CSC420 — Assignment 4

2. Object Detection and Stereo

For this question I have used the following intrinsic camera parameters (from the /calib folder)
f = 721.5377
px = 609.5593
py = 172.854
T = 532.7119288
* all values are in mm

c) Depth Map

In order to *visualize* the depth maps I applied a gaussian filter to the disparity map. This is to negate the effect of noise. Without the gaussian blurring the noisy points would make visualizing the depth map impossible as a few points would have abnormally high depth.

* code for the depth map is found in Appendix — depth.py

004945 - depth map

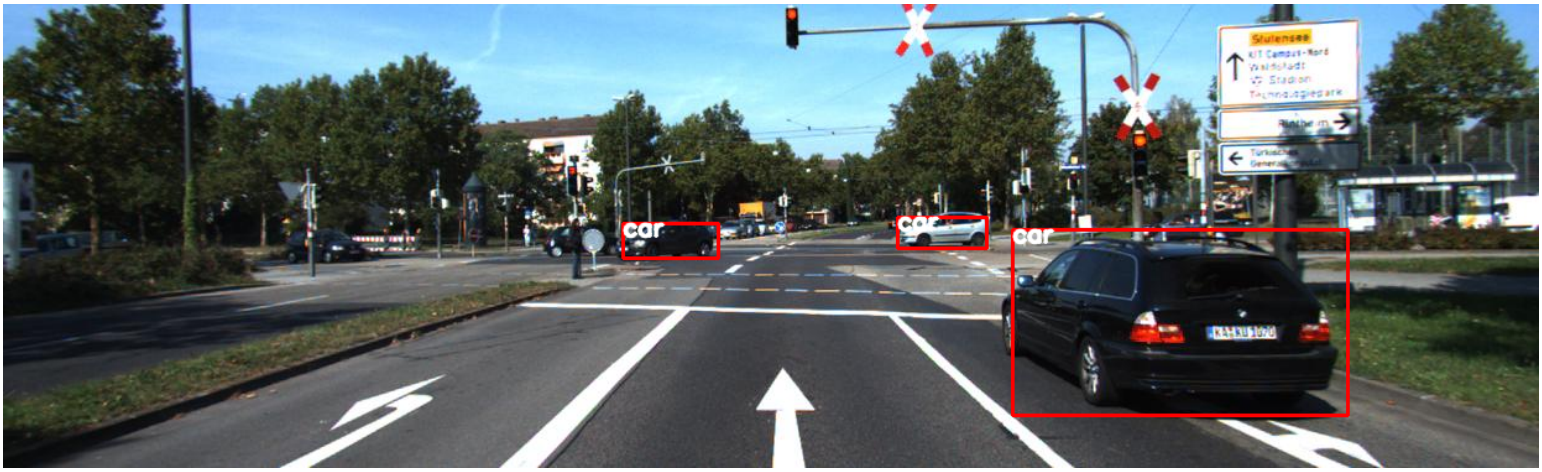
004964 - depth map

005002 - depth map

e) Object Detection

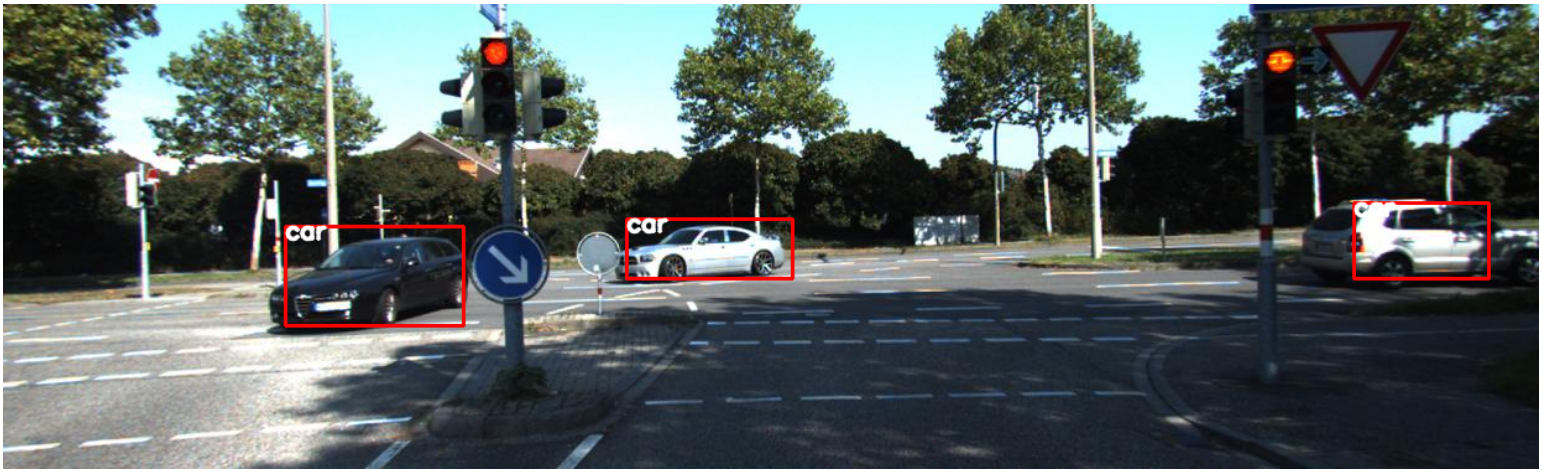
* code for the object detection and visualization is found in Appendix — detector.py

004945 detections



004964 detections

005002 detections



f) Detected Object Locations

In order to calculate the 3D location of each object, I first calculated the
1) Calculate the centre point of the detected object
2) Use the depth map to find the depth (Z) of this point
3) Since we know Z we can then compute both X, and Y
    - X = Z (x - px) / f
    - Y = Z (y - py) / f
    - since px, py, f are all in mm the point x, y must also be in mm. The conversion from pixels to mm is given by: mm_per_pixel = px / (width of image * 0.5) and py / (height of image * 0.5)

* All positions are in mm (since our intrinsic variables are also in mm)
* code for the 3D positions is found in Appendix — detector.py

004945 detected object 3D positions

```
{
  'car': [
    (3081.3618138636616, 609.4910902843111, 6863.781069087781),
    (9019.986400582717, -214.85657139518872, 48046.46748361447),
    (-3897.965137537725, 111.61380332217617, 34942.88544262871)
  ],
  'person': [
  ],
  'bicycle': [
  ]
}
```

004964 detected object 3D positions

```
{
  'car': [
    (2424.3441709076064, 156.25932465104634, 34942.88544262871),
    (13856.790702344464, -266.0128979178534, 32030.978322409646),
    (-7046.669845037708, 572.9508570538366, 18303.416184234084)
  ],
  'person': [
    (-5569.997903321401, 501.7768375440433, 16711.81477690938)
  ],
  'bicycle': [
  ]
}
```

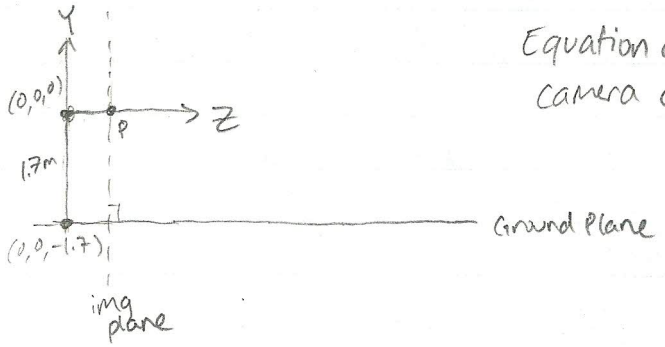
005002 detected object 3D positions

```
{
  'car': [
    (-1777.8523919989111, 278.29041628329236, 25624.782657927717),
    (-7015.544978231003, 644.5697141855662, 16015.489161204823),
    (13072.444058815536, 58.46437316875893, 18303.416184234084)
  ],
  'person': [
  ],
  'bicycle': [
  ]
}
```

1a) $f = 0.7215$ m
$p_x = 0.6096$ m
$p_y = 0.1729$ m

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.7215 & 0 & 0.6096 \\ 0 & 0.7215 & 0.1729 \\ 0 & 0 & 1 \end{bmatrix}$$

b)



Equation of ground plane in camera coordinates is

$$Y = -1.7$$

c) 3D → 2D:

$$K\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = w\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow$$

2D → 3D:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = wK^{-1}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = w\begin{bmatrix} 1.386 & 0 & -0.845 \\ 0 & 1.386 & -0.240 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Since $Y = -1.7$

$$Y = -1.7 = w(1.386y - 0.240) \Rightarrow w = \frac{-1.7}{1.386y - 0.24}$$

$$\Rightarrow Z = w = \frac{-1.7}{1.386y - 0.24}$$

we now have an expression for the 3D point $\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$ based only on (x, y) in 2D.

2 a) The disparity at each point is calculated as follows...
- for each point P_l in the left image:
  - find its corresponding match in the scanline ($y_l = y_r$) of the right image (using SSD or Norm. correlation)
  - calculate disparity at point to be
    $(x_l - x_r)$
  - calculate depth at point to be
    $$Z = \frac{f \cdot T}{x_l - x_r}$$

b) Find Fundamental Matrix F
- Find at least 7 matches between left and right images
- Solve for F

$$\begin{bmatrix} x_{r1}x_{l1} & x_{r1}y_{l1} & x_{r1} & y_{r1}x_{l1} & y_{r1}y_{l1} & y_{r1} & x_{l1} & y_{l1} & 1 \\ \vdots & & & & \vdots & & & \vdots & \\ x_{rn}x_{ln} & x_{rn}y_{ln} & x_{rn} & y_{rn}x_{ln} & y_{rn}y_{ln} & y_{rn} & x_{ln} & y_{ln} & 1 \end{bmatrix} \vec{f} = \vec{0}$$

- Given F we can compute homographies H_{ij} that make both images parallel
  - ↳ $[e_{ij}]_x F_{ij} = H_{ij}$ (Homography that maps points from image i to image j)
  - where e_{ij} is the epipole and can be computed as the left null-vector of F_{ij}
    - ↳ can use cv.SVD.solveZ(F^T) to find it
  - $[e_{ij}]_x$ refers to the 3x3 skew-symmetric operator
- Once we have the homographies H_{ij} we can use it to transform each image plane to be parallel
- Once parallel image planes are calculated, we can solve for depth just like you would for parallel cameras (part a)

Appendix

---

depth.py

```
import numpy as np
from skimage import io
from skimage.color import rgb2gray
from scipy import ndimage

# CAMERA INTRINSICS (in mm)
f = 721.5377
px = 609.5593
py = 172.854
T = 532.7119288

def depth_map(disparity_map):
    disparity_map[disparity_map<=0] = 1e-2 # approximate 0
    depth_map = f*T/(disparity_map)
    return depth_map

def main():
    img1_path = 'data/test/results/004945_left_disparity.png'
    img2_path = 'data/test/results/004964_left_disparity.png'
    img3_path = 'data/test/results/005002_left_disparity.png'
    disp_img1 = io.imread(img1_path, as_gray=True).astype(float)
    disp_img2 = io.imread(img2_path, as_gray=True).astype(float)
    disp_img3 = io.imread(img3_path, as_gray=True).astype(float)

    # apply gaussian blur to remove noise in disparity map
    # if we dont do this then we can not visualize the map b/c the noise
    # incorrectly scales the visualized image
    disp_img1 = ndimage.gaussian_filter(disp_img1, sigma=2)
    disp_img2 = ndimage.gaussian_filter(disp_img2, sigma=5)
    disp_img3 = ndimage.gaussian_filter(disp_img3, sigma=6)

    depth_map1 = depth_map(disp_img1)
    depth_map2 = depth_map(disp_img2)
    depth_map3 = depth_map(disp_img3)

    # normalize depth max for display only
    max = np.max(depth_map1)
    depth_map1 = depth_map1/max
    max = np.max(depth_map2)
    depth_map2 = depth_map2/max
    max = np.max(depth_map3)
    depth_map3 = depth_map3/max

    io.imsave('004945 - depth_map.png', depth_map1)
    io.imsave('004964 - depth_map.png', depth_map2)
    io.imsave('005002 - depth_map.png', depth_map3)

if __name__ == '__main__':
    main()
```

detector.py

```
import numpy as np
from skimage import io
from skimage.draw import polygon_perimeter
from skimage.color import rgb2gray
from scipy import ndimage
import scipy.io as sio
import cv2
from depth import *

# Parameters: dets
#   a dictionary of 'car', 'person', and 'bicycle' detections
#   with each row in the value matrix being [xleft,ytop,xright,ybottom,id,score]
# Return: detections
#   a dictionary of all detections
#   key: 'car', 'person', or 'bicycle'
#   value: (top_left_point, bottom_right_point) of detected object
def get_detections(dets):
    i = 0
    detections = {'car': [], 'person': [], 'bicycle': []}

    # order is always car, person, bicycle
    for obj in ['car', 'person', 'bicycle']:
        DS = dets[i][0]
        n_detections = DS.shape[0]

        if n_detections != 0:
            for j in range(n_detections):
                top_left = (DS[j,0], DS[j,1])
                bottom_right = (DS[j,2], DS[j,3])
                box = (top_left, bottom_right)
                detections[obj].append(box)
        i=i+1

    return detections



# Parameters: detections, img
#   detections: a dictionary of bounded boxes around detected objects
#               key: 'car', 'person', 'bicycle'
#               value: (top_left_point, bottom_right_point) of detected object
#   img: the image that detected objects are from
# Return:
#   a image with bounded boxes drawn and labels displayed
def outline_detected_objects(detections, img):
    for obj, detect_list in detections.items():
        for i in range(len(detect_list)):
            (top_left, bottom_right) = detect_list[i]
            # cast to int because opencv draw only accepts ints
            top_left = (int(top_left[0])+1, int(top_left[1])+1)
            bottom_right = (int(bottom_right[0])+1, int(bottom_right[1])+1)

            if obj=='car':
                color = (0,0,255)   # red in opencv
            elif obj=='person':
                color = (255,0,0)   # blue in opencv
            elif obj=='bicycle':
                color = (0,255,0)   # green in opencv (I choose to use green instead of cyan)
```

```
            cv2.rectangle(img, top_left, bottom_right, color, 2)

            font = cv2.FONT_HERSHEY_SIMPLEX
            text_pos = (top_left[0], top_left[1]+10)
            cv2.putText(img, obj, text_pos, font, 0.65, (255,255,255), 2, cv2.LINE_AA)

    return img



# p_img: point on the image plane
# returns the 3D point X, Y, Z
def calculate_3D_positon(depth_map, p_img):
    x_img = p_img[0]
    y_img = p_img[1]

    # calculate conversion from pixels to mm
    # = px / (width of image * 0.5)
    mm_per_pixel_y = py / (depth_map.shape[0]/2)
    mm_per_pixel_x = px / (depth_map.shape[1]/2)

    # get depth at point (x, y)
    # -- recall that indexing using opencv coordinates is (y, x)
    Z = depth_map[y_img, x_img]
    X = Z * (x_img * mm_per_pixel_x - px) / f    # recall px, py, f are globals
    Y = Z * (y_img * mm_per_pixel_y - py) / f

    return (X, Y, Z)



# Parameters:
#       depth_map: the depth map of the image
#       detections: dictionary of object detections for keys: 'car', 'person', 'bicycle'
#                   the values are lists of (top_left, bottom_right) tuples of the bounded box
#                   around the detected object
# Return: a dictionary of the (X, Y, Z) positions of all detected objects
def calculate_object_positions(depth_map, detections):
    obj_positions = {'car': [], 'person': [], 'bicycle': []}
    for obj, detect_list in detections.items():
        for i in range(len(detect_list)):
            (top_left, bottom_right) = detect_list[i]
            x_left = top_left[0]; y_top = top_left[1]
            x_right = bottom_right[0]; y_bottom = bottom_right[1]

            # Calculate the center of the detected object
            obj_x = (x_left + x_right)/2
            obj_y = (y_top + y_bottom)/2
            p_img = (int(round(obj_x)), int(round(obj_y)))

            # Calculate the 3D position
            p_3D = calculate_3D_positon(depth_map, p_img)
            obj_positions[obj].append(p_3D)

    return obj_positions
```

```
def main():
    # get images
    img1_path = 'data/test/left/004945.jpg'
    img2_path = 'data/test/left/004964.jpg'
    img3_path = 'data/test/left/005002.jpg'
    img1 = cv2.imread(img1_path)
    img2 = cv2.imread(img2_path)
    img3 = cv2.imread(img3_path)

    # get disparity maps
    img1_dis_path = 'data/test/results/004945_left_disparity.png'
    img2_dis_path = 'data/test/results/004964_left_disparity.png'
    img3_dis_path = 'data/test/results/005002_left_disparity.png'
    disparity_map1 = cv2.imread(img1_dis_path, cv2.IMREAD_GRAYSCALE).astype(float)
    disparity_map2 = cv2.imread(img2_dis_path, cv2.IMREAD_GRAYSCALE).astype(float)
    disparity_map3 = cv2.imread(img3_dis_path, cv2.IMREAD_GRAYSCALE).astype(float)

    # get matlab data
    img1_mat = 'data/test/results/dets-test/004945_dets.mat'
    img2_mat = 'data/test/results/dets-test/004964_dets.mat'
    img3_mat = 'data/test/results/dets-test/005002_dets.mat'
    detect_results1 = sio.loadmat(img1_mat)['dets']
    detect_results2 = sio.loadmat(img2_mat)['dets']
    detect_results3 = sio.loadmat(img3_mat)['dets']

    # DETECT AND VISUALIZE OBJECTS (car, person, bicycle)
    detections1 = get_detections(detect_results1)
    detections2 = get_detections(detect_results2)
    detections3 = get_detections(detect_results3)
    img_detections_1 = outline_detected_objects(detections1, img1)
    img_detections_2 = outline_detected_objects(detections2, img2)
    img_detections_3 = outline_detected_objects(detections3, img3)

    cv2.imwrite('004945-detections.png', img_detections_1)
    cv2.imwrite('004964-detections.png', img_detections_2)
    cv2.imwrite('005002-detections.png', img_detections_3)

    # COMPUTE THE 3D LOCATION OF EACH DETECTED OBJECT
    depth_map1 = depth_map(disparity_map1)
    depth_map2 = depth_map(disparity_map2)
    depth_map3 = depth_map(disparity_map3)

    img1_positions = calculate_object_positions(depth_map1, detections1)
    img2_positions = calculate_object_positions(depth_map2, detections2)
    img3_positions = calculate_object_positions(depth_map3, detections3)

    print(img1_positions)
    print(img2_positions)
    print(img3_positions)


if __name__ == '__main__':
    main()
```