, 5806.559, 993.403], [0, 0, 1]])
cam1=np.array([[5806.559, 0, 1543.51 ], [0, 5806.559, 993.403], [0, 0, 1]])
doffs=114.291
baseline=174.019
width=2960
height=2016
focal_length = 5806.559

# Camera Focal Length Parameters
width_sensor_size = 22.2
height_sensor_size = 14.8
width_camera_resolution = 3088
height_camera_resolution = 2056

# Parameters
global img, imgL, imgR
imgL = []
imgR = []
global num_disparities, block_size, img_type, th1, th2, aperture
num_disparities = 0
block_size = 0
img_type = 1
th1 = 50
th2 = 150
aperture = 0
global num_disparities_val, block_size_val
num_disparities_val = 0
block_size_val = 5
global disparity, depth
disparity = None
depth = None
global k, depth_threshold
k = 0
depth_threshold = 0
# ================================================

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, baseline, doffs, focal):
    # This just plots some sample points.  Change this function to
    # plot the 3D reconstruction from the disparity map and other values
    z = baseline * (focal / (disparity + doffs))
    x = np.zeros(z.shape)
    y = np.zeros(z.shape)
    for i in range(z.shape[0]):
        for j in range(z.shape[1]):
            x[i,j] = z[i,j] * (j / focal) - baseline / 2
            y[i,j] = - z[i,j] * (i / focal) + disparity.shape[0] / 2

    x = x.flatten()
    y = y.flatten()
    z = z.flatten()

    threshold_id = []
    for id, val in enumerate(z):
        if val > 8500:
            threshold_id.append(id)

    x = np.delete(x, threshold_id)
    y = np.delete(y, threshold_id)
    z = np.delete(z, threshold_id)

    # Plt depths
    ax = plt.axes(projection ='3d')
    ax.scatter(x, z, y, s=1)
    ax.view_init(elev=0, azim=-90)

    # Labels
    ax.set_xlabel('x')
    ax.set_ylabel('z')
    ax.set_zlabel('y')

    # plt.savefig('myplot.pdf', bbox_inches='tight') # Can also specify an
image, e.g. myplot.png
    plt.show()

# ================================================
# Trackbar functions

def print_stats_vals():
    print('-----------------------------------------------------')
```

```
    print("imgtype: {}".format(img_type))
    print("th1: {}".format(th1))
    print("th2: {}".format(th2))
    print("aperture: {}".format(3 + aperture * 2))
    print("num_disparities: {}".format(num_disparities_val))
    print("block_size: {}".format(block_size_val))
    print("k: {}".format(k))
    print("depth_threshold: {}".format(depth_threshold))
    print('-----------------------------------------------------')

def image_type_trackbar(val):
    global img_type
    global img
    img_type = val
    img = [imgL[val], imgR[val]]
    show_disparity()

def th1_trackbar(val):
    global th1
    global img
    th1 = val
    imgL[1] = edge_detection(imgL[0], th1, th2, 3 + aperture * 2)
    imgR[1] = edge_detection(imgR[0], th1, th2, 3 + aperture * 2)
    img = [imgL[1], imgR[1]]
    show_disparity()

def th2_trackbar(val):
    global th2
    global img
    th2 = val
    imgL[1] = edge_detection(imgL[0], th1, th2, 3 + aperture * 2)
    imgR[1] = edge_detection(imgR[0], th1, th2, 3 + aperture * 2)
    img = [imgL[1], imgR[1]]
    show_disparity()

def aperture_trackbar(val):
    global aperture
    global img
    aperture = val
    imgL[1] = edge_detection(imgL[0], th1, th2, 3 + aperture * 2)
    imgR[1] = edge_detection(imgR[0], th1, th2, 3 + aperture * 2)
    img = [imgL[1], imgR[1]]
    show_disparity()

def num_disparities_trackbar(val):
    global num_disparities_val
    num_disparities_val = val * 16
    show_disparity()

def block_size_trackbar(val):
```

```
    global block_size_val
    if val % 2 == 0:
        block_size_val = val + 5
    else:
        block_size_val = val + 4
    show_disparity()

def k_trackbar(val):
    global k
    k = val / 100
    show_disparity()

def depth_trackbar(val):
    global depth_threshold
    depth_threshold = val
    show_disparity()

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

def focal_length_px_to_mm(focal_length_px, sensor_size_mm, image_resolution):
    return focal_length_px * sensor_size_mm / image_resolution

def edge_detection(gray, th1, th2, aperture):
    # Apply Canny edge detection
    edges = cv2.Canny(gray, th1, th2, apertureSize=aperture)
    return edges

def show_disparity():
    global disparity
    global depth
    disparity = getDisparityMap(img[0], img[1], num_disparities_val,
block_size_val)
    disparity = np.interp(disparity, (disparity.min(), disparity.max()), (0.0,
1.0))

    depth = 1 / (disparity + k)
    depth = np.interp(depth, (depth.min(), depth.max()), (0, 255))

    mask = (depth < depth_threshold)[:, :, np.newaxis]
    mask_gray = (depth < depth_threshold)
    foreground = img_left
    foreground_gray = cv2.cvtColor(foreground, cv2.COLOR_BGR2GRAY)
    background = cv2.cvtColor(foreground, cv2.COLOR_BGR2GRAY)
    background = cv2.cvtColor(background, cv2.COLOR_GRAY2BGR)

    background_color = cv2.GaussianBlur(background, (15, 15), 256)
    background_gray  = cv2.GaussianBlur(cv2.cvtColor(foreground,
cv2.COLOR_BGR2GRAY), (15, 15), 256)
```

```
    focused_color = ((mask * foreground) + ((1 - mask) *
background_color)).astype(np.uint8)
    focused_color = np.interp(focused_color, (focused_color.min(),
focused_color.max()), (0.0, 1.0))

    focused_gray  = ((mask_gray * foreground_gray) + ((1 - mask_gray) *
background_gray)).astype(np.uint8)
    focused_gray  = np.interp(focused_gray, (focused_gray.min(),
focused_gray.max()), (0.0, 1.0))

    print_stats_vals()
    cv2.imshow('Disparity', disparity)
    cv2.imshow('Depth', depth.astype(np.uint8))
    cv2.imshow('Focused_color', focused_color)
    cv2.imshow('Focused_gray', focused_gray)

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

if __name__ == '__main__':

    # Load left image
    filename = 'girlL.png'
    imgL.append(cv2.imread(filename, cv2.IMREAD_GRAYSCALE))
    img_left = cv2.imread(filename, cv2.IMREAD_COLOR)
    if imgL[0] is None:
        print('\nError: failed to open {}.\n'.format(filename))
        sys.exit()

    # Load right image
    filename = 'girlR.png'
    imgR.append(cv2.imread(filename, cv2.IMREAD_GRAYSCALE))
    img_right = cv2.imread(filename, cv2.IMREAD_COLOR)
    if imgR[0] is None:
        print('\nError: failed to open {}.\n'.format(filename))
        sys.exit()

    imgL.append(edge_detection(imgL[0], th1, th2, 3 + aperture * 2))
    imgR.append(edge_detection(imgR[0], th1, th2, 3 + aperture * 2))

    cam0_x = focal_length_px_to_mm(cam0[0,0], width_sensor_size,
width_camera_resolution)
    cam0_y = focal_length_px_to_mm(cam0[1,1], height_sensor_size,
height_camera_resolution)

    cam1_x = focal_length_px_to_mm(cam1[0,0], width_sensor_size,
width_camera_resolution)
    cam1_y = focal_length_px_to_mm(cam1[1,1], height_sensor_size,
height_camera_resolution)

    print("cam0 x focal length in mm: ", cam0_x)
```

```
    print("cam0 y focal length in mm: ", cam0_y)
    print("cam1 x focal length in mm: ", cam1_x)
    print("cam1 y focal length in mm: ", cam1_y)


########################################################################
####

    # Create a window to display the image in
    cv2.namedWindow('Disparity', cv2.WINDOW_AUTOSIZE)
    cv2.namedWindow('Depth', cv2.WINDOW_AUTOSIZE)
    cv2.namedWindow('Focused_color', cv2.WINDOW_AUTOSIZE)
    cv2.namedWindow('Focused_gray', cv2.WINDOW_AUTOSIZE)

    cv2.createTrackbar('num_disparities', 'Disparity', num_disparities, 50,
num_disparities_trackbar)
    cv2.createTrackbar('block_size', 'Disparity', block_size, 250,
block_size_trackbar)
    cv2.createTrackbar('image_type', 'Disparity', img_type, 1,
image_type_trackbar)
    # cv2.createTrackbar('th1', 'Disparity', th1, 255, th1_trackbar)
    # cv2.createTrackbar('th2', 'Disparity', th2, 255, th2_trackbar)
    # cv2.createTrackbar('aperture', 'Disparity', aperture, 2,
aperture_trackbar)
    cv2.createTrackbar('k', 'Depth', 0, 500, k_trackbar)
    cv2.createTrackbar('depth_threshold', 'Focused_color', depth_threshold,
200, depth_trackbar)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # plot(disparity, baseline, doffs, focal_length)
```