Robotics: Assignment III

Robot Vision

Biomechatronics Engineering

Li-Wei Yang (B07611001), Cheng-Yen Chung (B07611048), Andrew Tay Jian An (B07611024), Hua-Ta Liang (B07611038)

Division of work:

Camera Calibration:

Coding: Li-Wei, Report: Cheng-Yen.

Object Detection:

Coding: Andrew, Report: Hua-Ta.

Part A: Camera Calibration

Cameras are one of the most commonly used sensors for a robot to gather visual/spatial information of its environment. With the help of image processing, a robot can analyze the image of the immediate environment imported from the camera and use the result to determine the appropriate action to take.

In this assignment, you will learn how to model the relationship of an image and the real environment with camera calibration and how to use the camera model to estimate the position of an object in the real world.

Camera calibration is the process of estimating the parameters of a pinhole camera model, such as focal length and principal point. This process is required for cameras before doing image processing. In this part, you should get familiar with the camera model, and the meaning of camera parameters.

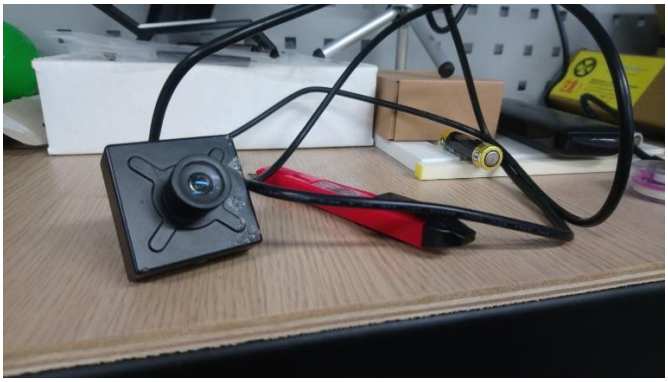
(1) There is a C/C++ implementation in OpenCV library [1]. For those who use OpenCV first time, you can refer to [2]. Follow the example code it provides. If you are more familiar with MATLAB, you can also download the toolbox [3].

(2) Print the checkerboard pattern in “AssignmentIII\part_a” on a sheet. Measure the physical size of the squares.

The size of squares are 2.25cm*2.25cm.

(3) Find a camera that you want to calibrate, e.g. your phone’s camera, webcam etc. Use the camera to capture 20 images with checkerboard for calibration. Try to shoot from different angle for each image. Please provide a clear description about what camera you use.

Note: Usually, commercial cameras – such as those found in cellphones, laptops, etc. – have an autofocus feature which will change the intrinsic parameters of the camera dynamically. You must disable this feature when taking pictures your calibration dataset to get a satisfactory result. For Android devices, if your default camera app cannot disable it you can try the Camera FV-5 Lite app.



The camera we used is a wide-angle lens with adjustable focal length. The focal length is fixed during the calibration process. Sadly, we have no more information about this camera. The spec. of a similar one we found is shown below:
https://world.taobao.com/item/651694501544.htm?spm=a21wu.10013406-tw.taglist-content.10.198354236lUYec

产品规格	
型　　号	4K800万_USB2.0
sensor 规格	2.8分之一 （高级 COMS 感光芯片 1/2.8inch)
像 素 大 小	Pixel Size　1.45μm x 1.45μm
默 认 速 度	30帧/秒
开 机 画 面	3840*2160（默认）　2952*1944（500万）
镜　　头	高清/红外/窄带：150度、120度、90度、60度、45度等
信　噪　比	41 dB
硬　　件	工业级 800万像素
功　　率	2W
工 作 电 压	5 V
工 作 电 流	220mA
使 用 环 境	室内、室外 均可
输出分辨率	1280*720/1920*1080/2048*1536/ 3840*2160等。
输 出 格 式	MJPG / YUY2 (出厂默认是：MJPG格式)
对　　焦	手动对焦
材　　质	金属外壳 （防摔、耐用寿命更长）
影 像 处 理	自动曝光、手动曝光 / 自动白平衡
接　　口	USB 接口 免驱动（USB2.0 支持 UVC 通信协议）
支持OTG协议	USB2.0　OTG
工 作 温 度	-30～70℃
低 照　度	0.01 lux
动 态 范 围	68 dB
适 用 设 备	笔记本、台式机、一体机、车载智能终端、平板电脑、工业工控电脑、ATM、广告机、安卓手机等
兼 容 系 统	XP/WIN7/WIN8/WIN10/Android4.0(安卓) /linux等系统
备　注：	清晰度好、细腻度不错，等！

清晰度 细腻度好

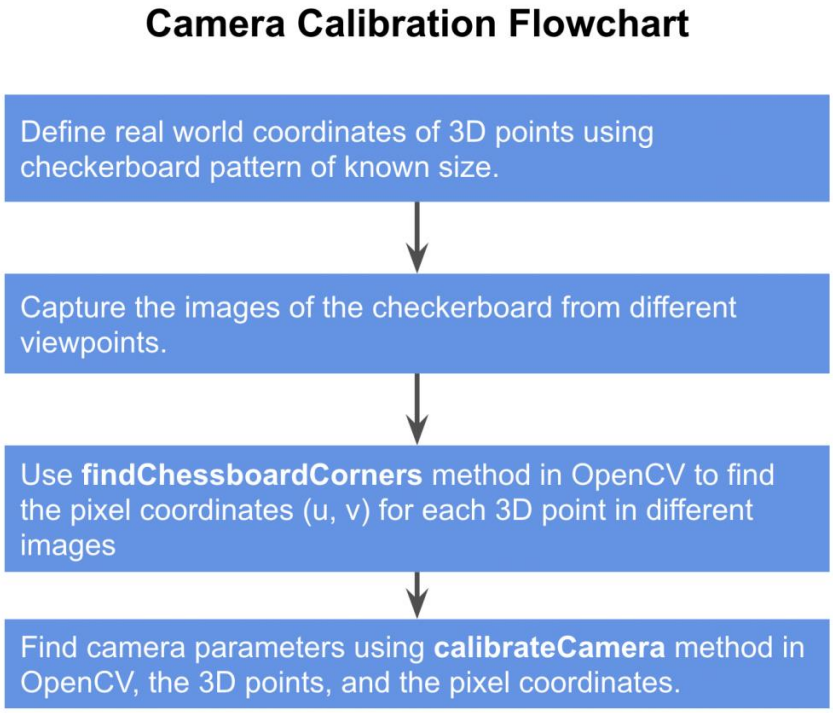
(4) Follow the instructions on the webpage to get all the intrinsic parameters. Write down your results and describe how do get it. Second, please interpret the physical meaning of each intrinsic parameter in your report in your own words. For those who use MATLAB, please refer to
http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example.html

- **Intrinsic parameters (unit: mm):**

```
Camera matrix :

[[941.78785941   0.         958.68622162]
 [  0.         936.54338321 479.77246758]
 [  0.           0.           1.        ]]
```

- **Steps:**



First, we define the dimensions of checkerboard and store all vectors of 2D/3D points for each checkerboard corner, then take pictures from various viewpoints.

Then we define the world coordinates for 3D points and we find the corners of the checkerboard and return the coordinates of each corner. (OpenCV provides "findChessboardCorners" function)

Then we do the camera calibration step. We provide all the 3D points and related 2D locations in all images. By "cv2,calibrateCamera" function, we can get parameter sets including

1. Intrinsic camera matrix
2. Lens distortion coefficients
3. Rotational and translational vector that brings the calibration pattern from the object coordinate space to the camera coordinate space (both specified as a 3x1 vector)

- **Physical meaning of each component of the Intrinsic matrix:**

$$\begin{matrix} fx & \gamma & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{matrix}$$

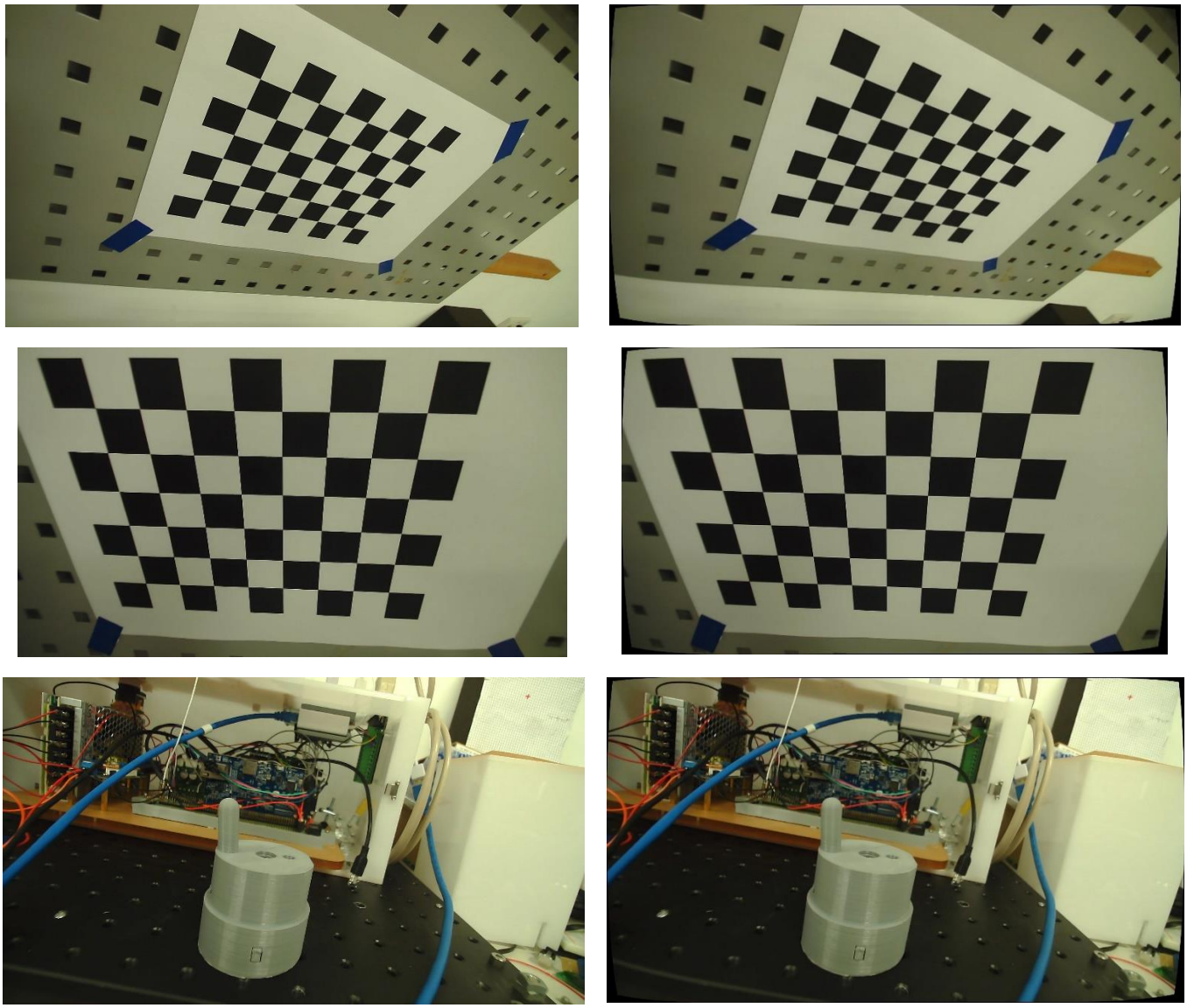
fx, fy are focal lengths in direction x, y; cx, cy are the optical center in image plane in direction x, y; γ is the skew between axis (usually equals to zero).

(5) Undistort the 20 images you captured and a new image of another object. Show those 21 sets of original and undistorted images in report and briefly comment what is the effect of this transformation.

Original images are in "final_cap", and "final_checkonly" does not contain the 21st image. The undistorted images are in "result". The program would show the image and waitKey, the user could press any key to switch to the next image.

We present some example from the sets of original and undistorted images for comparison.

Original | Undistorted



Some pin-hole camera induce radial dilation that distort the images. Take the checkerboard pattern for example, we captured the checkerboard images and found some of the lines are distorted into curves. With camera calibration, we can find the intrinsic matrix of the camera and the relation of corners across different images, then undistort the image using this relation.

However, in reality it is hard for us to observe the effect of the distortion since the images captured are slightly distorted—we could only tell the effect of transformation from the size of dark corners around the images when they are undistorted.

According to OpenCV documentation:
“Re-projection error gives a good estimation of just how exact the found parameters are. The closer the re-projection error is to zero, the more accurate the parameters we found are.”
So we use cv2.projectPoints() to “calculate the absolute norm between what we got with our transformation and the corner finding algorithm.” Then find the arithmetical mean of the errors calculated for all the calibration images.

The re-projection mean error of these images are 0.141, which is rather a small error.

Part B: Object Detection

Given an image taken from a camera, please use the algorithm you have learned in the section of binary machine vision of Lecture 6 (p.36- p.41) to write a program to process the image and to mark the foreground object(s). We have provided some examples of basic image processing functions in OpenCV (main.cpp). For MATLAB users, you can find corresponding functions in the MATLAB Toolbox[3].

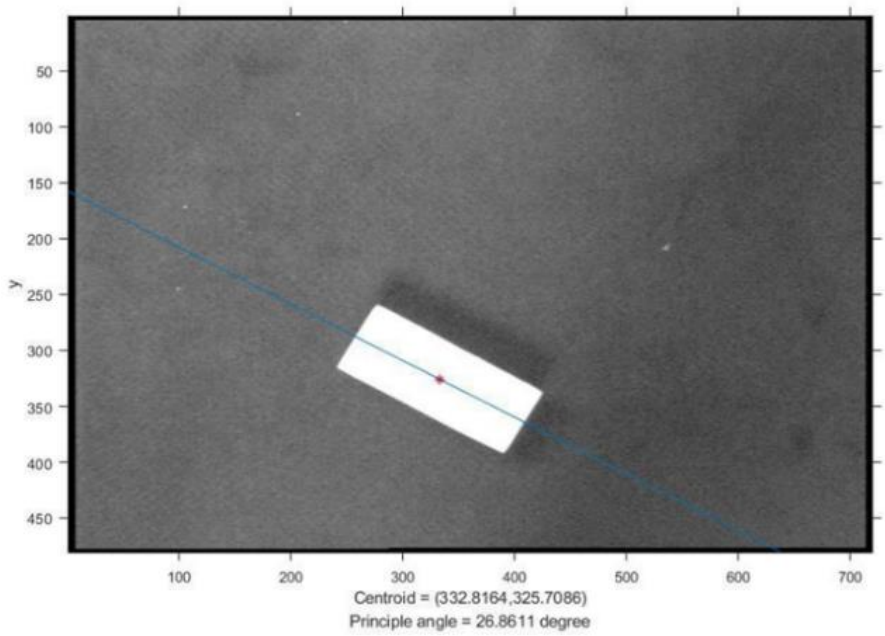
http://www.vision.caltech.edu/bouguetj/calib_doc/

Input Image:

We provide 4 images in **"AssignmentIII/part_b/images"** for testing. Your program should take the file name of the image from the standard input.

Output Results:

Your program should output the coordinate of the centroid, principal angle and principle line of all the object(s) in the following format and print the pixel coordinate on the image either.



For object detection, we mainly use "cv2::FindContours()" to find the centroids of the objects. However, before directly use the function, we must proceed the image first.

We first turn the picture into gray scale. Then, we blurred the image by using "cv2.GaussianBlur". By doing so, we can reduce the noise of the image in order to increase the accuracy of the detection. Finally, we change the image into binary (black and white) image using "cv2.threshold". So far the image processing part is finish. We can then use the findContours method to find all centroids.

In our code, the algorithm is introduced in page 35~41 of lecture ppt - "Lecture 6 Computer Vision and Camera Model". We first use cv2.moments to calculate the central moment we need and then we can calculate the Cx Cy and principle angles by using formulas provided on the lecture slides.

Finally we use matplotlib, a useful package for data visualization, to visualize our result. Notice that to have best experience of visualizing the data, please use a local computer instead of Colaboratory since the window size cannot change on Colaboratory. The final result is shown as below.

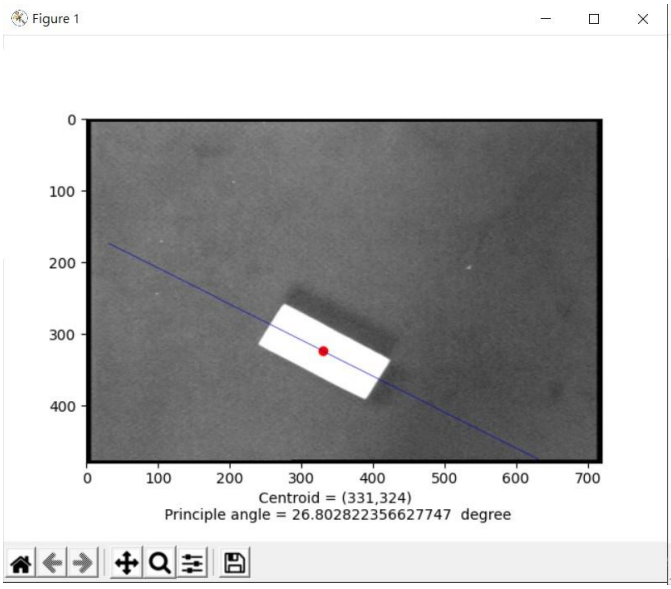


Figure 1
Centroid = (331,324)
Principle angle = 26.802822356627747 degree

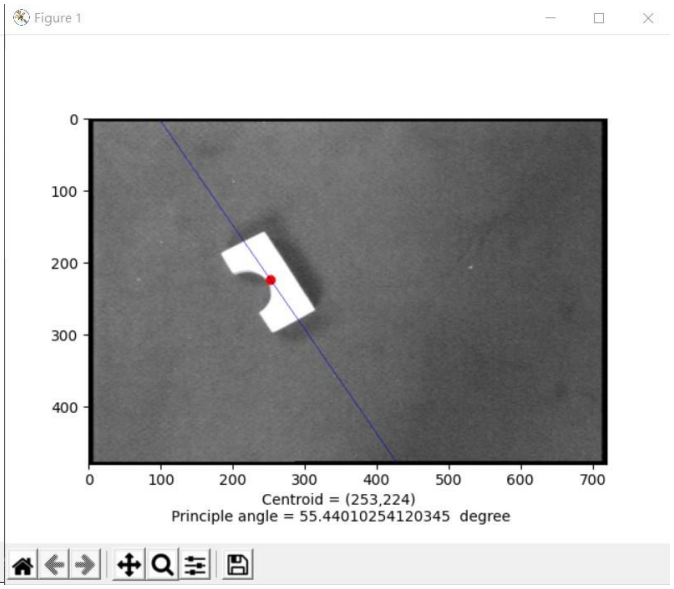


Figure 1
Centroid = (253,224)
Principle angle = 55.44010254120345 degree

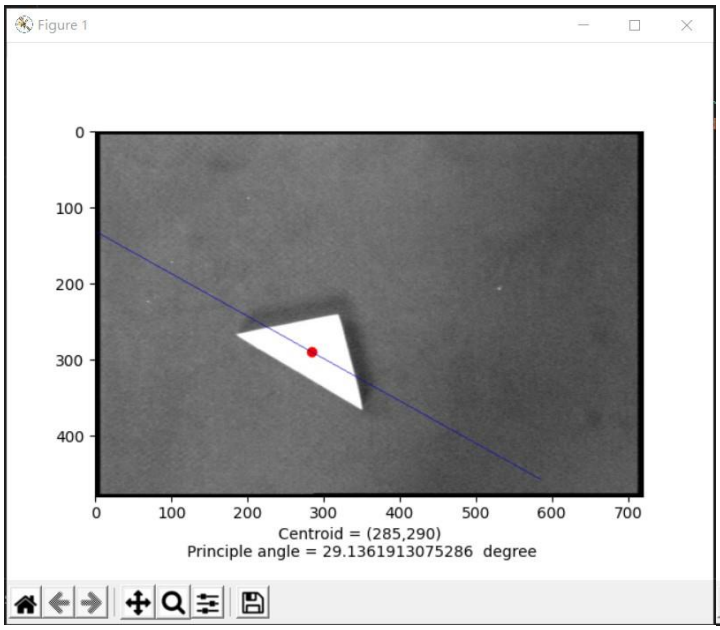
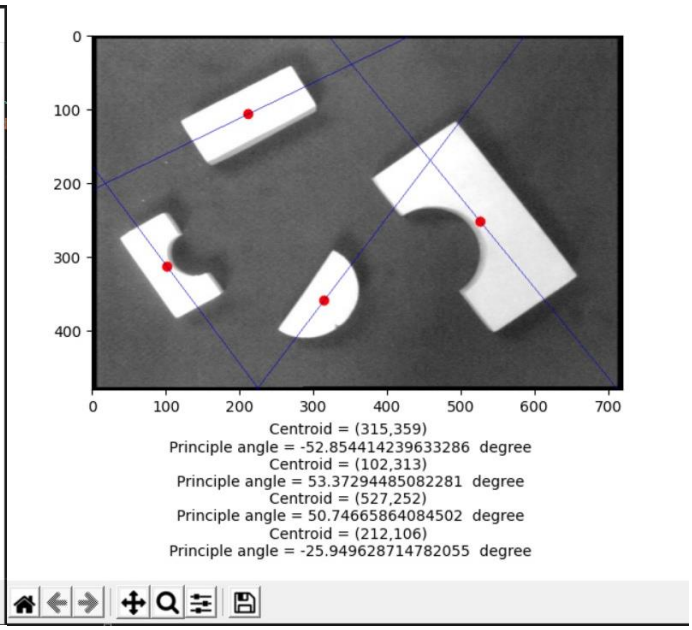


Figure 1
Centroid = (285,290)
Principle angle = 29.1361913075286 degree


Centroid = (315,359)
Principle angle = -52.854414239633286 degree
Centroid = (102,313)
Principle angle = 53.37294485082281 degree
Centroid = (527,252)
Principle angle = 50.74665864084502 degree
Centroid = (212,106)
Principle angle = -25.949628714782055 degree