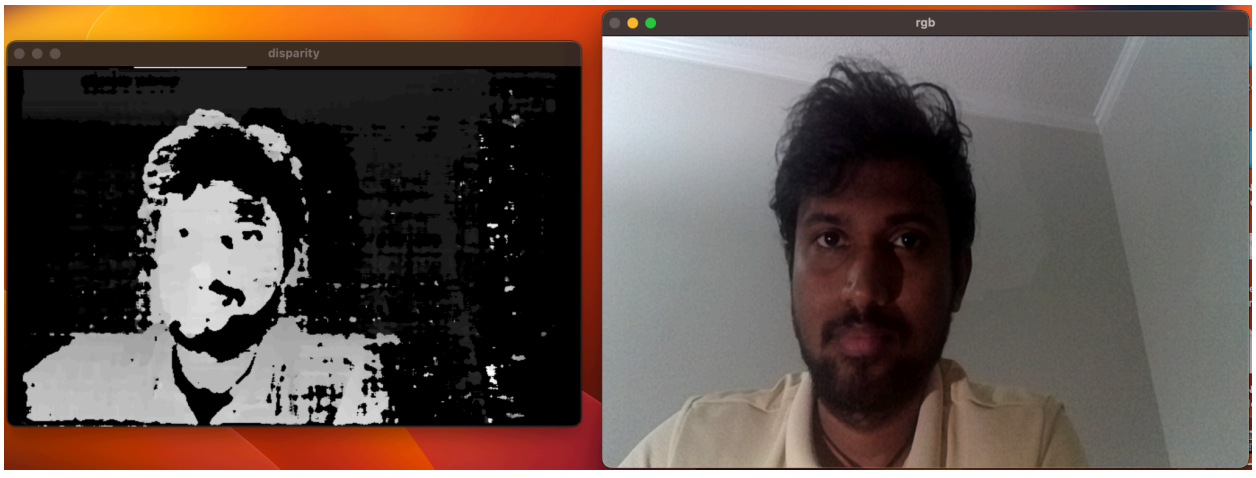
CSc 8830: CV Assignment-1 Solutions Report

(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?

Steps Involved:

1. **Import Libraries:** Import the necessary libraries such as time, cv2, numpy, and depthai.
2. **Define Parameters:** Set the parameters such as the resolution for the RGB camera (RGB_RESOLUTION), resolution for the mono cameras (DEPTH_RESOLUTION), and the desired dimensions for resizing the RGB frame (DIM).
3. **Create Pipeline:** Create a pipeline object using dai.Pipeline().
4. **Configure Cameras:** Configure the RGB camera and the left and right mono cameras with their respective resolutions and board sockets.
5. **Configure Stereo Depth:** Configure the stereo depth node with parameters such as extended disparity range, sub-pixel disparity refinement, and left-right consistency check.
6. **Create Output Queues:** Create output queues for the disparity frames and RGB frames.
7. **Main Loop:** Enter the main loop where you continuously retrieve frames from the output queues.
8. **Display Frames:** Display the RGB frame and the depth map frame using OpenCV's cv2.imshow() function.
9. **Calculate FPS:** Calculate the frames per second (FPS) and print it to the console.
10. **Keyboard Interrupt Handling:** Check for a key press, if 'q' is pressed, close all OpenCV windows and break the loop.



Maximum Frame Rate:92 FPS
Resolution: 1080 for RGB & 400 for depth map

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

   Steps Involved:

   1. **Capture Images:** Utilize the provided captureImages() and captureColorImages() functions to capture images from the desired camera sources (right, left, or RGB) of the OAK-D camera.
   2. **Calibrate Cameras:** Employ the calibrate() function to find corners, calibrate, and store the camera matrix and distortion vector for each camera source.
   3. **Load Camera Data:** Implement the load_camera_data() function to load the camera matrix and distortion vector from the saved files.
   4. **Undistort Image:** Use the undistort_image() function to undistort an example image using the loaded camera matrix and distortion vector.
   5. **Calculate Calibration Error:** Employ the calculate_calibration_error() function to calculate the calibration error using the undistorted image and known parameters such as the size of the chessboard square and the distance of the object from the camera.

   Calibration (Intrinsic) Matrix is:

```
1.10337648962356047e+03 0.00000000000000000e+00 2.24967356582175028e+02
0.00000000000000000e+00 1.10188906337563321e+03 2.85431664175799198e+02
0.00000000000000000e+00 0.00000000000000000e+00 1.00000000000000000e+00
```

   Verification: (Verified edge of a square in chessboard pattern):

```
Size of Square in mm: 32.32487685307927
Calibration Error: 2.3248768530792674 mm
```

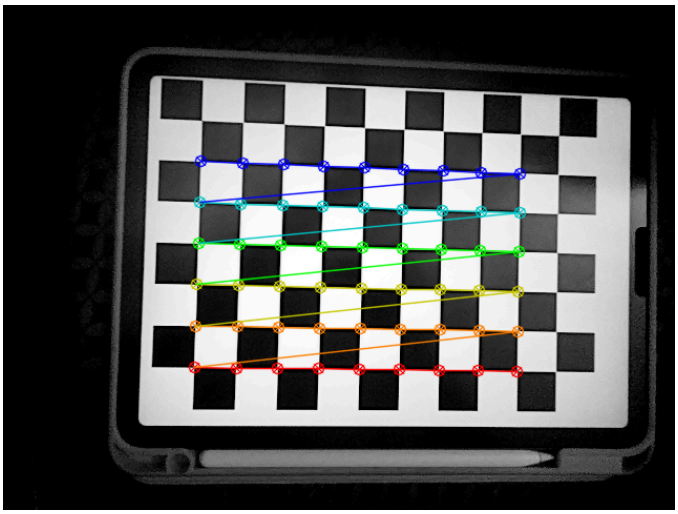
2. **Point the camera to a chessboard pattern or any known set of reference points that lie on the same plane. Capture a series of 10 images by changing the orientation of the camera in each iteration. Select any 1 image, and using the image formation pipeline equation, set up the linear equations in 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.**

   Steps Involved:

   1. **Capture Images:** Utilize the provided captureImages() and captureColorImages() functions to capture images from the desired camera sources (right, left, or RGB)

of the OAK-D camera.

2. **Calibrate Cameras:** Employ the calibrate() function to find corners, calibrate, and store the camera matrix and distortion vector for each camera source.
3. **Load Camera Data:** Implement the load_camera_data() function to load the camera matrix and distortion vector from the saved files.
4. **Undistort Image:** Use the undistort_image() function to undistort an example image using the loaded camera matrix and distortion vector.
5. **Compute Camera Parameters:** Utilize OpenCV functions to find the chessboard corners in the undistorted image, extract image points for PnP (Perspective-n-Point) calculation, define 3D real-world points, solve PnP to obtain rotation and translation vectors, and convert the rotation matrix to Euler angles.
6. **Display Results:** Print the intrinsic camera matrix, extrinsic rotation matrix, extrinsic translation vector, and rotation angles across X, Y, Z axes in degrees.



```
Intrinsic Camera Matrix:
[[210.45781727   0.         315.89660879]
 [  0.         210.13076857 238.12538675]
 [  0.           0.           1.        ]]

Extrinsic Rotation Matrix:
[[-0.99901378 -0.04423422 -0.00384739]
 [ 0.04406837 -0.9983854   0.0358409 ]
 [-0.00542657  0.03563601  0.9993501 ]]

Extrinsic Translation Vector:
[[130.86988591]
 [ 84.19519903]
 [169.59623503]]

Rotation Angles across X, Y, Z axes (degrees):
[  2.04225521   0.31092111 177.47421327]
```

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.**

Steps Involved:

1. **Capture Image:** Capture an image of the object using the camera from a specific distance. Ensure that you accurately measure this distance.
2. **Extract Object Region:** Utilize object detection or manual bounding box annotation to obtain the bounding box coordinates of the object in the image.
3. **Calculate Object Distance:** Use the calculate_object_distance() function to calculate the real-world dimensions of the object based on its apparent size in the image and the known distance between the object and the camera.
4. **Convert to Real-World Units:** Convert the calculated dimensions to the desired real-world units (e.g., millimeters) using the provided conversion function convert_milli_to_inch().
5. **Display and Validate:** Display the calculated dimensions on the image and validate the result visually.

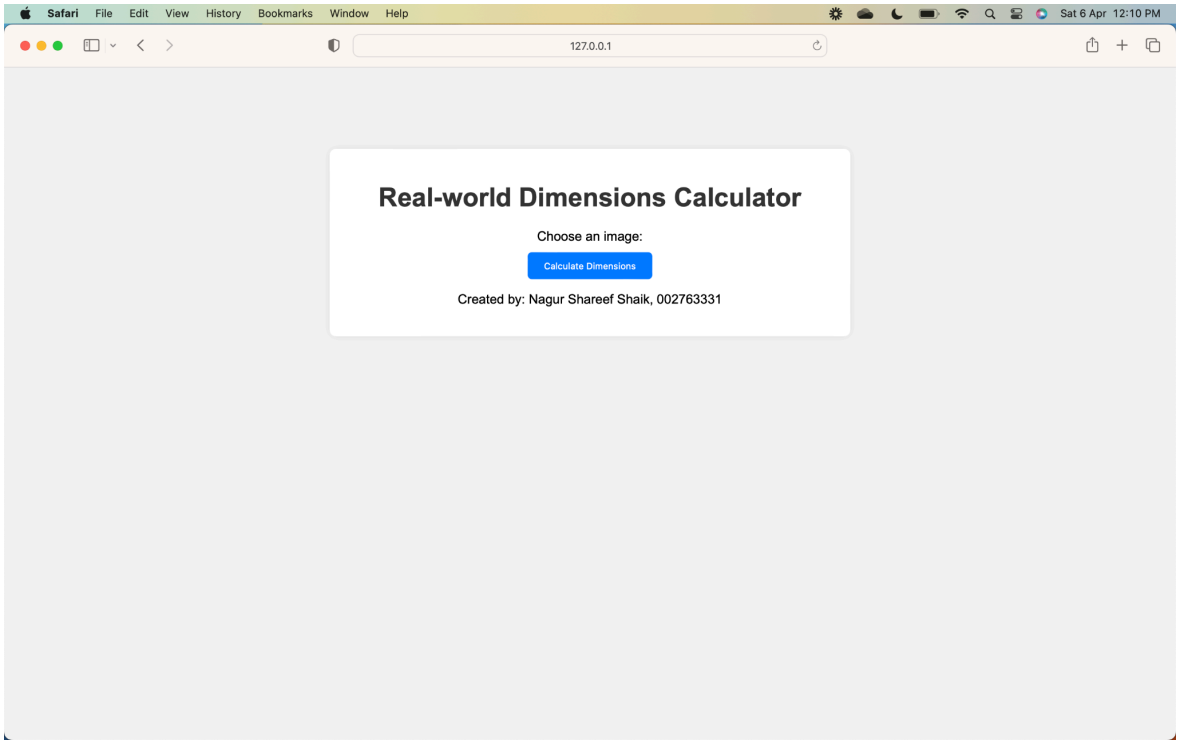
We choose to use circular objects to find real world dimensions. Real diameter is 105.0 mm — 0.17 mm is the error.

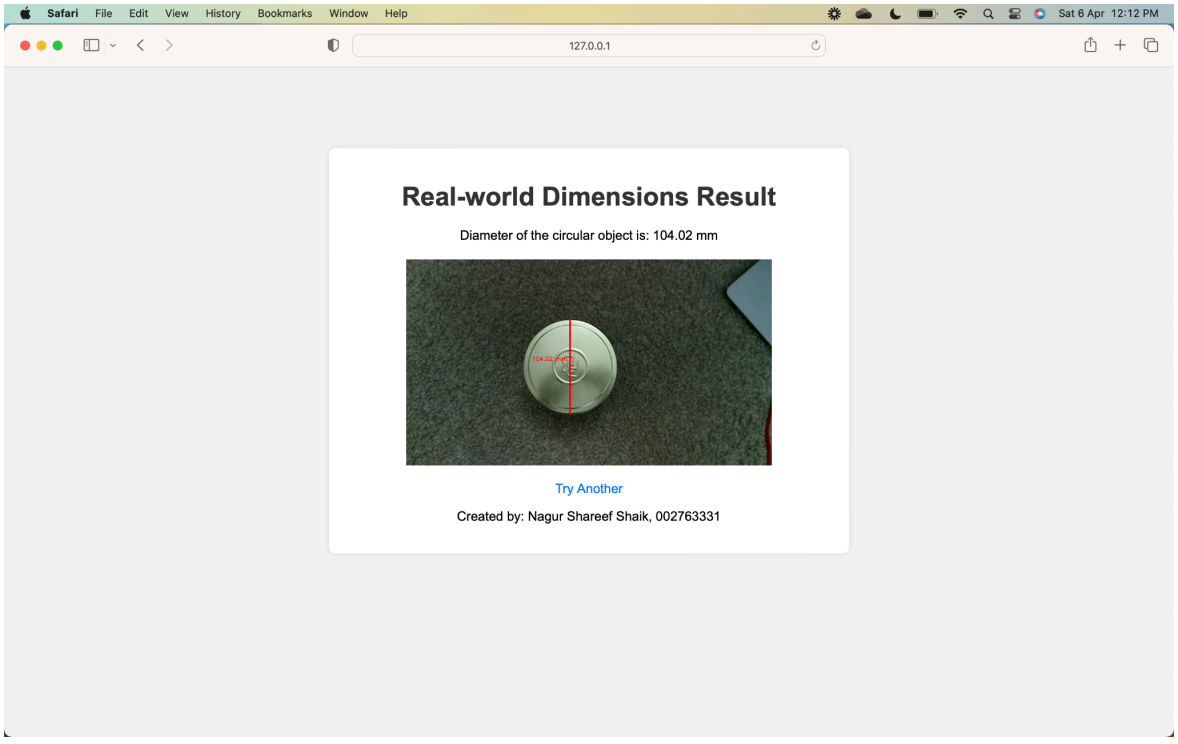
```
Camera Intrinsic Matrix:  [[210.45781726943, 0.0, 315.89660878657725], [0.0, 210.13076857289957, 238.1253867521593], [0.0, 0.0, 1.0]]
fx: 210.45781726943, fy: 210.13076857289957, Z: 300
Center (x, y): 877, 575; Width (w): 486; Height (h): 486

Real World Co-ordinates:
        1250.1317528309107
        820.917380027359
        1942.9071597588722
        1945.9311112648525

Diameter of circular object is: 104.03 mm
```

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).**

   Steps Involved:

   1. **Setup Flask Project:** Create a new directory for your Flask project and set up a virtual environment within it. Install Flask using pip.
   2. **Create Flask Application:** Create a Python file (e.g., app.py) to serve as the main Flask application. Import necessary modules including Flask, OpenCV (for image processing), and NumPy.
   3. **Set up Routes:** Define routes for different functionalities of your application. For example, you might have a route for the home page, a route for uploading an image, and a route for processing the uploaded image.
   4. **Create HTML Templates:** Create HTML templates for each route using Jinja2 templating. These templates will define the structure of your web pages.
   5. **Implement Image Upload:** Implement functionality to allow users to upload an image. This can be done using an HTML form with a file input field.
   6. **Process Uploaded Image:** Implement functionality to process the uploaded image. This includes extracting points of interest on the object, such as clicking on specific regions or using image processing techniques to detect features.
   7. **Compute Real-World Dimensions:** Implement the logic to compute the real-world dimensions of the object based on the points of interest identified in the previous step. This involves applying perspective projection equations.
   8. **Display Results:** Display the computed dimensions on the web page, along with the uploaded image. You can use JavaScript to update the DOM dynamically with the computed values.
   9. **Styling and User Interface:** Add CSS styling to improve the appearance of your web pages and provide a better user experience. Ensure the user interface is intuitive and easy to use.





Github Link:
https://github.com/ShaikNagurShareef/CSc8830-Computer-Vision/tree/main/Assignment-1