COMP37212: Computer Vision
Coursework 3

Boning Cui

April 29, 2024

Contents

1 Stereo Imagery

1.1 Focal Length Calculation

Given:

- the focal length (in pixels) of the two cameras in the calibration table, where:

$$f_{pixels} = f_{pixels}^{cam0} = f_{pixels}^{cam1} = 5806.559;$$

- the physical sensor size (in millimetres) of the cameras: $22.2\ mm \times 14.8\ mm$;
- the image size (in pixels) yielded by the cameras: 3088×2056;

The focal length in millimetres can thus be computed as:

$$\begin{aligned} f_{mm} &= f_{pixels} \times \frac{sensor\ width\ (in\ mm)}{image\ width\ (in\ pixels)} \\ &= 5806.599 \times \frac{22.2}{3088} \\ &= 41.74\ (mm) \end{aligned}$$

1.2 Disparity Map

The disparity calculation function in the given script takes two parameters: number of disparities (*numDisparities*) and block size (*blockSize*). According to the requirements in OpenCV(4.6.0):

- *numDisparities* must be positive and divisible by 16;
- *blockSize* must be odd within $[5, 255]$ and not larger than the image width or height.

Thus, to select the proper parameters values so that the resulting disparity image has fewer noise, the two parameters are decided via trackbars, from range:

- *numDisparities*: $[16, 512]$, increased by 16 per step;
- *blockSize*: $[5, 125]$, increased by 2 per step;

which can be achieved by adding the following codes into *disparity.py*:

```
# Initialise numDisparities:
# ranging from [16, 512], increased by 16 per step
numDisparities = 16
```

```
    # Initialise blockSize:
    # ranging from [5, 125], increase by 2 per step
    blockSize = 5

    def onChangeNumDisparities(val):
        global numDisparities
        numDisparities = 16 + 16 * val

        updateDisparity()

    def onChangeBlockSize(val):
        global blockSize
        blockSize = 5 + 2 * val

        updateDisparity()

    def updateDisparity():
        disparity = getDisparityMap(imgL, imgR,
           numDisparities, blockSize)
        disparityImg = np.interp(disparity, (disparity.min()
           , disparity.max()), (0.0, 1.0))

        cv2.imshow('Disparity', disparityImg)

    ... ...

    # Create a window and trackbars for the two parameters
    cv2.namedWindow('Disparity', cv2.WINDOW_NORMAL)
    cv2.createTrackbar('numDisparities␣Step', 'Disparity',
       0, 31, onChangeNumDisparities)
    cv2.createTrackbar('blockSize␣Step', 'Disparity', 0, 60,
        onChangeBlockSize)

    ... ...
```

Listing 1: Add trackbars for *numDisparities* and *blockSize*.

Next, two types of images pairs are used to calculate the disparity map: the **greyscale** images pair and the **edge-detected** images pair of the original left and right pictures.

1.2.1 Using Greyscale Images

Given the left and right greyscale images shown in Figure 1 (a) and (c), the (roughly) best disparity map trying to reduce noise while keeping the general shape of the umbrellas is obtained by testing different values of *numDisparities*

and *blockSize* via trackbars, and is displayed in Figure 1 (b), where $numDisparities = 64$ and $blockSize = 15$.

Additionally, (d) - (l) in Figure 1 are some of the test results of adjusting the two parameters via trackbars.

1.2.2 Using Edge-Detected Images

The greyscale images are first blurred by a 7×7 Gaussian kernel, and then processed using an edge detection operation from *cv2.Canny()*, in which two parameters *low_threshold* and *high_threshold* decide which edges to be kept (here, $low\_threshold = 7$ and $high\_threshold = 54$, selected using trackbars implemented in A.1). Finally, they are converted to binary edge-detected images, as shown in Figure 2 (a) and (c).

Given the pair of edge-detected images, same procedural has been applied as for greyscale images and the resulting disparity map is shown in Figure 2 (b), where $numDisparities = 64$ and $blockSize = 7$. (d) - (l) show some resulting images during parameters' tuning.

1.2.3 Comparison

The disparity map yielded by the greyscale images contains more noise, which can be eased to some extent by increasing *blockSize*, despite the bones of the umbrellas being fatter, distorted or missed.

On the other hand, the edge-detected images are first blurred and then keep only the relatively strong edges representing the umbrellas, thus produce a better disparity map which balances background noise reduction and objects structure preservation.

1.3 Views of the Scene

This section uses the disparity map computed using the edge-detected images pair, as shown in Figure 2 (b).

1.3.1 Calculate the Depth into the Scene (Z)

Given:

- $baseline = 174.019\ mm$, with respect to the camera used;
- $f_{pixels} = 5806.559$;
- $doffs = 114.291$;
- a disparity map d;

The depth (Z) of any pixel i in the image can thus be calculated using:

$$Z_i = baseline \times \frac{f_{pixels}}{d_i + doffs} \ \ (mm)$$

1.3.2 Calculate X and Y

Now the Z value for every pixel is obtained, the property of similar triangles can then be utilised to calculate the world coordinates X and Y for arbitrary pixel i:

$$\frac{x_i}{X_i} = \frac{y_i}{Y_i} = \frac{f_{pixels}}{Z_i};$$

where x_i and y_i are scene coordinates and can be represented by horizontal pixel coordinate u_i, vertical pixel coordinate v_i and the principal point of the camera:

$$x_i = u_i - c_x \ ; \ y_i = v_i - c_y$$

The c_y is the same for both cameras, while the c_x uses that of the left camera since $doffs = c_x^1 - c_x^0$ and thus the left camera is the reference frame when generating the disparity map. Now the whole calculation can be carried:

$$X_i = \frac{(u_i - c_x^0) \times Z_i}{f_{pixels}} \ \ (mm)$$
$$Y_i = \frac{(v_i - c_y) \times Z_i}{f_{pixels}} \ \ (mm)$$

1.3.3 Display the Views

Now generate 3D, top and side views using the function below (full code given in A.2):

```
        def plot(disparity, f_pixels, baseline, doffs, cx,
           cy):
            x = []
            y = []
            z = []

            height, width = disparity.shape
            min_disparity = np.min(disparity)

            # Calculate pixel by pixel
            for v in range(height):
                for u in range(width):
                    d_i = disparity[v, u]
```

```
                    # Skip if the pixel is from the region
                       that left and right images do not
                       overlap
                    if d_i > min_disparity:
                        Z_i = baseline * f_pixels / (d_i +
                           doffs)

                        # X_i = (u - cx) * Z_i / f_pixels
                        X_i = u * Z_i / f_pixels

                        # Y_i = (v - cy) * Z_i / f_pixels
                        Y_i = v * Z_i / f_pixels

                        x.append(X_i)
                        y.append(Y_i)
                        z.append(Z_i)

            # Plt depths
            ax = plt.axes(projection ='3d')
            ax.scatter(x, z, y, 'green', s=0.2)

            # 3D view
            ax.view_init(elev=25, azim=65)

            # Top view
            # ax.view_init(elev=90, azim=90)

            # Side view
            # ax.view_init(elev=180, azim=0)

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

            plt.savefig('myplot.png', bbox_inches='tight')
            plt.show()
```

The three resulting views are plotted in Figure 3.

2 Selective Focus

2.1 Find Proper Mask for Segmentation

Given an approximation of the depth utilising the disparity map as:

$$depth = \frac{1}{disparity + k} \ ,$$

a depth image that roughly separates the girl from the background can be obtained by tuning these parameters: *numDisparities*, *blockSize* and k. This depth image is then converted into a binary mask (either 0 or 255) according to a threshold value, which is used later for performing different operations on the girl and the background respectively.

The following functions (full codes given in A.3) achieve such tasks by implementing trackbars for tuning:

- *numDisparities* and *blockSize* (controlling the disparity map);
- k (controlling the depth image);
- *threshold* (controlling the binary mask);

and displaying corresponding windows to visualise the changes made:

```
    # Kernel size for blurring the background
    GAUSSIAN_SIZE = 21

    # Ranging from [16, 512], increased by 16 per step
    numDisparities = 16

    # Ranging from [5, 125], increase by 2 per step
    blockSize = 5

    # Ranging from [0, 5], increased by 0.05 per step
    k = 0

    # Ranging from [0, 255], increased by 1 per step
    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 onChangeNumDisparities(val):
        global numDisparities
        numDisparities = 16 + 16 * val

        updateAll()

    def onChangeBlockSize(val):
        global blockSize
        blockSize = 5 + 2 * val

        updateAll()

    def onChangeK(val):
        global k
        k = val / 20

        updateAll()

    def onChangeThreshold(val):
        global threshold
        threshold = val

        updateAll()

    def updateAll():
        # Get disparity map and depth image
        disparity = getDisparityMap(imgL, imgR,
            numDisparities, blockSize)
        depth = 1 / (disparity + k)

        # Normalise for display
        disparityImg = np.interp(disparity, (disparity.min()
            , disparity.max()), (0.0, 1.0))
        depth_normalized = cv2.normalize(depth, None, alpha
            =0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=
            cv2.CV_8U)

        # Apply threshold
```

```
        _, mask = cv2.threshold(depth_normalized, thresh=
            threshold, maxval=255, type=cv2.THRESH_BINARY)

        # Blur the whole left greyscale image first
        blurred_image = cv2.GaussianBlur(imgL, (
            GAUSSIAN_SIZE, GAUSSIAN_SIZE), sigmaX=0, sigmaY
            =0)

        # Then use the mask to protect the object area
        final_image = np.where(mask == 255, blurred_image,
            imgL)
        final_image = final_image.astype('uint8')

        cv2.imshow('Disparity', disparityImg)
        cv2.imshow('Depth', depth_normalized)
        cv2.imshow('Mask', mask)
        cv2.imshow('Final Image', final_image)

    ... ...

    # Create the disparity window
    cv2.namedWindow('Disparity')
    cv2.createTrackbar('numDisparities Step', 'Disparity',
       0, 31, onChangeNumDisparities)
    cv2.createTrackbar('blockSize Step', 'Disparity', 0, 60,
        onChangeBlockSize)

    # Create the depth window
    cv2.namedWindow('Depth')
    cv2.createTrackbar('k * 20', 'Depth', 0, 100, onChangeK)

    # Create the mask window
    cv2.namedWindow('Mask')
    cv2.createTrackbar('Threshold', 'Mask', 0, 255,
       onChangeThreshold)

    # Create window for the final image
    cv2.namedWindow('Final Image')

    # Initialise the four windows
    updateAll()

    ... ...
```

The results are shown in Figure 4, in which:

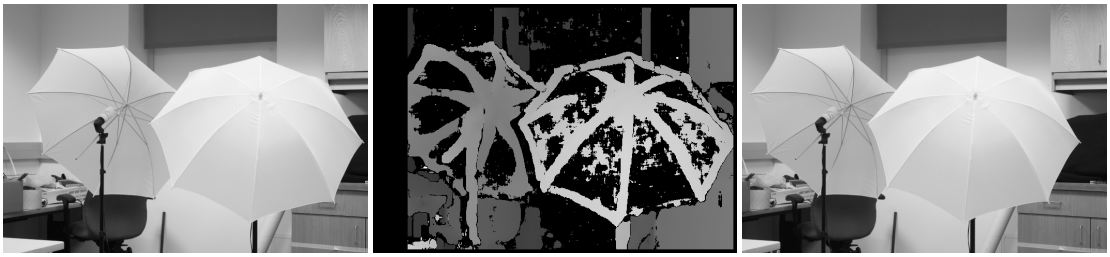
- the left column represents the disparity map with the specified *numDisparities* and *blockSize*;
- the middle column represents the depth image with the specified k and disparity map;
- the right column represents the binary mask obtained by performing thresholding using the specified threshold value.

Additionally, (a) - (c) in Figure 4 are the selected final settings, while (d) - (f) and (g) - (i) are two sets of experimental settings during the tuning.

2.2 Apply Different Operations on Foreground & Background

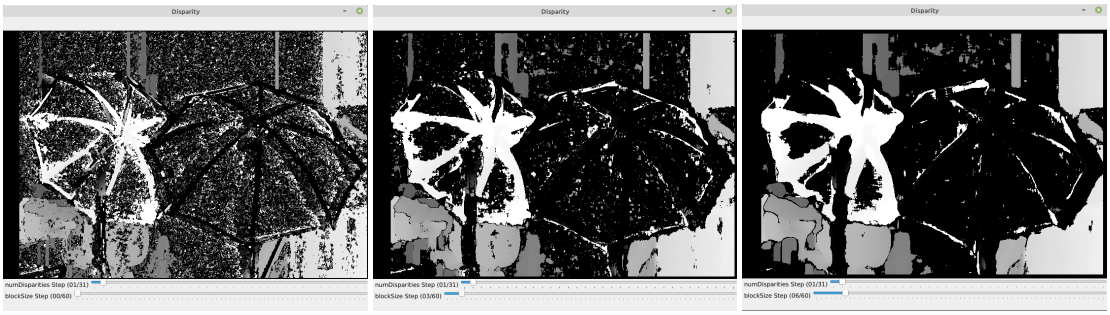
The mask shown in Figure 4 (c) is then used to protect the girl from being blurred, while the other parts of the image are considered as background and thus get blurred by a Gaussian kernel of size (21, 21).

The resulting greyscale image with only the background heavily blurred is displayed in Figure 5.

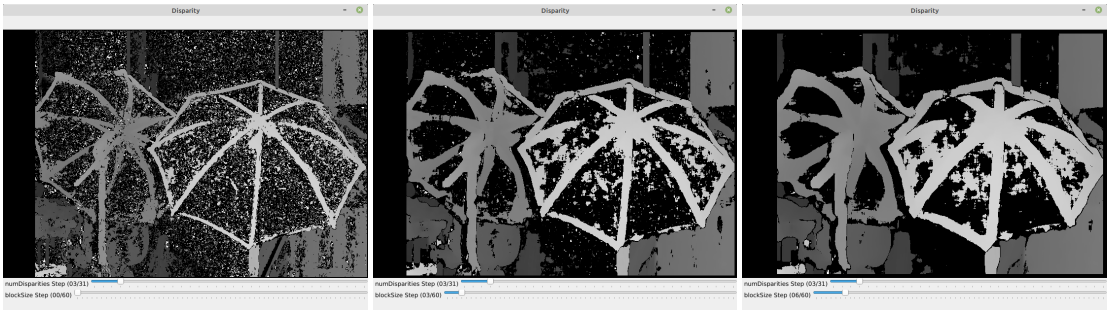


(a) Left Greyscale (b) Selected: (64, 15) (c) Right Greyscale

(d) Test: (16, 5) (e) Test: (16, 11) (f) Test: (16, 17)

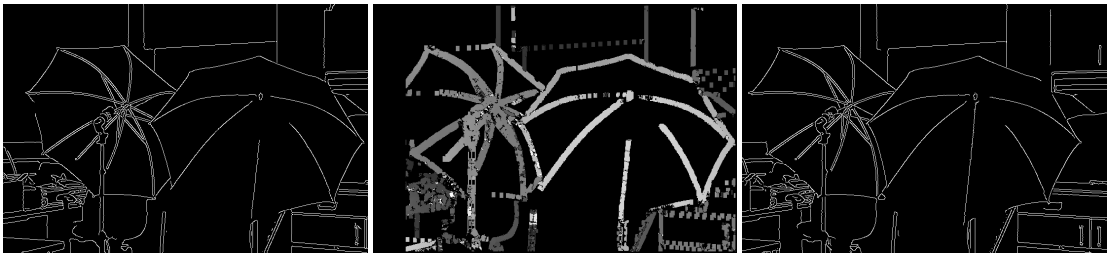


(g) Test: (32, 5) (h) Test: (32, 11) (i) Test: (32, 17)

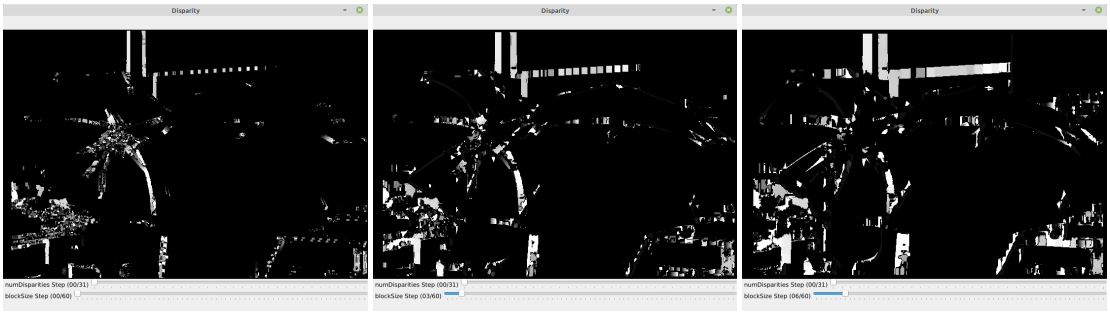


(j) Test: (64, 5) (k) Test: (64, 11) (l) Test: (64, 17)

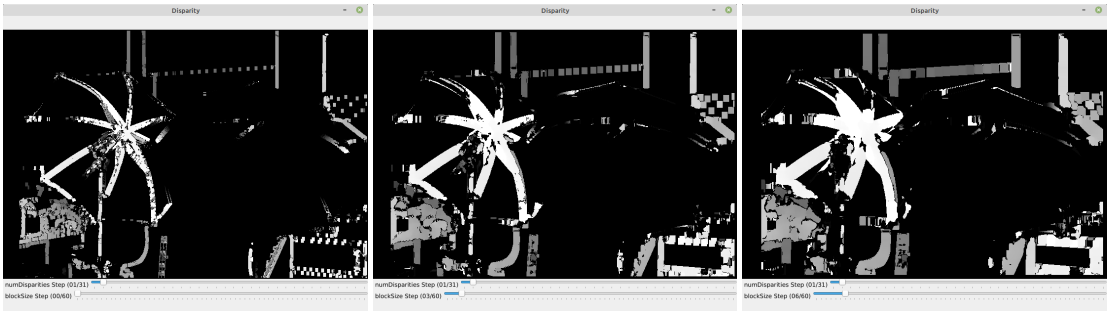
Figure 1: The disparity image calculated using the pair of greyscale images. Tuple (a, b) means $numDisparities = a$ and $blockSize = b$.



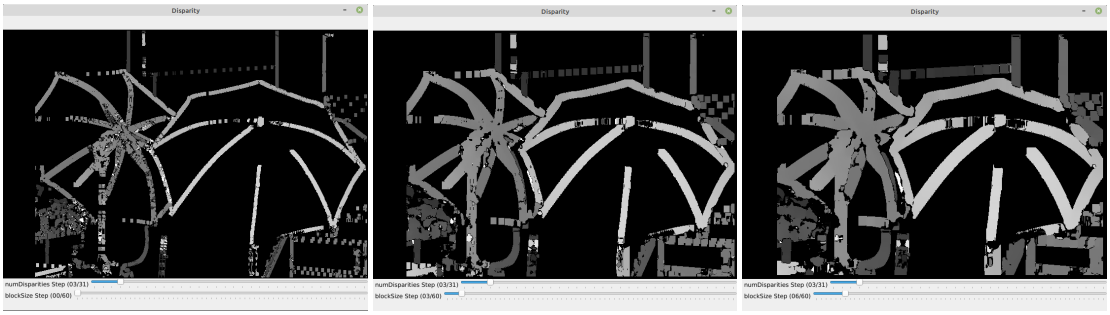
(a) Left Edge-Detected (b) Selected: (64, 7) (c) Right Edge-Detected



(d) Test: (16, 5) (e) Test: (16, 11) (f) Test: (16, 17)

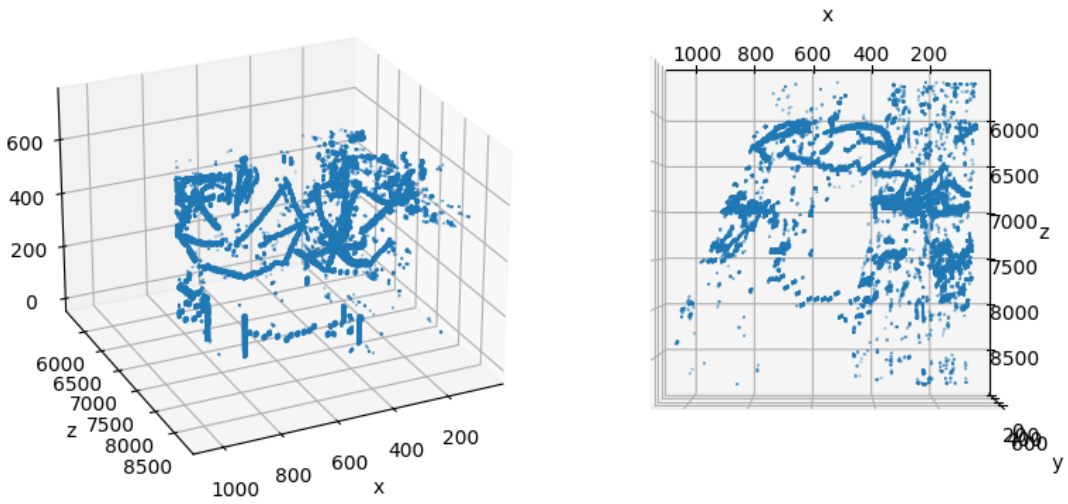


(g) Test: (32, 5) (h) Test: (32, 11) (i) Test: (32, 17)



(j) Test: (64, 5) (k) Test: (64, 11) (l) Test: (64, 17)

Figure 2: The disparity image calculated using the pair of edge-detected images. Tuple (a, b) means $numDisparities = a$ and $blockSize = b$.



(a) 3D View

(b) Top View

(c) Side View

Figure 3: Views of the Scene.

(a) Selected: (32, 51)

(b) $k = 1.00$

(c) $Threshold = 141$

(d) Test 1: (16, 9)

(e) $k = 1.10$

(f) $Threshold = 63$

(g) Test 2: (80, 45)

(h) $k = 0.70$

(i) $Threshold = 93$

Figure 4: Tune various parameters to find a proper mask. Left column shows disparity maps, middle column shows depth images, right column shows masks.

Figure 5: The resulting greyscale with the background being heavily blurred.

A Code Snippets

A.1 Code for 1.2 - Disparity Map

```
import numpy as np
import cv2
import sys
from mpl_toolkits import mplot3d
from matplotlib import pyplot as plt


GAUSSIAN_SIZE = 7

# Initialise numDisparities, ranging from [16, 512],
   increased by 16 per step
numDisparities = 16
# Initialise blockSize, ranging from [5, 125], increase by 2
    per step
blockSize = 5

# Initialize default threshold values in Canny
low_threshold = 10
high_threshold = 50

# ================================================
#
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 onChangeNumDisparities(val):
    global numDisparities

    numDisparities = 16 + 16 * val

    updateDisparity()
```

```
def onChangeBlockSize(val):
    global blockSize

    blockSize = 5 + 2 * val

    updateDisparity()

# Update disparity map and display it
def updateDisparity():
    # Get disparity map
    disparity = getDisparityMap(imgL, imgR, numDisparities,
       blockSize)

    # Normalise for display
    disparityImg = np.interp(disparity, (disparity.min(),
       disparity.max()), (0.0, 1.0))

    cv2.imshow('Disparity', disparityImg)


def getBlurredEdgesImage(image, low_T, high_T, blur_size):
    blurred_image = cv2.GaussianBlur(image, (blur_size,
       blur_size), 0)

    edges = cv2.Canny(blurred_image, low_T, high_T)

    return edges

def onChangeLowThreshold(val):
    global low_threshold

    low_threshold = val

    updateEdges()

def onChangeHighThreshold(val):
    global high_threshold

    high_threshold = val

    updateEdges()

def updateEdges():
```

```
    edges = getBlurredEdgesImage(imgL, low_threshold,
       high_threshold, GAUSSIAN_SIZE)

    cv2.imshow('Edge Image', edges)

# ================================================
#
if __name__ == '__main__':

    # Load left image
    filename = 'umbrella_edgesL_7x7_7-54.png'
    imgL = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    #
    if imgL is None:
        print('\nError: failed to open {}.\n'.format(
           filename))
        sys.exit()

    # cv2.imshow('Left Image', imgL)

    # Load right image
    filename = 'umbrella_edgesR_7x7_7-54.png'
    imgR = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    #
    if imgR is None:
        print('\nError: failed to open {}.\n'.format(
           filename))
        sys.exit()

    # cv2.imshow('Right Image', imgR)


    # # Choose Canny edge parameters:
    # cv2.namedWindow('Edge Image')
    # cv2.createTrackbar('Low Threshold', 'Edge Image',
       low_threshold, 255, onChangeLowThreshold)
    # cv2.createTrackbar('High Threshold', 'Edge Image',
       high_threshold, 255, onChangeHighThreshold)

    # # Initial display
    # updateEdges()

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

```
    # Create a window and trackbars for the two parameters
    cv2.namedWindow('Disparity', cv2.WINDOW_NORMAL)
    cv2.createTrackbar('numDisparities␣Step', 'Disparity',
       0, 31, onChangeNumDisparities)
    cv2.createTrackbar('blockSize␣Step', 'Disparity', 0, 60,
        onChangeBlockSize)

    # Initialise the disparity image with default parameters
    updateDisparity()


    # Wait for spacebar press or escape before closing,
    # otherwise window will close without you seeing it
    while True:
        key = cv2.waitKey(1)
        if key == ord('␣') or key == 27:
            break

    cv2.destroyAllWindows()
```

A.2 Code for 1.3 - Views of the Scene

```
import numpy as np
import cv2
import sys
from mpl_toolkits import mplot3d
from matplotlib import pyplot as plt


NUM_DISPARITIES = 64
BLOCK_SIZE = 7

F_PIXELS = 5806.559
C_X = 1429.219
C_Y = 993.403
DOFFS = 114.291
BASELINE = 174.019


# ================================================
#
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_pixels, baseline, doffs, cx, cy):
    x = []
    y = []
    z = []

    height, width = disparity.shape
    min_disparity = np.min(disparity)

    # Calculate pixel by pixel
```

```
    for v in range(height):
        for u in range(width):
            d_i = disparity[v, u]

            # Skip if the pixel is from the region that left
                and right images do not overlap
            if d_i > min_disparity:
                Z_i = baseline * f_pixels / (d_i + doffs)

                # X_i = (u - cx) * Z_i / f_pixels
                X_i = u * Z_i / f_pixels

                # Y_i = (v - cy) * Z_i / f_pixels
                Y_i = v * Z_i / f_pixels

                x.append(X_i)
                y.append(Y_i)
                z.append(Z_i)

    # Plt depths
    ax = plt.axes(projection ='3d')
    ax.scatter(x, z, y, 'green', s=0.2)

    # 3D view
    ax.view_init(elev=25, azim=65)

    # Top view
    # ax.view_init(elev=90, azim=90)

    # Side view
    # ax.view_init(elev=180, azim=0)

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

    plt.savefig('myplot.png', bbox_inches='tight')
    plt.show()


# Load left image
filename = 'umbrella_edgesL_7x7_7-54.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 = 'umbrella_edgesR_7x7_7-54.png'
imgR = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
#
if imgR is None:
    print('\nError: failed to open {}.\n'.format(filename))
    sys.exit()


# Get disparity map
disparity = getDisparityMap(imgL, imgR, NUM_DISPARITIES,
   BLOCK_SIZE)

# Show 3D plot of the scene
plot(disparity, F_PIXELS, BASELINE, DOFFS, C_X, C_Y)
```

A.3 Code for 2 - Selective Focus

```
import numpy as np
import cv2
import sys
from mpl_toolkits import mplot3d
from matplotlib import pyplot as plt

# Kernel size for blurring the background
GAUSSIAN_SIZE = 21

# Ranging from [16, 512], increased by 16 per step
numDisparities = 16

# Ranging from [5, 125], increase by 2 per step
blockSize = 5

# Ranging from [0, 5], increased by 0.05 per step
k = 0

# Ranging from [0, 255], increased by 1 per step
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 onChangeNumDisparities(val):
    global numDisparities
    numDisparities = 16 + 16 * val

    updateAll()

def onChangeBlockSize(val):
    global blockSize
    blockSize = 5 + 2 * val
```

```
    updateAll()

def onChangeK(val):
    global k
    k = val / 20

    updateAll()

def onChangeThreshold(val):
    global threshold
    threshold = val

    updateAll()

def updateAll():
    # Get disparity map and depth image
    disparity = getDisparityMap(imgL, imgR, numDisparities,
       blockSize)
    depth = 1 / (disparity + k)

    # Normalise for display
    disparityImg = np.interp(disparity, (disparity.min(),
       disparity.max()), (0.0, 1.0))
    depth_normalized = cv2.normalize(depth, None, alpha=0,
       beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)

    # Apply threshold
    _, mask = cv2.threshold(depth_normalized, thresh=
       threshold, maxval=255, type=cv2.THRESH_BINARY)

    # Blur the whole left greyscale image first
    blurred_image = cv2.GaussianBlur(imgL, (GAUSSIAN_SIZE,
       GAUSSIAN_SIZE), sigmaX=0, sigmaY=0)

    # Then use the mask to protect the object area
    final_image = np.where(mask == 255, blurred_image, imgL)
    final_image = final_image.astype('uint8')

    cv2.imshow('Disparity', disparityImg)
    cv2.imshow('Depth', depth_normalized)
    cv2.imshow('Mask', mask)
    cv2.imshow('Final␣Image', final_image)

# ============================================================
```

```
# 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()

# cv2.imshow('Left Image', imgL)

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

# cv2.imshow('Right Image', imgR)
# ============================================================

# Create the disparity window
cv2.namedWindow('Disparity')
cv2.createTrackbar('numDisparities Step', 'Disparity', 0,
   31, onChangeNumDisparities)
cv2.createTrackbar('blockSize Step', 'Disparity', 0, 60,
   onChangeBlockSize)

# Create the depth window
cv2.namedWindow('Depth')
cv2.createTrackbar('k * 20', 'Depth', 0, 100, onChangeK)

# Create the mask window
cv2.namedWindow('Mask')
cv2.createTrackbar('Threshold', 'Mask', 0, 255,
   onChangeThreshold)

# Create window for the final image
cv2.namedWindow('Final Image')

# Initialise the four windows
updateAll()


# Wait for spacebar press or escape before closing,
```

```
# otherwise window will close without you seeing it
while True:
    key = cv2.waitKey(1)
    if key == ord('␣') or key == 27:
        break

cv2.destroyAllWindows()
```