**Robotics: Assignment III**

Team 16

鄭達郁 B09507016, 邱奕翔 B09901058

王泓予 B09901060, 薛尚齊 B09902037

**Part A: Camera Calibration**

**1. The camera we use**

We use the iPhone 14’s main camera, which is 12MP, and has a 2.6 mm, f/1.5 aperture. Detailed information of images are listed below.

 ****

Fig 1. Camera information

**2. Calibration result**

* **Size**

The size of each square in the chessboard is.

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

Camera intrinsic parameters include the focal length and the optical center. The camera intrinsic matrix (unit: mm) we found is shown below.

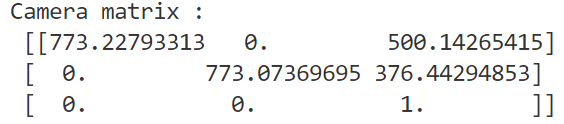


Fig 2. Camera matrix of the camera we used

* **Steps**

First, we define the world coordinates for 3D points and also define arrays *objpoints* and *imgpoints* to store 3D and 2D points for each checkerboard corner respectively, and then take photos on the checkerboard from different orientations.

Applying the *findChessboardCorners* function from the OpenCV library, we may find the corners of the checkerboard and their coordinates.

After the above steps are done, we then do the camera calibration. Knowing all the 3D points and related 2D relations in each image, we can get

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

by calling the *calibrateCamera* function in the OpenCV library.

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

: Focal length in x-direction

: Focal length in y-direction

: Optical center in image plane in x-direction

: Optical center in image plane in y-direction

: The skew between axis (usually equals to zero).

* **Program task**

Original checkerboard images are in the *calibrate\_src* folder, and the original image is in the input folder. The undistorted images will be generated into the output folder after running the program.

Our program will show the original images with their corners marked and then undistorted images. The user may press any key to switch to the next image. The information we get from *calibrateCamera* method will be printed in the console when the program gets to its end.

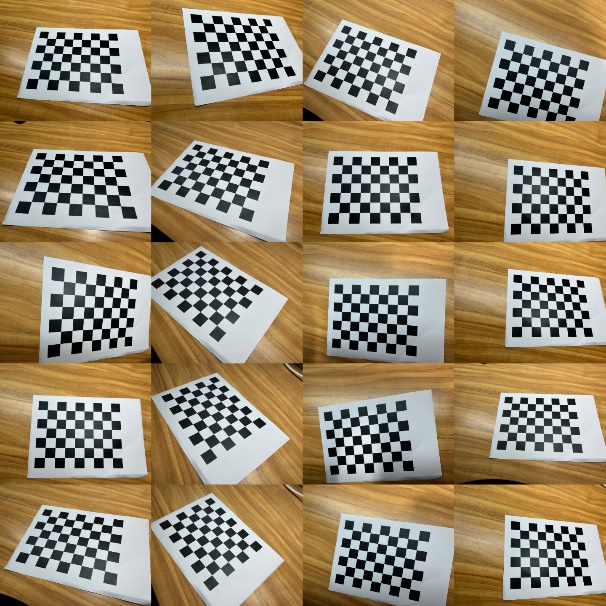
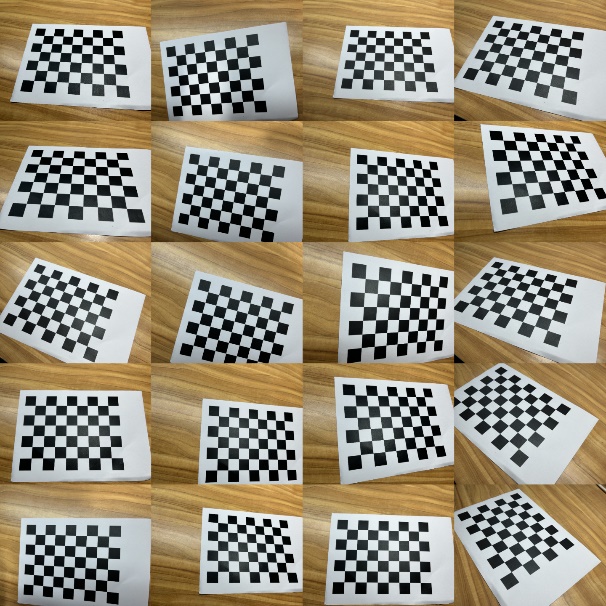
 

Fig 4. Undistorted 20 images

Fig 3. Original 20 images

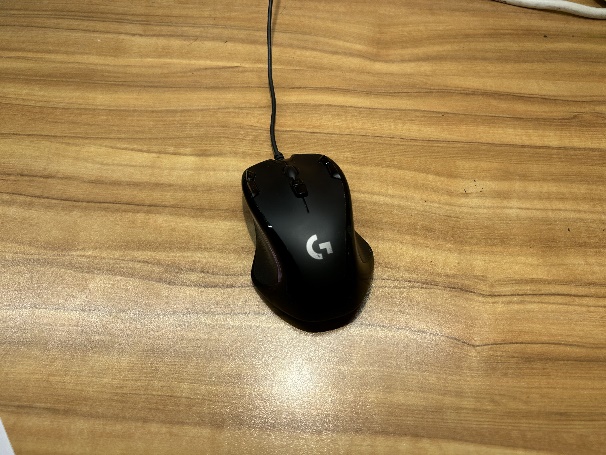
 

Fig 6. Undistorted image

Fig 5. Original image

* **Discussion**

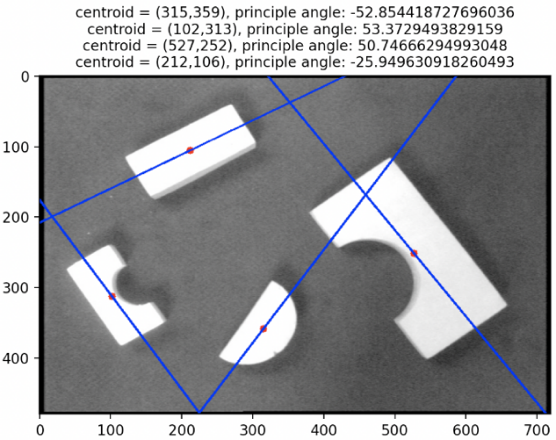
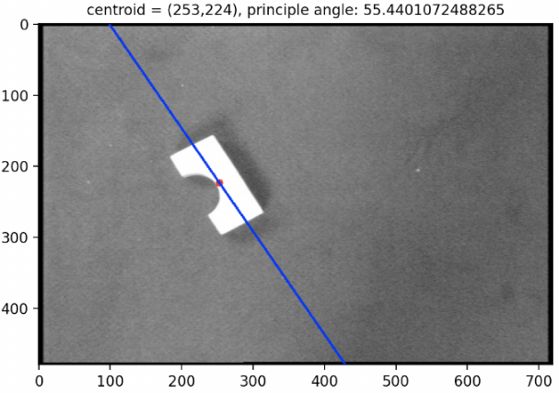
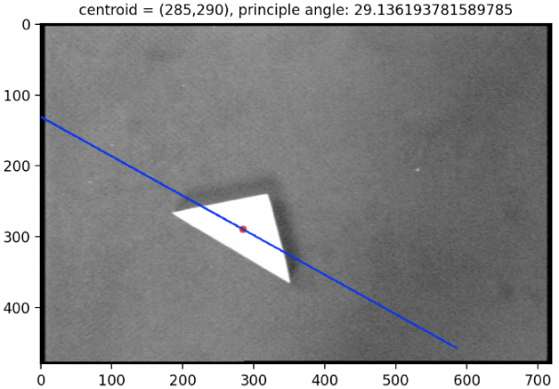
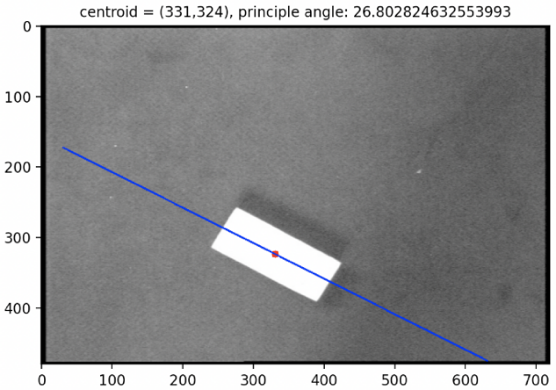
Some pin-hole cameras induce radial dilation which makes the images distort. For example, we observed the original checkerboard images and found some of the lines are distorted into curves in the checkerboard pattern. Applying camera calibration, we can find the camera’s intrinsic matrix and the relation of corners among different images. Using the relation we found, we may then undistort the image.

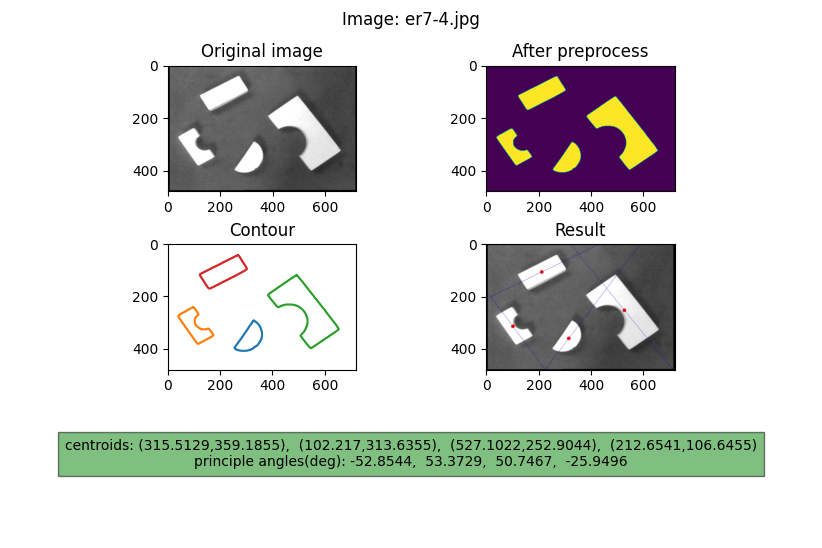
However, in reality, it’s not that easy to observe the effect of the undistortion since the images we captured are only slightly distorted, but we still can tell the effect of transformation from the size of dark corners around the images when they are undistorted.

According to OpenCV documentation, Re-projection error gives a good estimation of just how exact the found parameters are, thus we use *projectPoints* method in the OpenCV library 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 is 0.0412, which is rather a small error, showing that the parameters we found are quite accurate.

**Part B: Object Detection**





In this part, we used OpenCV to process the images. The *FindContours()* function can find the contours of each object, but we have to preprocess the images before applying it. First, we used *cvtColor(COLOR\_BGR2GRAY)* to turn the picture into grayscale. Then, we used *GaussianBlur()* to blur the image a bit so that the noise can be reduced thus increasing the accuracy. Also, we make the images become binary images by using *threshold()*. After that, we can apply *FindContours()* and get the contours.

For each object detected, we can calculate the central moment with *moments()*. Then we can calculate the centroids and principle angles as taught in class. With *line()* and *circle()*, we can plot the centroids and the principal line on the images.

**Division of work**

* Image collecting: B09507016
* Coding: B09902037, B09901058
* Code review: B09507016, B09901060
* Report:
  + PartA: B09507016, B09901060
  + PartB: B09902037
* Report revision: B09901058

**GitHub link**

<https://github.com/ZOIjimmy/Robotics/tree/master/AssignmentIII>

**References**

[1] <https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html>