Robotics — Homework #3

Lecture 6: Computer Vision and Camera Model

Due Date: 11/17/2025 before **13:20** on NTU COOL

Name: ______________ **Student ID:** ______________ **Department:** ______________

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 or even depth information from an advanced RGB-D camera, 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, we will explore 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. We will use `OpenCV` as our primary tool in `Python` programming language to complete this assignment.

Instructions:

- Write the report in **English** and keep it brief and concise.
- For part B, C (and D if implemented), you should present the results by marking it directly on the output image and include the output image in your report.
- Include the division of work within your team in the report.
- Submit a single zip file named `hw3_team<team_number>.zip` that contains:
  - Report in pdf format named `hw3_report.pdf`.
  - Source code of part A, B, C (and D if implemented) in `hw3_a.py`, `hw3_b.py`, `hw3_c.py` (and `hw3_d.py` if implemented).
  - `requirements.txt` file listing all the required libraries to run your code.
- **Only one member per team should submit the zip file.**

Part A — Camera Calibration (30%)

Camera calibration is the process of estimating the parameters of a pinhole camera model, including the focal length and principal point. This process is essential before conducting any image processing. In this section, you should become acquainted with the camera model and the meaning of each camera parameter.

1. Print the image `hw3/a/checkerboard.png`, which displays the checkerboard pattern, onto a sheet of paper. Then, measure the physical size of the squares.

2. Select a camera for calibration, such as your phone's camera or a webcam. Capture 20 images of the checkerboard for calibration, ensuring that you vary the shooting angles for each image. Please include a clear description of the camera you used.

   Note

   Typically, commercial cameras, like those in cell phones and laptops, come with an autofocus feature that can dynamically alter the camera's intrinsic parameters. To achieve a satisfactory calibration result, it's important to disable this feature when capturing images for your calibration dataset. If your default camera app on Android devices doesn't provide an option to disable autofocus, you can consider using the Camera FV-5 Lite app.

3. Follow the steps outlined in the provided tutorial[1] for conducting camera calibration. Record the intrinsic parameters you derive and briefly describe the process used to obtain them. Additionally, explain the physical meaning of each intrinsic parameter in your own words.

4. Utilize the camera parameters you acquired in the previous step to remove distortion from the 20 images you've taken, as well as a new image of a different object. Include all 21 pairs of original and undistorted images in your report and try to observe and describe the impact of this transformation.

[1]Camera Calibration and 3d Reconstruction in OpenCV

Part B — Object Detection (30%)

Given an image taken from a camera, apply the algorithms you learned in the binary machine vision section of Lecture 6 to develop a program to identify objects and their principal angles.

We provide you with a sample code `hw3/b/hw3_b_sample.py` that demonstrates how to read an image and use `findContours()` to detect objects. You can install the required libraries from the provided `hw3/b/requirements.txt` file.

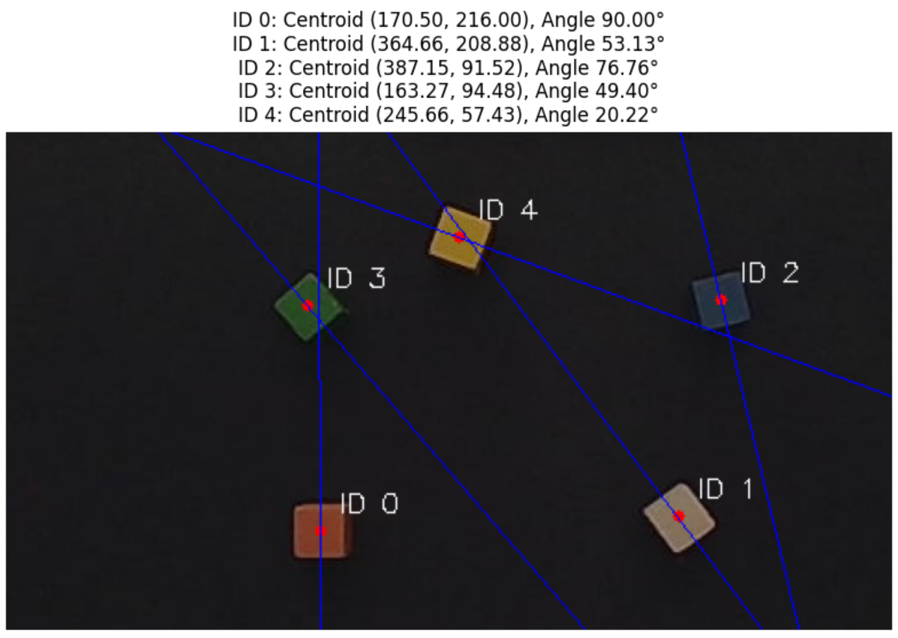
Input Image

We provide a sample input image `hw3/b/cubes.png`, which contains several objects of different colors on a flat surface for testing. Your program should take the input image from command line arguments such as:

```
python3 hw3_b.py cubes.png
```

Output Result

Your program should display the centroid and principal line for each object on the given image. Additionally, it should provide the centroid's coordinates and the principal angle. Your result should resemble the sample output shown in Figure 1.



```
ID 0: Centroid (170.50, 216.00), Angle 90.00°
ID 1: Centroid (364.66, 208.88), Angle 53.13°
ID 2: Centroid (387.15, 91.52), Angle 76.76°
ID 3: Centroid (163.27, 94.48), Angle 49.40°
ID 4: Centroid (245.66, 57.43), Angle 20.22°
```

Figure 1: Sample output of part B

Part C — Object Position Estimation (30%)

Now that you have obtained the camera intrinsic parameters from Part A and can detect objects in an image from Part B, you can estimate the 3D position of each object relative to the camera with the help of depth information.

Input Data

The camera matrix of the camera used to capture the images is provided below:

$$K = \begin{bmatrix} 613.57 & 0 & 286.54 \\ 0 & 613.54 & 251.36 \\ 0 & 0 & 1 \end{bmatrix}$$

The **aligned** raw depth data of `hw3/b/cubes.png` is provided in `hw3/c/depth.png`, which is a 16-bit grayscale png image where each pixel's value represents the depth from the camera to the object in millimeters. However, since the depth values are relative small, the depth image appears very dark, as shown in Figure 2. Thus, we also provide a fake color version of the depth image in `hw3/c/depth_fake.png` using color map `COLORMAP_JET` as showned in Figure 3 for your reference. In the fake color image, closer objects are represented with cooler colors (e.g., blue and green), while farther objects are represented with warmer colors (e.g., red and yellow).

Note

Typically, RGB image and depth data from a RGB-D camera are not perfectly aligned due to the different physical locations or different camera matrices of the RGB and depth sensors. And the process to project depth data to RGB image is refered to as **Depth-to-Color (D2C) Alignment**. However, in this assignment, we provide you with aligned RGB and depth images to simplify the problem.

Figure 2: Raw depth image

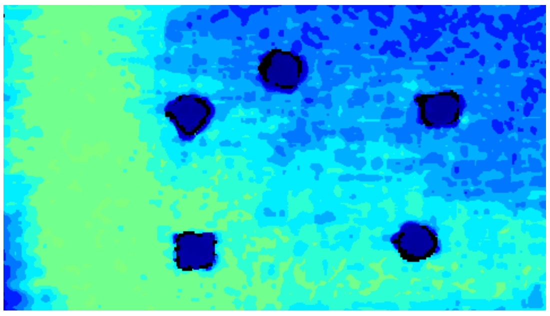


Figure 3: Depth image in fake color

Note

Since the depth data for some pixels is not available due to the limitations of the camera we used, which is the Gemini 335L available in the final project, those pixels will have a depth value of 0 in the depth image. The invalid pixels are colored black in Figure 3.

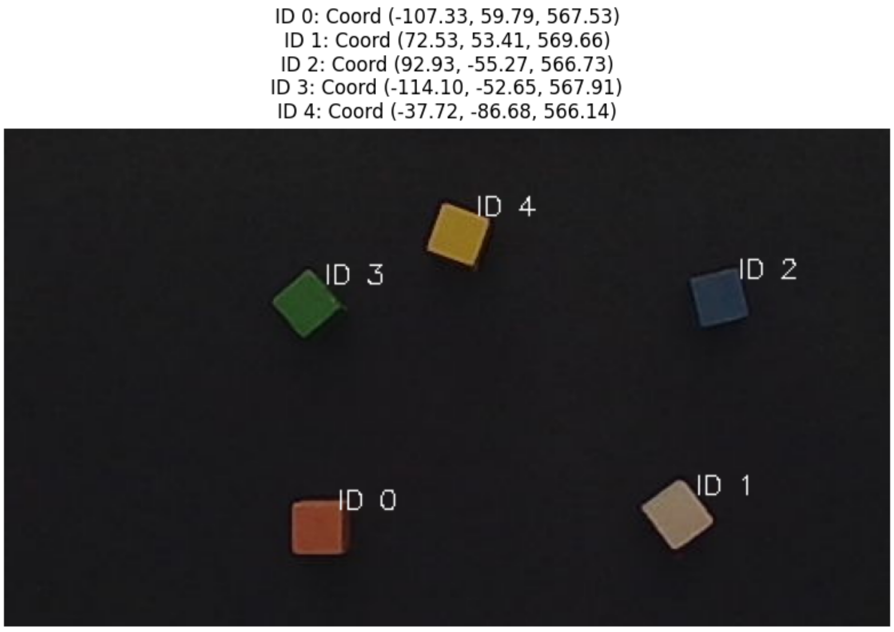
Your program should take the input image from command line arguments such as:

```
python3 hw3_c.py cubes.png depth.png
```

Output Result

Your program should estimate the 3D coordinate (X, Y, Z) of each object's centroid in the camera coordinate system. The origin of the camera coordinate system is at the camera's optical center, with the Z-axis pointing forward along the camera's viewing direction, the X-axis pointing to the right, and the Y-axis pointing downward.

Your result should resemble the sample output shown in Figure 4.



```
ID 0: Coord (-107.33, 59.79, 567.53)
ID 1: Coord (72.53, 53.41, 569.66)
ID 2: Coord (92.93, -55.27, 566.73)
ID 3: Coord (-114.10, -52.65, 567.91)
ID 4: Coord (-37.72, -86.68, 566.14)
```

Figure 4: Sample output of part C

Part D — Object Pose Estimation (10% + 10% Bonus)

Now that you know how to estimate the 3D position of each object relative to the camera, you can further estimate the full 6D pose (position and orientation) of each object in the scene.

In this part, you only need to provide a brief description of the method you would use to estimate the 6D pose of each object in the scene. You do **NOT** need to implement the program for this part. However, if you do implement the program and provide the results, you will receive an additional **10% bonus**. If you would like to use a robot arm in your final project, we encourage you to implement this part in order to identify the pose of the object for grasping.

Input Image

We provide a sample input image `hw3/d/cubes.png` shown in Figure 5. The image is taken at an angle to mimic typical viewing conditions for robot manipulation tasks. And the corresponding depth image is also provided as `hw3/d/depth.png`.

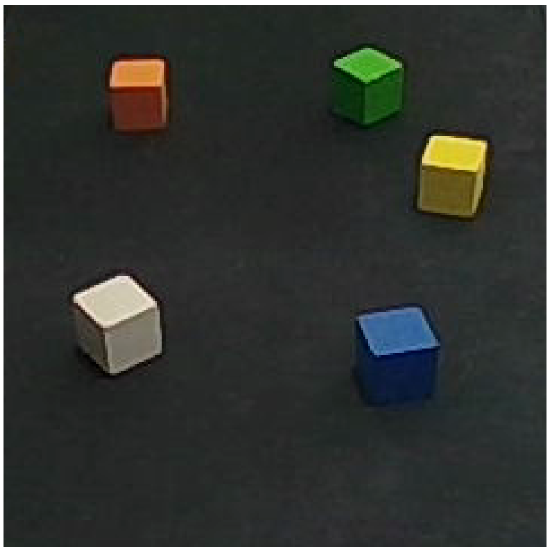


Figure 5: Image taken at an angle

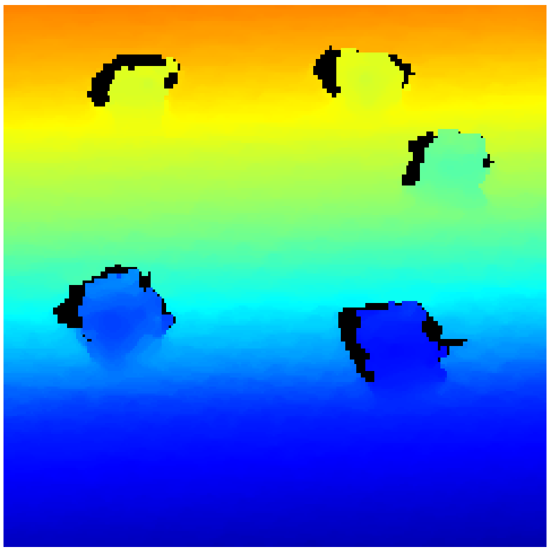


Figure 6: Depth image in fake color

Output Result

If you implement the program, your program should estimate the 6D pose (X, Y, Z, roll, pitch, yaw) of each object's centroid in the camera coordinate system.