97/100

Robot Design Lab
Tutor: Teena Hassan, Mihaela Popescu

WiSe 2022/2023
Group: 09

Joel Köper
Maarten Behn
Nhu Van Nguyen

Worksheet 05

Hand-in date: January 13, 2023

14p

1 Pinhole Camera Model (15%)

1.1 What is the height of the projection (in millimeters) of object A on the focal plane? (5%)

$$h' = \frac{f}{z}h$$
$$h' = \frac{150mm}{20000mm}300mm$$
$$h' = 2.25mm$$

Nice, was wars? uff Hab den Fehler mit dem Publischern gefunden :) Beim erstellen der Publischer hast du UInt32() als Type angeben statt UInt32 ← Great! :)

1.2 What is the height (in meters) of object B in the real world? (5%)

Assuming a pixel is one millimeter in size due to variability of pixel-size and to skip the conversion:

$$h = \frac{z}{f}h'$$
$$h = \frac{35000mm}{150mm}5px$$ ← Pixel size was 15, not 5.
$$h = 11666.\bar{6}px$$
$$1166.\bar{6}px \div 1000 = \sout{1.16\bar{6}m}$$

3.5 m

1.3 What is the height of object C and what is its distance from the optical center of the camera? If you think this cannot be computed, what kind of information do you need to make this computation? (5%)

With focal length and projection size alone it is impossible to compute the distance and size, only a ratio of distance and size. To make this computation work you would need at least three parameters (here either distance or size would work).

15p

2 Quiz: Camera Calibration [15%]

Very well explained!

1. **What is the role of camera calibration? (3%)**

   Camera calibration has an important role in robotics for instance for navigation systems or 3-D scene construction. Camera calibration estimates the parameters of the camera model. For example we can build the function that allows us to represent the pinhole camera model and find a way to estimate camera parameter values. You can use these parameters to correct the lens distortion (image rectification), measure the distances, calculate the size of objects or estimate the location of the camera.

2. **What is the procedure to calibrate the robot's camera? (3%)**

   To estimate the camera parameters, you need to have 3-D points (p_c) and their corresponding image coordinates (u, v). It is crucial to get the correct focal length. One procedure includes using multiple images of a calibration pattern like a chessboard to get these correspondences. With these correspondences you can solve the camera parameters of the robot.

3. **What are the intrinsic camera parameters and what do they represent? (3%)**
   The parameters are:

   - k_u = pixel density for u
   - k_v = pixel density for v
   - u_o = image coordinate (origin dimension u of the image)
   - u_v = image coordinate
   - f = focal length

   They belong to the matrix called 'camera intrinsics matrix (K)' and K represents what happens inside the camera.

4. **What are the extrinsic camera parameters and what do they represent? (3%)**

   - R = rotation
   - t = translation
   - $[X, Y, Z]$ = World coordinates
   - X_c, Y_c, Z_c = Camera coordinates
     The origin of the camera's coordinate system is at its optical center and its x- and y-axis define the focal plane.

5. **What is the calibration matrix composed of? (3%)**

$$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} k_u f & 0 & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

   You can build this matrix from these values and it is the same as the matrix:

   $u = u_0 + k_u x \Rightarrow u = u_0 + \frac{k_u f X_c}{Z_c}$

   $v = v_0 + k_v x \Rightarrow v = v_0 + \frac{k_v f Y_c}{Y_c}$

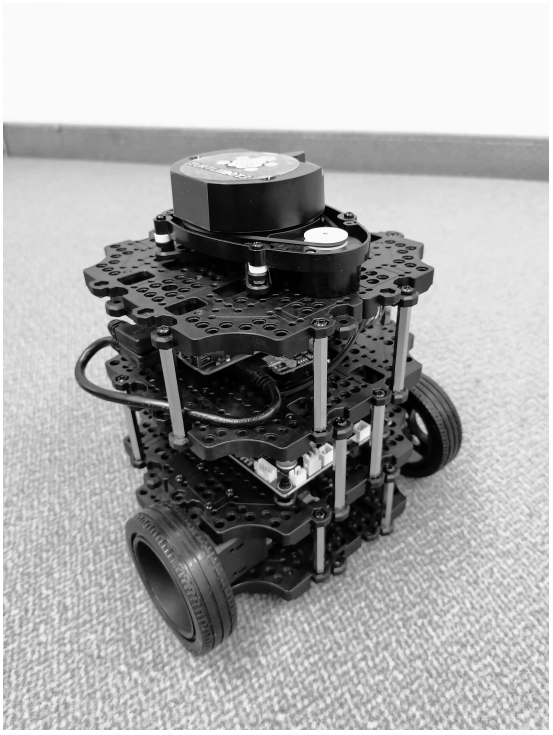
9p

3 Image Transformations (10%)

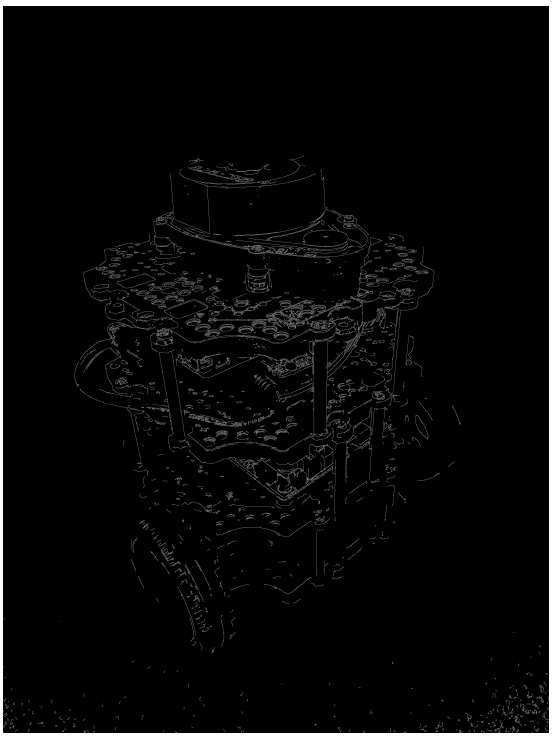
3.1 Rotated by 180°

You should provide the Python code for this task as well.

3.2 Grayscale

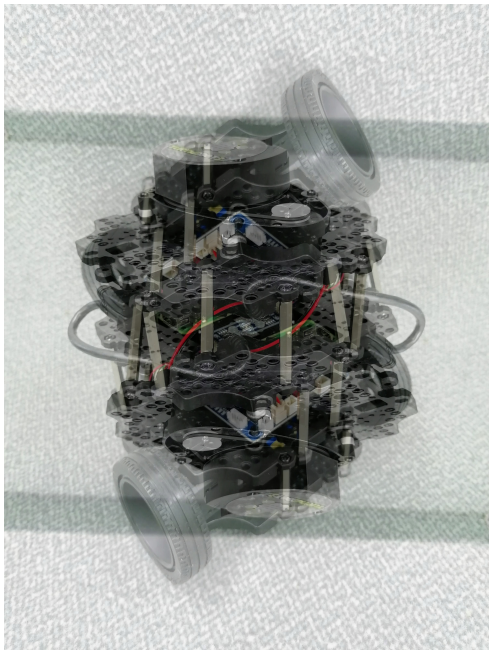


3.3 Edges



3.4 Blurred

3.5 Overlayed



15%

4 Circle Detection (15%)

4.1 detect_all_circles()

Circle Detection

5%

```
1  def detect_all_circles(image):
2      # Blur image to reduce noise, easier circle detection.
3      blur = cv2.GaussianBlur(image, (5,5),0)
4
5      # Convert image to grayscale.
6      gray = cv2.cvtColor(blur, cv2.COLOR_RGB2GRAY)
7
8      list_of_circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT,
           1, image[0].__len__() / 8, param1 = 100, param2 = 25,
           minRadius = 0, maxRadius = 0)
9
10     return list_of_circles
```

The first step in this function is to blur the image in order to smooth out some rough edges which come from the image taking process.
Then we convert the image to grayscale in order to simplify the next computation.
At last we detect the circles in the image using the HoughCircles function with the hough-gradient algorithm and return the resulting array of positions and radius.

4.2 mark_circles_on_image()

5%

Circle Detection

```
1  def mark_circles_on_image(image):
```

```
2       circles = np.uint16(np.around(detect_all_circles(image)))
3
4       image = image
5       for i in circles[0,:]:
6           image = cv2.drawMarker(image, (i[0], i[1]), color =
                (0,0,0), thickness=3, markerType = cv2.MARKER_CROSS)
7
8       return cv2.imwrite('../results/shapes.png', image)
```

Here we are rounding the positions we got from the first function to integers because we can't place a marker between pixels.
We then iterate over all the detected circles and place a cross-marker in the middle.

4.3 find_largest_circle()

Circle Detection

```
1  def find_largest_circle(image):
2       largest_circle = [0.0, 0.0, 0.0]
3
4       for i in detect_all_circles(image)[0,:]:
5           if i[2] > largest_circle[2]:
6               largest_circle = i
7
8       return largest_circle
```

Here we simply iterate over the list of circles obtained from the first function and compare, whether the current circle is larger than the currently largest and if yes store it in a temporary variable to compare against in the next iterations.

5%

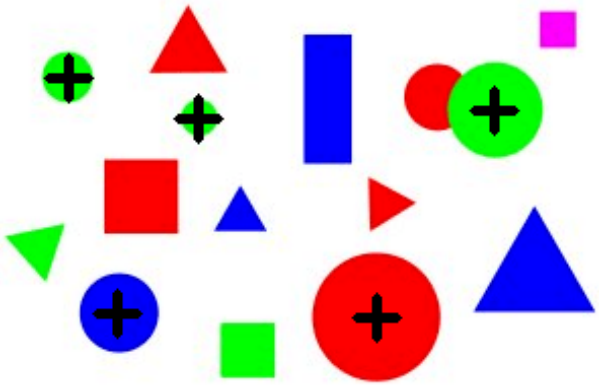


Figure 1: Marked Circles

```
[16:46] joel-schlep:scripts (main *%) | python circle_detection.py
all: [[[185.5 159.5  32. ]
  [244.5  57.5  23.9]
  [ 58.5 157.5  19.9]
  [ 33.5  41.5  12.7]
  [ 98.5  61.5   9.1]]]
largest: [185.5 159.5  32. ]
[16:46] joel-schlep:scripts (main *%) |
```

Figure 2: Circle Detection

15%

5 Polygon Detection (15%)

5.1 Detect Polygons

5%

Polygon Detection

```
1  def detect_polygons(image):
2      list_of_squares = []
3      list_of_rectangles = []
4      list_of_triangles = []
5
6      # Convert image to grayscale.
7      gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
8
9      # Apply binary threshold on the image and obtain the contours
10     _, thrash = cv2.threshold(gray_image, type=cv2.THRESH_BINARY,
           thresh=160, maxval=255)
11     contours, _ = cv2.findContours(thrash, cv2.RETR_TREE, cv2.
           CHAIN_APPROX_NONE)
12
13     # Find closed polygons in contours
14     for contour in contours:
15         shape = cv2.approxPolyDP(contour, cv2.arcLength(contour,
               True)*0.04, True)
```

Here we first convert the image to grayscale and then make everything darker than 160 black and everything brighter white.
Then we detect all edges using the RETR_TREE algorithm and find all the edges which connect to a closed shape.

5.2 Find And Mark Rectangles

Polygon Detection

```
1          x_cor = shape.ravel()[0]
2          y_cor = shape.ravel()[1]
3
4          if len(shape) == 4:
5              # Get rectangle coordinates and size: width(w),
                   height(h)
6              _, _, w, h = cv2.boundingRect(shape)
7
8              if (shape[0][0][0] == [0, 0]).all():
9                  continue
10
11             # Draw contours of rectangles on the image
12             cv2.drawContours(image, shape, 0, (0, 0, 255), 5)
13
14             # Distinguish between square and rectangle by
                   computing the aspect ratio
15             aspectRatio = float(w) / h
16             if  .8 <= aspectRatio <= 1.2:
```

```
17                  cv2.putText(image, "Square", (x_cor, y_cor), cv2.
                        FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0))
18                  list_of_squares.append(shape)
19              else:
20                  cv2.putText(image, "Rectangle", (x_cor, y_cor),
                        cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 0, 0))
21                  list_of_rectangles.append(shape)
```

To find rectangles just count the number of edges forming the closed shape and if it's 4 it's a rectangle.
If additionally the edges are roughly equal in length it's a square.
We then draw a border around the rectangles and tag the rectangles either as square or else just as rectangle.

5.3 Find And Mark Triangles

Polygon Detection

```
1
2           if len(shape) == 3:
3               # Draw contours of triangles on the image
4               cv2.drawContours(image, shape, 0, (0, 0, 255), 5)
5
6               cv2.putText(image, "Triangle", (x_cor, y_cor), cv2.
                    FONT_HERSHEY_COMPLEX, 0.5, (255, 0, 0))
7               list_of_triangles.append(shape)
8
9       # Write image to file
10      cv2.imwrite('../results/shapes_polygons.jpg', image)
11      return list_of_squares, list_of_rectangles, list_of_triangles
```

Finding the triangles is nearly identical to finding rectangles, just check for three edges instead of four and we can also skip further differentiation although you could easily distinguish between for example right-angle and equilateral triangles.

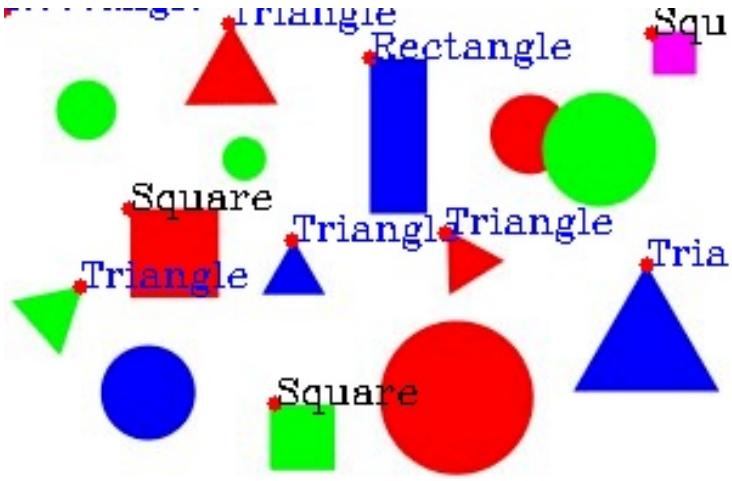


Figure 3: Marked Polygons

5%

```
[16:46] joel-schlep:scripts (main *%) | python polygon_detection.py
Squares: [array([[[111, 162]],

       [[136, 163]],

       [[135, 190]],

       [[109, 188]]], dtype=int32), array([[[ 51,  82]],

       [[ 87,  81]],

       [[ 89, 117]],

       [[ 52, 119]]], dtype=int32), array([[[266,  10]],

       [[284,   9]],

       [[285,  26]],

       [[267,  27]]], dtype=int32)],
Rectangles: [array([[[150,  20]],

       [[174,  21]],

       [[173,  85]],

       [[149,  83]]], dtype=int32)],
Triangles: [array([[[ 31, 114]],

       [[ 22, 142]],

       [[  3, 120]]], dtype=int32), array([[[264, 105]],

       [[293, 158]],

       [[234, 157]]], dtype=int32), array([[[118,  95]],

       [[131, 118]],

       [[106, 117]]], dtype=int32), array([[[181,  92]],

       [[205, 104]],

       [[183, 118]]], dtype=int32), array([[[ 92,   6]],

       [[111,  40]],

       [[ 73,  39]]], dtype=int32)]
[16:50] joel-schlep:scripts (main *%) |
```

Figure 4: Polygon Detection

14%

6 Shape Detection in ROS (15%)

6.1 Detecting Shapes in Bright Conditions

4%

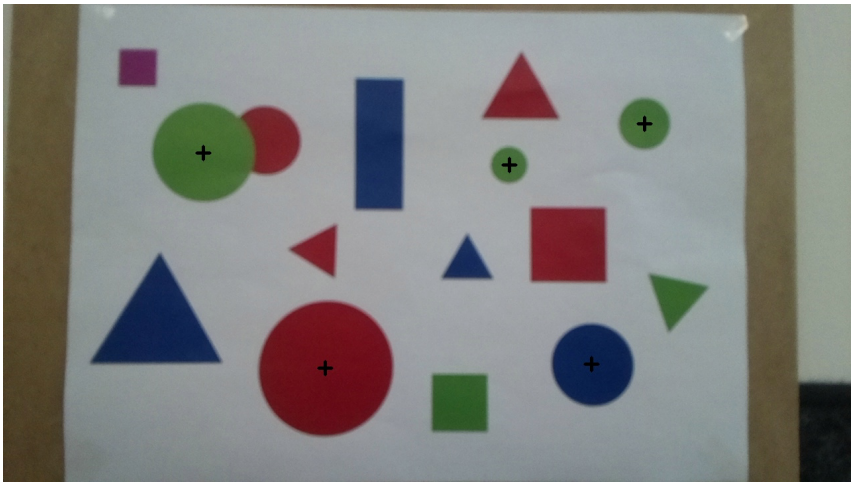


Figure 5: Circle Detection In Bright Conditions

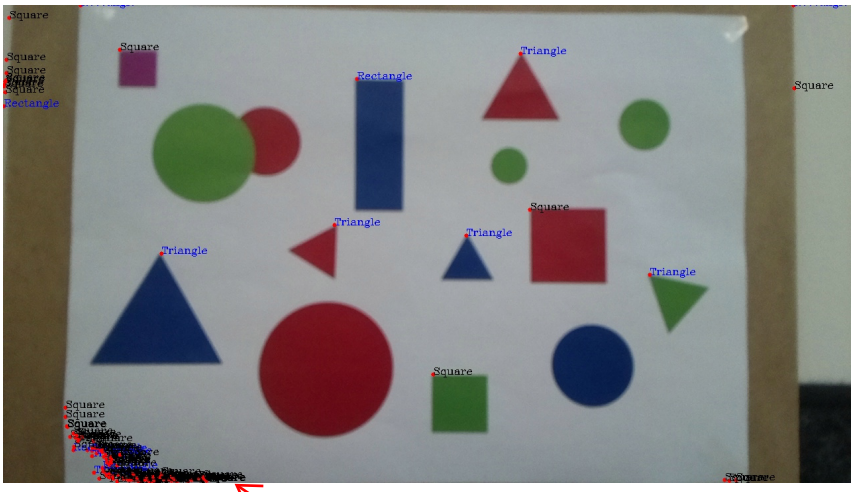


Figure 6: Polygon Detection In Bright Conditions

You can filter out the outlier polygons based on their size.

```
List of circles: [[[493.5 557.5 104.2]
  [915.5 553.5  65.2]
  [305.5 215.5  81.1]
  [998.5 169.5  41.1]
  [781.5 236.5  28.5]]]
Largest circle: [493.5 557.5 104.2]
```

Figure 7: Circle Detection In Bright Conditions

6.2 Detecting Circles in Dark Conditions

5%

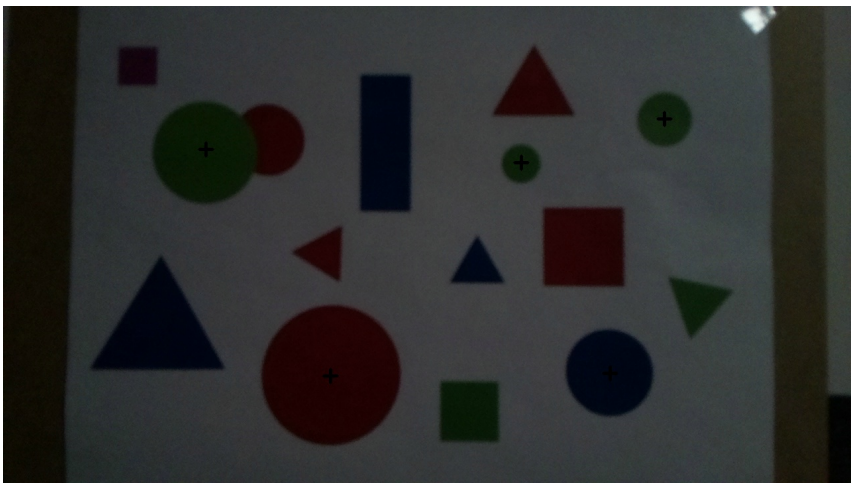


Figure 8: Caption

```
List of circles: [[[493.5 557.5 104.2]
  [915.5 553.5  65.2]
  [305.5 215.5  81.1]
  [998.5 169.5  41.1]
  [781.5 236.5  28.5]]]
Largest circle: [493.5 557.5 104.2]
```

Figure 9: Caption

Actual number of circles: 6
Number of circles detected: 5
The radius of the largest Circle: 104.2

6.3 Detecting Polygons While Moving

5%

Number of triangles: 5
Number of squares: 3
Number of rectangles: 2

15 P 7 Task Planning with PyTrees (15%)

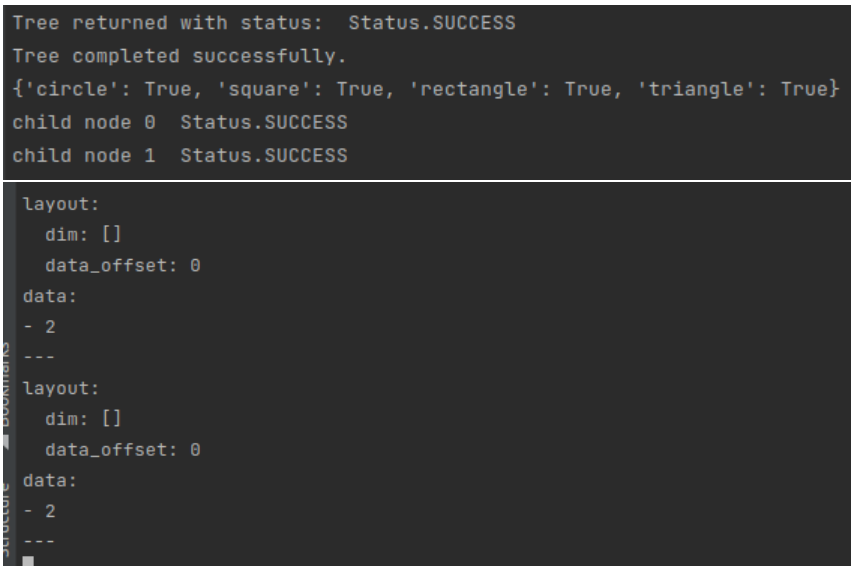


Figure 10: Screenshot of the Planning Tree and Topic /most_detected_shapes