LAB 6

Robot Vision I: Image Processing

6.1 Important

Read the entire lab before starting and especially the "Grading" section so you are aware of all due dates and requirements associated with the lab. There will be only one online session for this lab, so it is important that you prepare for that one opportunity to work with your TA.

6.2 Scope

- Image Representation including Gray-scale image representation, Thresholding and Binarization.
- Image Processing including Filtering, Connectivity and Edge detection
- Image Analysis including Object segmentation, Pose estimation and Classification

6.3 Objectives

This is the capstone lab of the semester and will integrate your work done in previous labs with python, ROS, and forward and inverse kinematics. It will also introduce you to some basic OpenCV techniques. In this lab you will:

- Use OpenCV functions to distinguish the shape of the blocks and find the centroid and angle of them.
- Develop transformation equations that relate pixels in the image to coordinates in the world frame

- Report the world frame coordinates (x_w, y_w) of the centroid of each block in the camera's view.
- Move the block to a predefined position in the robot's work area.

6.4 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.

6.5 Tasks

In this lab, we will be using some built in functions in the **OpenCV** library to locate blocks based on their shape and find their centroids. **OpenCV** is a powerful set of open source tools that are useful for computer vision tasks. They will allow us to efficiently complete tasks without having to develop the algorithms ourselves.
In this lab, we have several randomly placed blocks with different shapes and directions under an industrial camera, as is showed in the Figure 6.2. All we need to do is to stack the block of the **same shape** in the **same direction** together. The destination is not specified. Any place in the upper right area is fine.

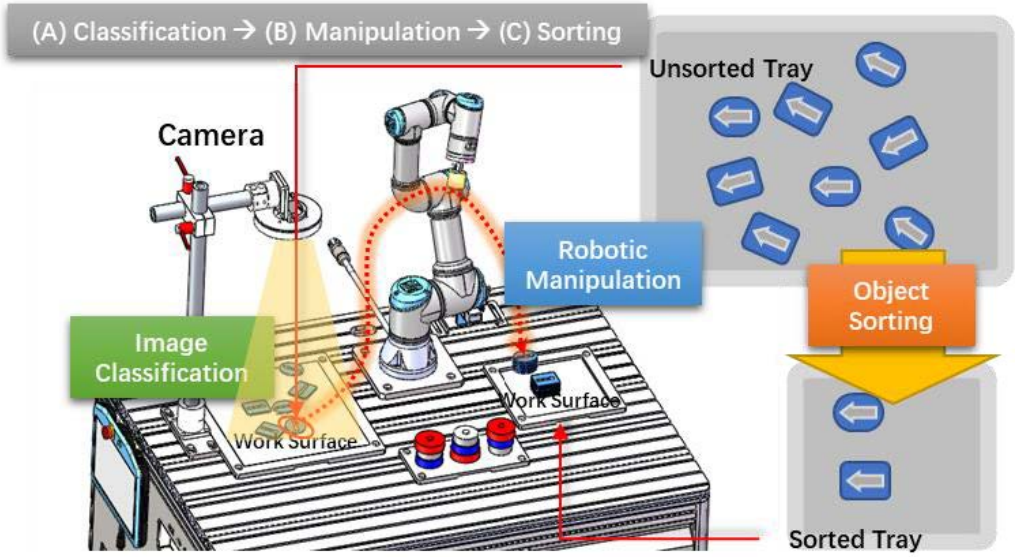

Figure 6.1: Task Description including Image Classification, Robot Manipulation and Object Sorting

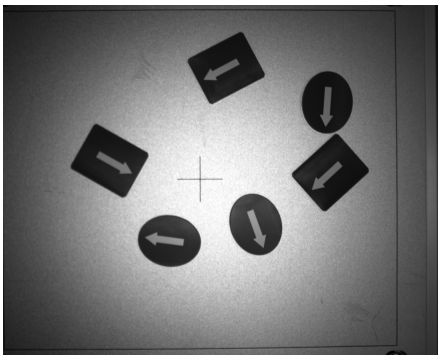

Figure 6.2: Example of randomly placed blocks

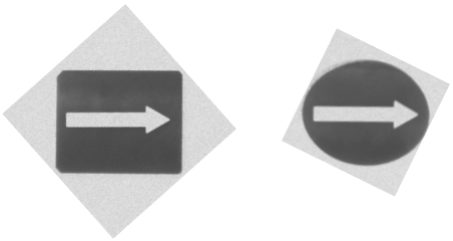

Figure 6.3: example.jpg

6.6 Procedure

6.6.1 Lab Set Up

Start as you normally do by downloading the Lab 5 starter files from the lab website, unzip them and place them in your **src** directory along with the rest of the lab packages.
If you look inside **lab5pkg_py/scripts**, you will see a number of Python files, but we will only be editing a few during this lab. A brief description of each file is provided here:

- **lab5_exec.py** - The main executable. It contains all the code to move the arm. You will edit this to complete the pick and place task.
- **lab5_func.py** - This contains the forward and inverse kinematics code. You should edit to add your solutions from Lab 3 and 4.
- **lab5_header.py** - This is a standard header file with imports and constant definitions. There is no need to edit this.
- **lab5_img.py** - This contains all the image processing, camera calibration and transformation code. You will edit this to process your image, find the camera transformation equation.
- **CMakeLists.txt** a file that sets up the necessary libraries and environment for compiling lab5_exec.py.
- **package.xml** This file defines properties about the package including package dependencies.
- **README** To run lab5 code on real robot, please read it **FIRST**!
- Every time you open a new terminal, you need to run **source devel/setup.bash** in your catkin folder.

6.6.2 Image Acquisition

In this lab, we just capture the image manually. First, copy the folder **Snapshot** in folder **lab5pkg** to the **Windows** System. Then you can edit the python script **save_image.py** to change the directory where your snapshots will save before you run it to store a series of images. After that, we can use these images in our ROS programming.

6.6.3 Image Processing

In order to distinguish the shape of the blocks, we use a template matching method here. The **example.jpg** ,shown in Figure 6.3,is the given template, including a rectangular block and an elliptical one. Then the similarity measurement of the target block can help with the classification problem when comparing with the template.

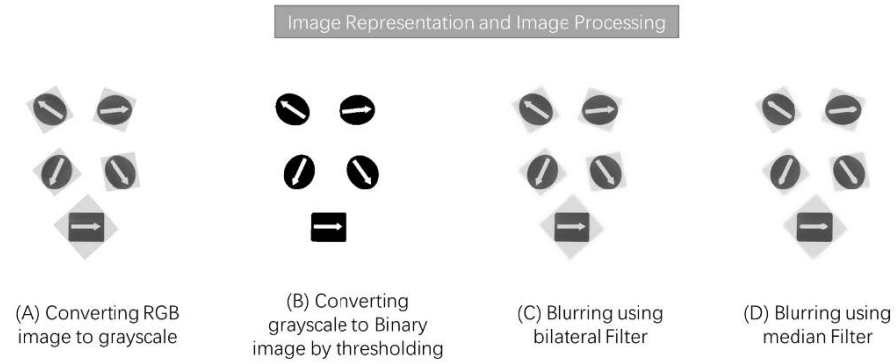

Figure 6.4: Image Representation and Image Processing

```
rospack = rospkg.RosPack()
lab5_path = rospack.get_path('lab5pkg_py')
img_path = os.path.join(lab5_path, 'scripts', 'example.jpg')
_img = cv2.imread(img_path)
_draw_img = _img.copy()

_blur = cv2.bilateralFilter(_img, 19, 130, 30)
_blur = cv2.medianBlur(_blur, 9)

_img_gray = cv2.cvtColor(_blur, cv2.COLOR_BGR2GRAY)
_xgrd = cv2.Sobel(_img_gray, cv2.CV_16SC1, 1, 0)
_ygrd = cv2.Sobel(_img_gray, cv2.CV_16SC1, 0, 1)
_img = cv2.Canny(_xgrd, _ygrd, 30, 220)
```

Code Explanation: First, we obtain the path to the template **example.jpg** and read it subsequently by the **OpenCV** method **cv2.imread**. Then we preprocess the image by **cv2.bilateralFilter** and **cv2.medianBlur**. Bilateral filter can smooth images while preserving edges by means of a nonlinear combination of nearby image values. And median filter is also common to eliminate isolated noise. After converting image to gray-scale using **cv2.cvtColor**, we can exploit **Sobel** and **Canny** operator to detect the edge of the template.

```
_w, _h = _img.shape
# rectangle
_img_1 = _img[:, :_w]
# ellipse
_img_2 = _img[:, _w:]

# rectangle
_img_1, _contours_1, hierarchy = cv2.findContours(_img_1, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
# ellipse
_img_2, _contours_2, hierarchy = cv2.findContours(_img_2, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

Code Explanation: Here we divide the image into two parts, i.e., rectangle and ellipse. Then we use **cv2.findContours** to find contours. Variables **_contours_1** and **_contours_2** store all contours, each of which is composed of a series of

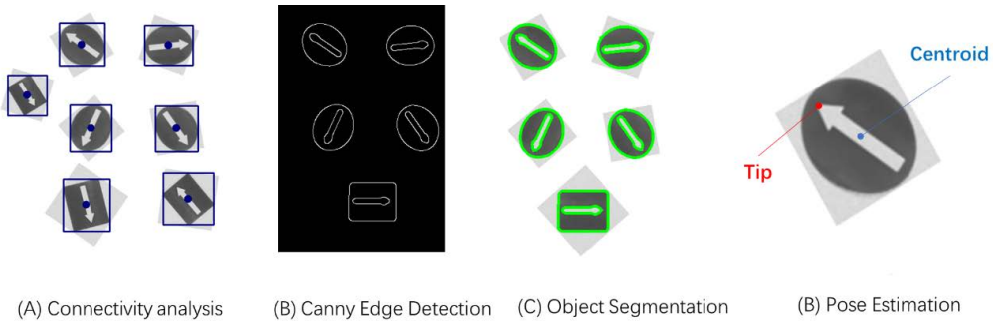

Figure 6.5: Image Representation and Image Processing

pixel points.
IMPORTANT: Before you use the contours found by **cv2.findContours**, please check and ensure these are what you want! And Try to figure out why the number of contours is twice.

```
# compute the center of a contour
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(draw_img, (int(_center_x), int(_center_y)), 7, [0,255,0], -1)
```

Code Explanation: Here we use **cv2.moments** to compute the centroids and label a circle on the center point by **cv2.circle**.

6.6.4 Debugging

This lab can be a bit tricky to work through as several parts require trial and error. Here are some suggestions:

- Approach the lab sequentially and make sure you get good results at each step.
- Comment out parts that you are not currently using (Don't forget the uncomment them though...)

6.7 Report

This lab does not require a formal lab report - only the submission of your code. You should a submit a document to Blackboard containing all of the files that you edited to complete the lab. This will be submitted to Blackboard as in the past.

6.8 Demo

Your TA will require you to run your program twice, each time with different initial block positions and verify that your program can locate the blocks and place them in the correct location. You will also demonstrate that suction feedback is working.

6.9 Grading

- 100 points, successful demonstration and submission of the code. (No code = 0 points)