Assignment-1
Advanced Computer Vision

-By Ruchir Namjoshi
002803974

A1. Calibration

To perform calibration and find the intrinsic parameters of the camera I performed the following steps-

1. Mounting the camera on a fixed height
2. Clicking 20 Images of a chessboard pattern by the camera using the OpenCV library in Python
3. Now I ran a for loop to get each image and find a chessboard pattern on it by the **cv2.findChessboardCorners()** function, and kept only those images where I found a pattern.
4. If the pattern was found I stored the corners for the pattern in both real-world space and image plane(3D and 2D coordinates)
5. Now I used the **cv2.calibrateCamera()** function to calibrate the camera and get all the intrinsic parameters.
6. To verify the calibration process I used an image of the chess board which was captured from a different fixed height with different angles. I undistorted the image by **cv2.undistort()** function in Python , calculated the length of one square in pixels and converted it into millimetres using the focal length I got during calibration and then compared it with the actual length of each square and computed the calibration error.

The calibration matrix after verifying is reported below.

```
Intrinsic Matrix:
[[2.02100114e+03 0.00000000e+00 9.28608282e+02]
 [0.00000000e+00 2.04959675e+03 4.87634022e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

Verification -

```
Square size in pixels: 124.19742584228516
Square size in mm: 26.849553228124034

Calibration error: 0.15044677187596633 mm
```

The Python file contains the whole code for getting and verifying the calibration matrix.

A2.

To calculate the Rotational matrix and angle of rotation, I performed the operation to capture 10 images with fixed height and different angles and then chose 1 image to calculate the rotational matrix by following the steps –

1. Found a chessboard pattern in the image and stored the corner points as the **2D points**
2. Calculated the real-world corner points as the **3D points** manually by using the dimension of the square and the pattern size.
3. Using these 2D and 3D points along with the intrinsic matrix and other parameters obtained during calibration, I calculated the rotational matrix and the translational vector using the equations of the image formation pipeline.
4. Using this rotational matrix I calculated the angle of rotations about each axis by using the math library in Python.

The intrinsic and extrinsic parameters, the Rotation matrix and the angles of rotation along each axis are given below.

```
Intrinsic Matrix:
[[2.02100114e+03 0.00000000e+00 9.28608282e+02]
 [0.00000000e+00 2.04959675e+03 4.87634022e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
----------------------------------------------------------------------
Extrinsic Rotation Matrix:
[[ 0.99900082 -0.04141265  0.01680367]
 [ 0.01733315  0.70557862  0.70841963]
 [-0.04119384 -0.70742053  0.70559143]]
----------------------------------------------------------------------
Extrinsic Translation Vector:
[[-41.57820113]
 [-68.12964785]
 [369.68556532]]
----------------------------------------------------------------------
Rotation angles (degrees) around x, y, z axes:
[-45.07416735   2.36090135   0.99401005]
```

The full experiment is performed in the Python file.

A3. Real-world dimensions of an object are calculated and marked over the frame itself-

Assumption – fixed height,
Intrinsic Parameters obtained above are used

Steps Explained:

1.**Grayscale Conversion:** I first converted the input frame to grayscale (`cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)`) because edge detection and contour analysis are typically more efficient and less complex when performed on single-channel images.

2. **Blurring**: Then I applied Gaussian blurring (`cv2.GaussianBlur(gray, (5, 5), 0)`) to the grayscale image to reduce noise. This helped in achieving more accurate edge detection by smoothing out irregularities.

3. **Edge Detection**: Then I used the `cv2.Canny` for edge detection. This algorithm identifies edges by looking for areas in the image with a high gradient of intensity, which are likely boundaries between objects.

4**. Finding Contours**: Then I applied `cv2.findContours` to detect the contours in the edged image.

5. **Contour Filtering**: I selected the largest one among the detected contours, (`sorted(contours, key=cv2.contourArea, reverse=True)[:1]`) assuming it corresponds to the object of interest.

6. **Approximating Contours**: Each contour is approximated (`cv2.approxPolyDP`) to a simpler shape with fewer vertices. If this approximated contour has four points, it's assumed to be a rectangle or a quadrilateral.

7**. Dimension Calculation**: The bounding rectangle of the approximated contour gives the width and height in pixels. These are then converted to **real-world** dimensions using the known distance to the object and the camera's intrinsic parameters (`mtx[0, 0]` for \(f_x\) and `mtx[1, 1]` for \(f_y\)).

The conversion formula used is

$$\text{Dimension in mm} = \frac{\text{Dimension in pixels} * \text{Distance } to \text{ object}}{\text{Focal length in pixels}}$$

This formula is derived from similar triangles formed by the object, its image on the camera sensor, and the focal length.

Reasons for Preference:

1. It can be applied to various objects and scenes, provided the object can be distinguished from the background and approximated as a quadrilateral.

2. By using the camera's intrinsic parameters, the method can provide accurate measurements without needing a complex setup or additional hardware.

A4. A basic web app using Flask and HTML is created to find the dimensions of an object in real time-

The application works in real-time by detecting an object in each frame making a boundary around it and showing its dimensions on the video frame itself.
The flask server runs on the app.py file. It needs camera_mtx.npy and dist_coeffs.npy files as input from previous steps.
The app runs on an index.html file.

Read-Me {Also included in the github repository}
This is a project to caliberate a camera and then use the obtained parameters to detect dimensions of a planer object.

This repsitory also contains a web app to diplay dimensions of an object in real time with video capturing based on flask and HTML.

The IPYNB file contains full python code.

Steps to Run the Web APP

1 The index.HTML files need to be in a directory named as templates and the app.py file should be in the same directory as template and other numpy files.
2. The camera should be connected to the device.
3. The app.py file should be executed to make a local host server (while executing app.py please check that the OpenCV VideoCapture function is using the desired camera to capture the video feed)
4. Click on the local host server, this will direct you to the front end of the application.
5. Click on the button "Start Measurement" and place your object in the view of the camera. (Ensure that the object is at a fixed distance{you can update this in the app.py file} from the camera and the object plane is parallel to the camera plane.)
6. The dimensions of the object would be visible below the real-time camera feed.

The video shows real-time working demonstration of the app.

Link for the GitHub repository - https://github.com/ruchirnamjoshi/module-1-CV-