Lab 6 : Robot Vision 1 Image Processing

Jie Wang, Shenghua Ye, Xuan Tang
Student ID: 3200112404 3200112420, 3200115277
Instructor: Yang Liangjing TA: Songjie Xiao

May 5, 2024

1 Introduction

In Lab 6, we started to apply our experience from control to perception. Using modern computer vision library like *OpenCV*, we can easily implement image processing in Robotics. The goal of Lab 6 is to identify block shapes, find centroids and angles, and calculate transformations relating image coordinates to real-world coordinates. This included developing Python code to process images captured by an industrial camera, segment objects based on shape and orientation, and manipulate these using a robotic arm UR3e.

2 Methodology

1. **Image Acquisition** Images were captured manually and stored using ***'realsense_camera.py'***, a provided Python script. They are input for our OpenCV-based processing. Notice the new camera has a higher resolution, so we need to take more care on the parameter setting.

2. **Image Processing and Analysis**

   - **Thresholding and Binarization**: These techniques were used to simplify the image for further processing.
   - **Edge Detection and Object Segmentation**: Utilized Sobel and Canny operators to detect edges and segmented the blocks based on these edges.
   - **Shape Classification and Pose Estimation:** Template matching was applied to classify shapes. The centroid and orientation of each block were calculated using moments.

3 Results

Images were processed to detect and classify various block shapes within the camera's view. The system successfully calculated the centroids and orientations for each block. The transformation from image to world coordinates was accurately computed, enabling the robotic arm to precisely move and organize the blocks in the designated area. Debugging and iterative testing were crucial in refining the image processing algorithms and ensuring accurate arm movements.

Figure 1: Detected Contour for Single Rectangular Block

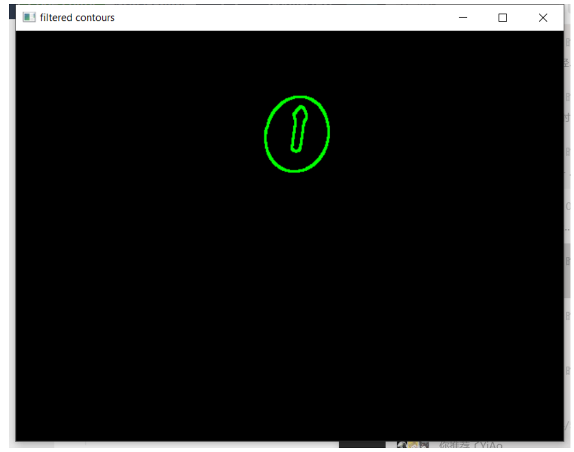

Figure 2: Detected Contour for Single Rectangular Block

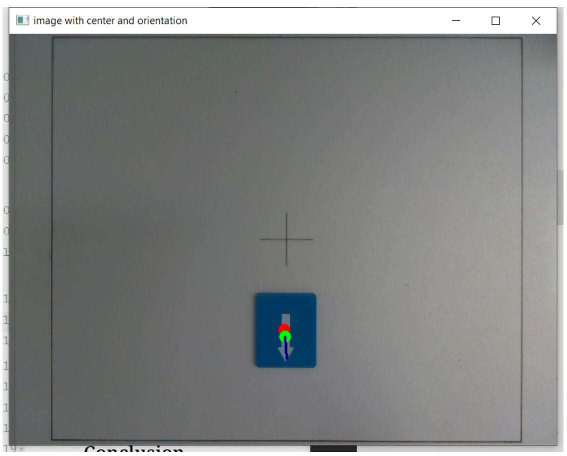

Figure 3: Detected Centroid and Orientation for Single Rectangular Block

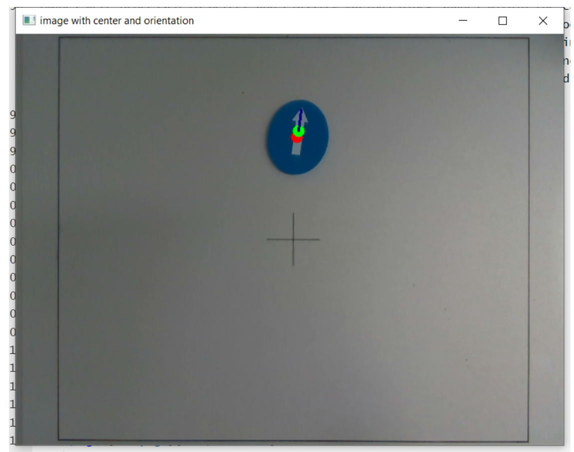

Figure 4: Detected Centroid and Orientation for Single Rectangular Block

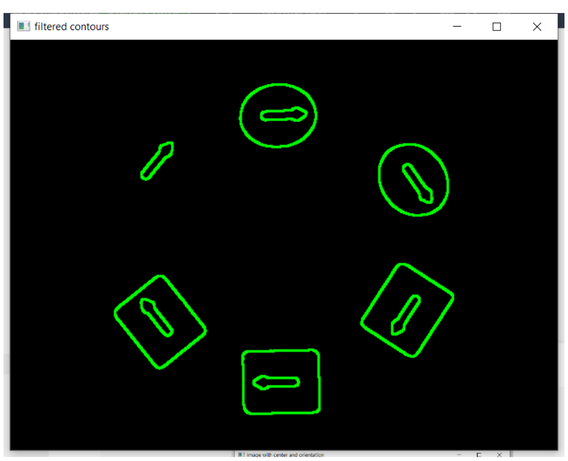

Figure 5: Detected Contours for All 6 Blocks

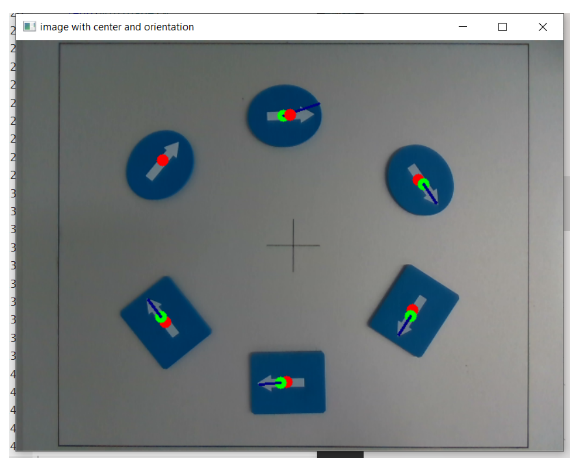

Figure 6: Detected Centroids and Orientations for All 6 Blocks

4 Discussion

The lab demonstrated the effective use of OpenCV functions for real-time image processing in a robotics. In our development, we encountered two major problems:

1. How to ensure the accuracy of edge detection and object segmentation. They were critical for subsequent robot manipulation tasks.

2. How to adjust parameters like thresholds and filter sizes were necessary to adapt to different block orientations and lighting conditions.

5 Conclusion

This lab provides us hand-on experience in combining image processing with practical robotics applications. It highlighted the importance of precise image analysis in directing robotic actions and the potential of computer vision in industrial automation settings. In lab 7, we will apply the perception information for robot arm manipulation next week.

6 References

1. ECE470 Lab6 Manual: Robot Vision 1 Image Processing.

2. Lynch, Kevin M., and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control.* Cambridge University Press, 2017. Preprint version.

3. Universal Robots A/S. ”User Manual - UR3e e-Series - SW 5.10 - English international.” Last modified on Mar 16, 2021. Available online: https://www.universal-robots.com/download/manuals-e-seriesur20ur30/user/ur3e/510/user-manual-ur3e-e-series-sw-510-english-international-en/

4. Clearpath Robotics. ”URE ROS Setup.” UR Setup Tutorial 0.0.0 documentation. Available online: https://www.clearpathrobotics.com/ure-ros-setup-universal-robots-ros-driver/

5. Universal Robots. ”UR3e: Ultra-lightweight, compact cobot.” Available online: https://www.universal-robots.com/products/ur3-robot/

7 Appendix: Code Implementation

```python
import cv2
import numpy as np

class ImageProcess():
    def image_process(self, img_path):

        # read image from img_path
        img_src = cv2.imread(img_path)

        if img_src is None:
            print('Source Image is None, Please check image path!')
            return

        img_copy = img_src.copy()

        ############## Your Code Start Here ##############
        # TODO: image process including filtering, edge detection, contour detection and so on.

        # Important: check contours and make sure the contour is available.
        # For one block, there will be 4 contours, 2 for rectangle or ellipse outside and 2 for arrow inside.
        # For one rectangle edge, there will be 2 contours in image resolution.
        # Tips: use cv2.contourArea(contour) as thres hold to filter out the contour.

        img_blur = cv2.bilateralFilter(img_copy, 5, 130, 30)
        img_blur = cv2.medianBlur(img_blur, 7)
        # img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
        img_gray = cv2.cvtColor(img_blur, cv2.COLOR_BGR2GRAY)
        x_grid = cv2.Sobel(img_gray, cv2.CV_16SC1, 1, 0)
        y_grid = cv2.Sobel(img_gray, cv2.CV_16SC1, 0, 1)
        img_canny = cv2.Canny(x_grid, y_grid, 30, 180)

        # cv2.imshow("canny_image", img_canny)
        # cv2.waitKey()
        # cv2.destroyAllWindows()

        _contours, hierarchy = cv2.findContours(img_canny, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

        # print(_contours)
        threshold_small = 1
        threshold_large = 200000
        contours = []

        for cnt in _contours:
            area = cv2.contourArea(cnt)
            # print("area: ", area)
            if area > threshold_small and area < threshold_large:
                print("area: ", area)
                contours.append(cnt)

        contour_image = np.zeros_like(img_copy)
        cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)
        cv2.imshow("filtered contours", contour_image)
```

```python
        cv2.waitKey()
        cv2.destroyAllWindows()
        # print(contours)

        ############### Your Code End Here ###############

        # length of contours equals to 4 times the number of blocks
        print("length of contours: ", len(contours))

        ############## Your Code Start Here ##############
        # TODO: compute the center of contour and the angle of arrow,
        # as well as match shapes of your block
        center_value = []
        shape = [] # 0 represents rectangle while 1 represents ellipse
        theta = []
        for i in range(len(contours) // 2):
            if i % 2 == 0:# Even
                ############## Your Code Start Here ##############
                # TODO: compute the center of external contour (rectangle/ellipse) and match shapes of your block
                # Tips: ret = cv2.matchShapes(contour1, contour2, 1, 0.0)

                # cv2.circle(draw_img, (int(center_x), int(center_y)), 7, [0,0,255], -1)

                rect_likelihood = cv2.matchShapes(contours[i * 2], self.contours_rect[1], 1, 0.0)
                elip_likelihood = cv2.matchShapes(contours[i * 2], self.contours_elip[1], 1, 0.0)
                if rect_likelihood < elip_likelihood:
                    shape.append(0)
                else:
                    shape.append(1)

                N = cv2.moments(contours[i * 2])
                x = int(N["m10"] / N["m00"])
                y = int(N["m01"] / N["m00"])
                cv2.circle(img_copy, (int(x), int(y)), 7, [0,0,255], -1)
                center_value.append([x,y])
                print(len(center_value))
                ############### Your Code End Here ###############

            else:
                # TODO: compute the center of internal contour (arrow) and compute the angle of arrow
                N = cv2.moments(contours[i * 2])
                _center_x = int(N["m10"] / N["m00"])
                _center_y = int(N["m01"] / N["m00"])
                # draw a circle on the center point
                cv2.circle(img_copy, (int(_center_x), int(_center_y)), 7, [0,255,0], -1)

                ############## Your Code Start Here ##############
                # TODO: compute the angle of arrow
                # Tips: compute the distance between center point of external contour and every point of internal contour,
                # find the furthest point, then you can compute the angle.
                print("i//2: ", i//2)
                print("center_value: ", center_value)
```

```
                x_ex = center_value[i // 2][0]
                print("x_ex: ", x_ex)
                y_ex = center_value[i // 2][1]
                print("y_ex: ", y_ex)

                max_dis = 0
                x = 0
                y = 0
                for cnt in contours[i * 2]:
                    dis = np.sqrt((cnt[0][0] - x_ex) ** 2 + (cnt[0][1] - y_ex) ** 2)
                    if dis > max_dis:
                        x = cnt[0][0]
                        y = cnt[0][1]
                        max_dis = dis

                theta_tmp = np.arctan2(_center_y - y, _center_x - x)

                cv2.line(img_copy, (_center_x, _center_y), (x, y), (128, 0, 0), 2)

                theta.append(theta_tmp)


                ############### Your Code End Here ###############

        cv2.imshow('image with center and orientation', img_copy)
        cv2.waitKey()
        cv2.destroyAllWindows()

        return center_value, shape, theta
```