CSc 8830: Computer Vision

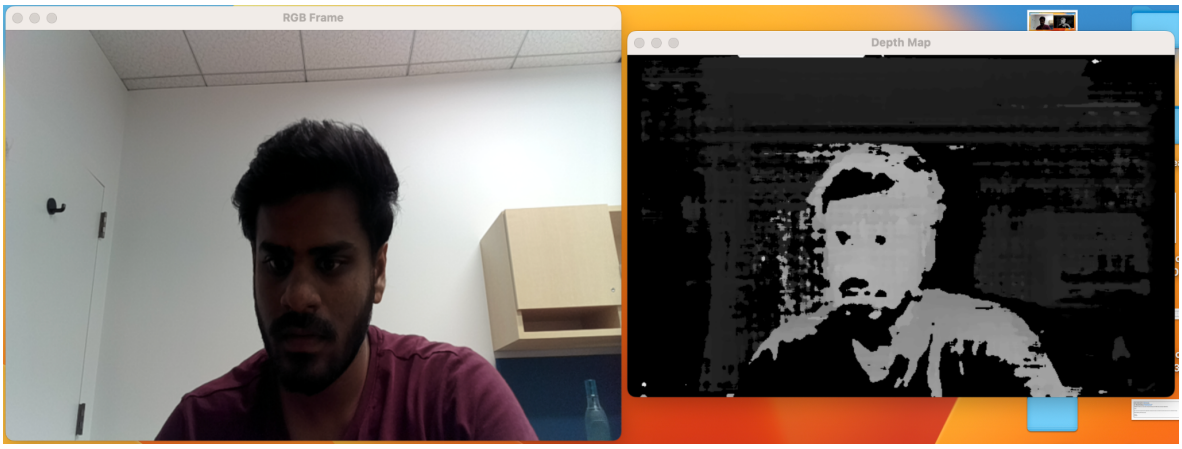
Assignment-1 Report

(4) With the OAK-D camera, set up your application to show a RGB stream from the mono camera and a depth map stream from the stereo camera simultaneously. Make a note of what is the maximum frame rate and resolution achievable?

Approach:

1. **Import Required Libraries:** Import the necessary libraries, including OpenCV, NumPy, and depthai.
2. **Set Parameters:** Define parameters such as the resolutions for the RGB and depth cameras, the dimensions for resizing the RGB frame, and options for stereo depth configuration.
3. **Create Pipeline:** Initialize the depthai pipeline.
4. **Configure RGB Camera:** Create a color camera node and configure its resolution. Create an output link for the RGB stream.
5. **Configure Mono Cameras:** Create left and right mono camera nodes and configure their resolutions and board sockets.
6. **Configure Stereo Depth**: Create a stereo depth node and configure options such as extended disparity, subpixel disparity, and left-right check. Link the left and right mono cameras to the stereo depth node.
7. **Create Output Queues:** Define output queues for the disparity and RGB streams.
8. **Initialize FPS Variables:** Initialize variables for calculating frames per second (FPS).
9. **Execute Pipeline:** Use the depthai device context manager to execute the pipeline.
10. **Retrieve Frames:** Continuously retrieve frames from the output queues for RGB and depth streams.
11. **Display Frames:** Display the RGB frame and depth map using OpenCV. Resize the RGB frame for better visualization. Normalize the depth map for better visualization.
12. **Calculate and Display FPS:** Calculate and print the FPS in the console.

Note: RGB & depth map are displayed using given stereo-vision camera
Maximum Frame Rate obtained is **95** Frames Per Second
Resolution observed is **1080** for RGB & **400** for depth map



1. Report the calibration matrix for the camera chosen and verify (using an example) the same.

Approach:

1. Capture Images for Calibration:
    a. Capture images using the capture_monochrome_images() and capture_color_images() functions. These functions capture images from the monochrome (left and right) and color (RGB) cameras, respectively.
2. Calibrate Cameras:
    a. Use the caliberate_camera() function to calibrate each camera separately. This function finds chessboard corners in the captured images, calibrates the cameras, and saves the calibration parameters such as the camera matrix and distortion coefficients.
3. Load Camera Parameters:
    a. Implement the load_camera_parameters() function to load the camera matrix and distortion coefficients from the saved files.
4. Correct Distortion:
    a. Implement the correct_distortion() function to correct distortion in an input image using the loaded camera parameters.
5. Calculate Calibration Error:
    a. Implement the calculate_error() function to calculate the calibration error by comparing the size of a chessboard square in the undistorted image with a known size. This step verifies the accuracy of the calibration.
6. Verify Calibration with Example:
    a. Load an example image.
    b. Load the camera parameters.
    c. Correct distortion in the image using the loaded parameters.
    d. Calculate the calibration error for the example image.

Calibration Matrix:

Assignment 1.ipynb | camera_matrix.txt

```
8.555304623164415716e+02 0.0000000000000000e+00 3.229116451292840679e+02
0.0000000000000000e+00 8.460577697253153246e+02 2.571479798894740725e+02
0.0000000000000000e+00 0.0000000000000000e+00 1.0000000000000000e+00
```

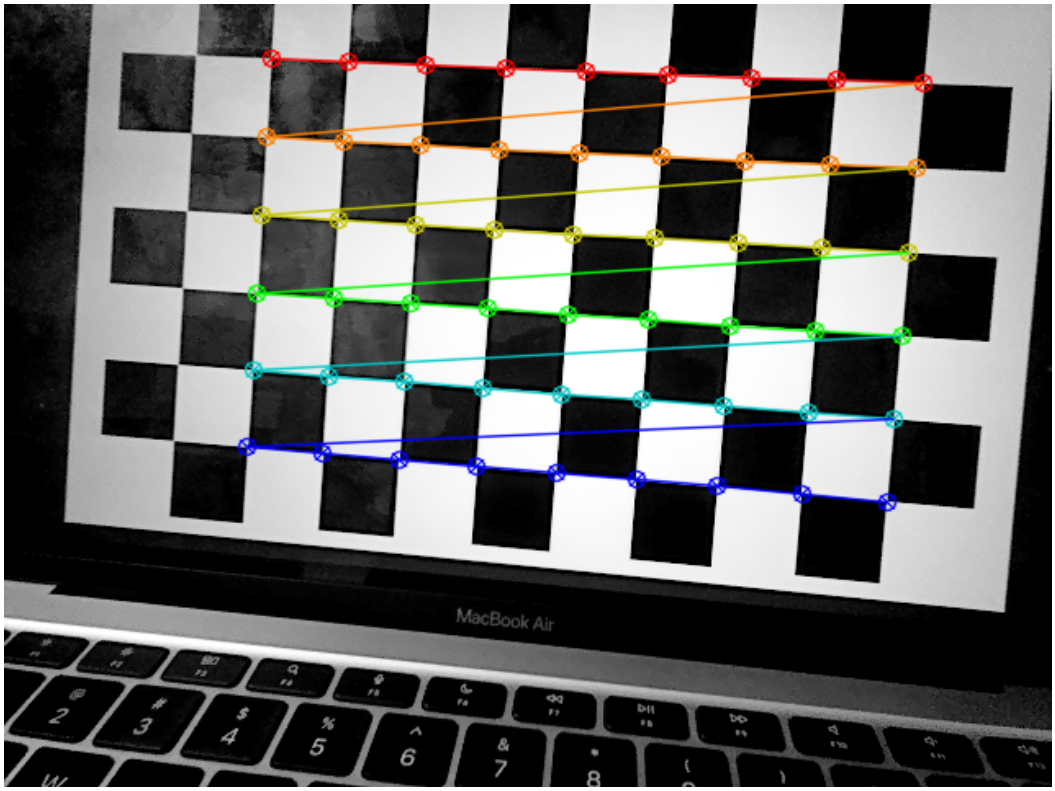
Above matrix is verified by finding the side of a square in the checker-board used for calibration and reported calibration error.

```
Size of Square in mm: 21.92162150557055
Calibration Error: 1.921621505570549 mm
```

2. Point the camera to a chessboard pattern or any known set of reference points that lie on the same plane. Capture a series of 10 images by changing the orientation of the camera in each iteration. Select any 1 image, and using the image formation pipeline equation, set up the linear equations in matrix form and solve for intrinsic and extrinsic parameters (extrinsic for that particular orientation). You will need to make measurements of the actual 3D world points, and mark pixel coordinates. Once you compute the Rotation matrix, you also need to compute the angles of rotation along each axis. Choose your order of rotation based on your experimentation setup.

Approach:

1. Capture Images with Varied Camera Orientations:
    a. Point the camera to a chessboard pattern or reference points lying on the same plane.
    b. Capture a series of 10 images while changing the orientation of the camera in each iteration.
2. Select an Image:
    a. Choose any one image from the captured series for further processing.
3. Identify Corresponding 3D World Points and Pixel Coordinates:
    a. Manually measure or determine the 3D world coordinates of reference points on the chessboard pattern.
    b. Correspondingly, mark the pixel coordinates of these points in the selected image.
4. Set up Linear Equations:
    a. Using the image formation pipeline equation, set up linear equations in matrix form.
5. Solve for Intrinsic and Extrinsic Parameters:
    a. Use the Linear Least Squares method to solve the linear equations for the intrinsic and extrinsic parameters.
    b. Extract the rotation matrix R from the intrinsic-extrinsic decomposition.
    c. Compute the angles of rotation along each axis from the rotation matrix.




```
Intrinsic Camera Matrix:
[[855.53046232   0.         322.91164513]
 [  0.         846.05776973 257.14797989]
 [  0.           0.           1.        ]]

Extrinsic Rotation Matrix:
[[ 0.91573981  0.11456659  0.38509104]
 [ 0.09173585  0.87354035 -0.47802907]
 [-0.39115872  0.47307691  0.7894258 ]]

Extrinsic Translation Vector:
[[ -99.27374148]
 [-130.57869143]
 [ 513.37507446]]

Rotation Angles across X, Y, Z axes (degrees):
[30.93286966 23.02661748  5.72062055]
```

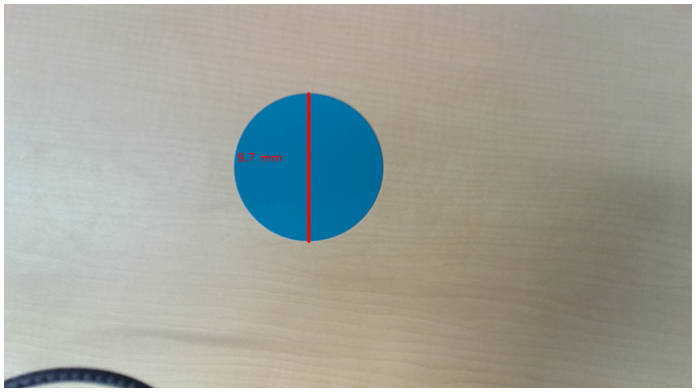
3. Write a script to find the real world dimensions (e.g. diameter of a ball, side length of a cube) of an object using perspective projection equations. Validate using an experiment where you image an object using your camera from a specific distance (choose any distance but ensure you are able to measure it accurately) between the object and camera.

Approach:

1. Define Conversion Function: Implement a function to convert millimeters to inches. This conversion may be necessary depending on the units used in the experiment.
2. Calculate Object Distance: Define a function (calculate_object_distance()) that takes the captured image, bounding box coordinates of the object, camera focal lengths (fx, fy), and the distance (Z) between the camera and the object as input.
    - Use perspective projection equations to calculate the real-world coordinates of the object's corners.
    - Compute the distance between two corners in the real world.
    - Convert the distance to the desired units (e.g., millimeters or inches).
    - Display the calculated dimensions on the image.
3. Image Processing and Validation: Load the captured image and call the calculate_object_distance() function with the necessary parameters.
    - Provide the bounding box coordinates, focal lengths, and distance between the object and camera.
    - Print or display the calculated dimensions of the object.

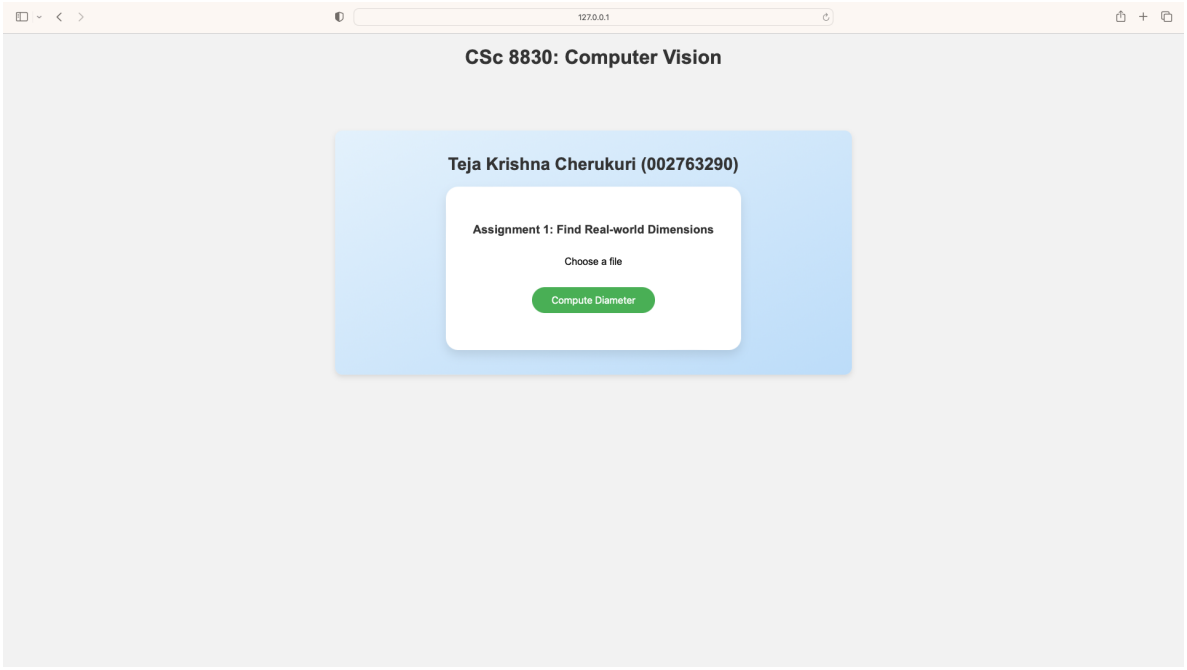
To solve this, I have chosen a circul;ar disk and found its real world dimensions.

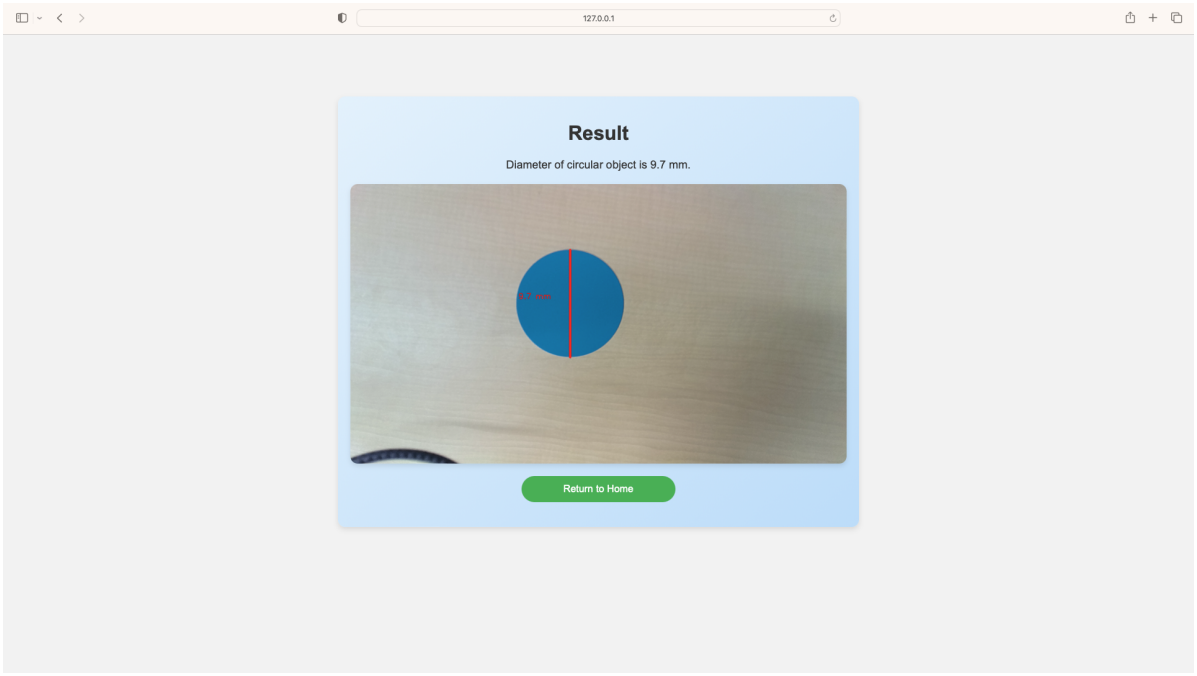
```
Real World Co-ordinates:
        231.10878507752008
        125.51172762920292
        343.9442507330152
        344.40853676993856

Diameter of circular object is: 9.7 mm
```



Actual diameter is 9.5 cm, but predicted as 9.7 cm, hence the error observed is 0.2 cm.

4. Write an application – must run as a Web application on a browser and be OS agnostic – that implements the solution for problem (3) [An application that can compute real-world dimensions of an object in view]. Make justifiable assumptions (e.g. points of interest on the object can be found by clicking on the view or touching on the screen).





Repo: https://github.com/TejaCherukuri/CSC8830_ComputerVision/tree/main/Assignment-1