Lab 5A- Robot Vision: Image Processing

Scope & Objective

1. Image Representation
   - Grayscale image representation, thresholding and binarization
2. Image Processing
   - Filtering, connectivity and edge detection
3. Image Analysis
   - Object segmentation, pose estimation and classification

Background

Image Acquisition

Image sensors, like many other types of sensors, acquire signal from the surrounding to obtain information pertaining to the environment. Signals can be generated from light wave, or other form of energy for example (acoustic in ultrasound imaging) by an imaging system. The process of acquiring images of the surrounding scene is an important aspect for robot applications. As robots are usually expected to accomplish specific tasks while interacting with the environment, a typical robot vision system involves processing video stream on-the-fly for timely feedback control or decision making. Hence, connecting imaging systems to the robot becomes an important aspect for robot vision.

A camera device is often used to acquire timely visual information of the surrounding. In this lab, we will connect a camera to capture images represented as digital information for processing and computational operation.

Image Processing

Behind the scenes of robot vision lies image processing operations that enhance, analyze and/or transform the image data. In robot vision, a robot makes sense of the surrounding through acquired and processed images that present useful information including colors, geometries, structures and motions.

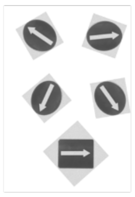
In this lab, we process images by applying thresholding, binarization and filtering. Image analysis is performed through edge detection, object segmentation, pose estimation and similarity score measurement.

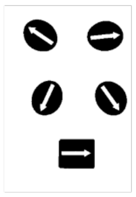
Lab Activities

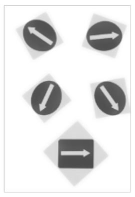
1. Image Representation and Processing

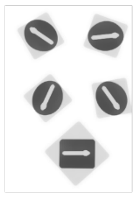
Perform the series of operations from 1 (A) to (D).

Image Representation and Image Processing


(A) Converting RGB image to grayscale


(B) Converting grayscale to Binary image by thresholding


(C) Blurring using bilateral Filter


(D) Blurring using median Filter

<u>O) OpenCV Installation and Declaration:</u>

```
>> pip install opencv-python
>> import cv2
```

<u>A) Grayscale Conversion:</u>

```
>> img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

<u>B) Image Binarization:</u>

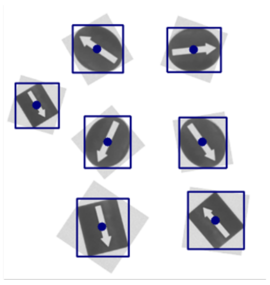
```
>> ret, thresh = cv2.threshold(img_gray, 170, 255, 0)    # to 0 or 1
```

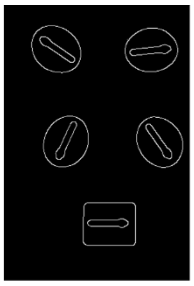
<u>C) Image Filtering:</u>

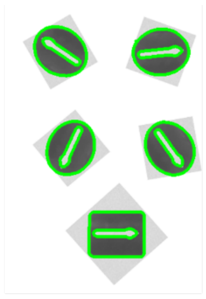
```
>> blur = cv2.bilateralFilter(img, 19, 70, 50)
blur = cv2.medianBlur(blur,9)
```

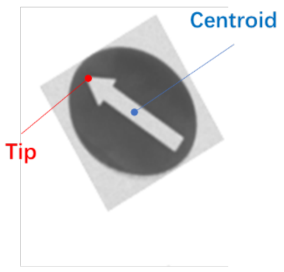
2. Image Analysis: Segmentation and Labeling

Perform the series of operations from 2 (A) to (F).


(A) Connectivity analysis

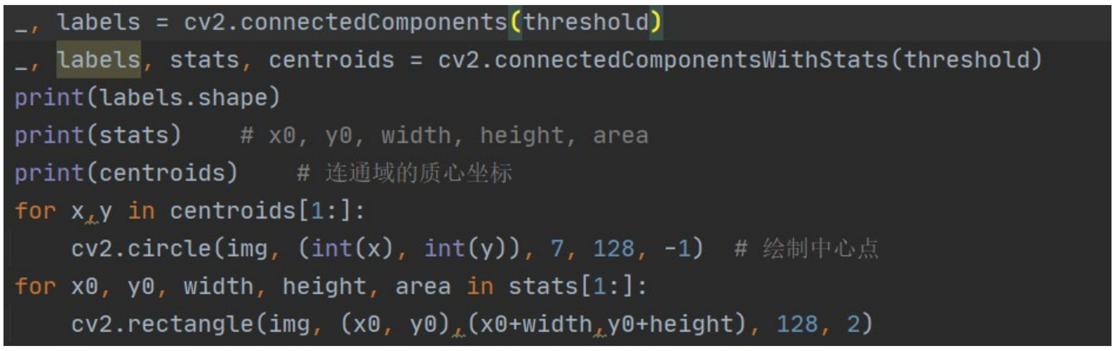

(B) Canny Edge Detection


(C) Object Segmentation



(B) Pose Estimation

A) Connectivity Analysis and Labeling
Using the binary image, obtained in Exercise 1, perform connectivity analysis to label region of connected pixels and obtain the geometrical and statistics information.



```
_, labels = cv2.connectedComponents(threshold)
_, labels, stats, centroids = cv2.connectedComponentsWithStats(threshold)
print(labels.shape)
print(stats)    # x0, y0, width, height, area
print(centroids)    # 连通域的质心坐标
for x,y in centroids[1:]:
    cv2.circle(img, (int(x), int(y)), 7, 128, -1)  # 绘制中心点
for x0, y0, width, height, area in stats[1:]:
    cv2.rectangle(img, (x0, y0),(x0+width,y0+height), 128, 2)
```

B) Edge detection

Use Canny edge detection method to obtain the contours

```
>> img = cv2.Canny(img_gray, 30, 220)
```

C) Object Segmentation

```
>> contours, hierarchy = cv2.findContours(img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
>>img_1 = cv2.drawContours(draw_img, contours, -1, (0, 255, 0), 3)   # contour
```

D) Computing Centroids (Position)

```
>> M = cv2.moments(contours)
>> center_x = int(M["m10"] / M["m00"])
>> center_y = int(M["m01"] / M["m00"])
```

E) Computing Orientation

```
_x = contours[i * 2][:, 0, :][index][0] - _center_x
_y = contours[i * 2][:, 0, :][index][1] - _center_y
xie = np.sqrt(np.square(_x) + np.square(_y))
theta = np.arccos(_x/ xie)/math.pi*180
if _y > 0:
    theta = -theta
elif _y < 0:
    theta = theta
```

F) Image Comparison (Classification)

Compare images and look at similarity Score

```
>> (score, diff) = compare_ssim(img1, img2, full=True)
```