**CV Assignment 1 : Vamsi Krishna Reddy Devireddy (002793151)**

**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?**

The project involved the setup and calibration of the OAK-D camera to capture RGB and depth map streams simultaneously. This required the use of essential libraries such as `depthai` and `opencv-python`.

**Setup and Pipeline Creation**

The initial step encompassed setting up the camera pipeline. Nodes for the color camera, left and right mono cameras, and stereo depth were created to facilitate data acquisition.

**Stream Capture and Processing**

Using the established pipeline, RGB and depth map streams were captured concurrently. Output queues were configured to retrieve frames from these streams. Frames were processed within a loop, involving depth map conversion for enhanced visualization and potential resizing of the RGB frame.

**Visualization and Monitoring**

The visual output was facilitated through OpenCV, showcasing both RGB and depth map frames. Additionally, real-time frames per second (FPS) calculations were displayed for monitoring purposes.

**Camera Calibration**

To ensure accurate measurements, images were captured from various sources including left, right, and RGB cameras. These images were pivotal for camera calibration, involving chessboard corner detection, refinement, and the derivation of intrinsic parameters such as the camera matrix and distortion coefficients.

**Coordinate Conversion and Distance Measurement**

Leveraging the intrinsic parameters, particularly focal length, pixel coordinates from the RGB image were converted to real-world coordinates. Subsequently, distances between points in the image were accurately measured using Euclidean calculations and real-world unit conversions based on the camera calibration data.

**Results and Conclusion**

The culmination of these processes resulted in the precise measurement of distances within the captured images. This information, along with relevant annotations, was displayed on the images to provide a comprehensive overview of the OAK-D camera setup and its functionalities.

A person smiling and a person in headphones

Description automatically generated  
  
**1. Report the calibration matrix for the camera chosen and verify (using an example) the same.**  
**Camera Calibration Report:**

Firstly, I embarked on calibrating the chosen camera by capturing a series of 10 images of a chessboard pattern from varying angles. Utilizing OpenCV's functionality, I meticulously detected and refined the corners of the chessboard in each image. This process was instrumental in computing the camera matrix and distortion coefficients, pivotal for rectifying distortions within the captured images. These calibration parameters, encompassing details like the focal length and distortion coefficients, were methodically stored in a designated file named `camera\_matrix.txt`.

To validate the efficacy of the calibration, I chose one of the calibration images and applied the calculated camera matrix to undistort the image. Subsequently, I conducted a visual comparison between the undistorted image and the original to affirm the precision of the calibration process.

A black and white text

Description automatically generated with medium confidence  
  
**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.**  
**Camera Orientation and Parameter Calculation:**

Next, I positioned the camera towards a chessboard pattern and proceeded to capture a sequence of 10 images while systematically altering the camera's orientation with each iteration. From this captured set, I singled out a representative image for in-depth analysis.

Within this selected image, I meticulously identified known points residing on the chessboard and meticulously measured their pixel coordinates. Concurrently, I conducted precise measurements of the actual 3D coordinates of these points in the physical world. Leveraging the image formation pipeline equation and the amalgamation of known pixel and world coordinates, I adeptly formulated a series of linear equations in matrix form.

The resolution of these equations facilitated the computation of intrinsic parameters, such as the focal length and principal point, alongside the extrinsic parameters including rotation and translation matrices specific to the chosen image orientation. Additionally, I adeptly derived the rotation angles along each axis (roll, pitch, and yaw) from the established rotation matrix, employing a rotation order tailored to my experimentation setup.

This comprehensive approach, encompassing calibration validation and precise parameter calculation, was pivotal in enhancing our understanding of the camera's behavior and ensuring precise measurements for future imaging endeavors.  
  
A screenshot of a computer

Description automatically generated

A black and white text

Description automatically generated with medium confidence

A black and grey rectangular object

Description automatically generated

A black and white logo

Description automatically generated with medium confidence

A black and white text

Description automatically generated with medium confidence

**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.**

I started by taking a picture of the object with my camera, making sure to note the exact distance between the camera and the object. This step was crucial for accurate calculations later.

After capturing the image, I used object detection techniques to outline the object in the picture. This helped in pinpointing the area of interest for dimension calculation.

With the object region extracted, I wrote a function called `calculate\_object\_distance()` to figure out its real-world dimensions. This involved using the object's apparent size in the image and the known distance from step one.

Once I had the dimensions in pixels, I used a conversion function called `convert\_milli\_to\_inch()` to convert them into millimeters or inches, depending on my preference.

To validate the accuracy of my calculations, I displayed the calculated dimensions directly on the image. I also compared these dimensions with the actual measurements of circular objects with known diameters to ensure precision.

Through this process, I developed a script that accurately determines real-world dimensions using perspective projection equations. The validation experiments were crucial in confirming the reliability of the calculations.

ACTUAL: 5.8 cms

PREDICTED: 5.08cms

Error: 0.72cms

A screenshot of a computer

Description automatically generated

**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).**

I started by organizing my Flask project in a new directory, setting up a virtual environment, and installing Flask using pip for managing dependencies.

In my Flask application file, named app.py, I imported essential modules like Flask, OpenCV for image processing, and NumPy for numerical operations to create the core functionality.

To handle different tasks, I defined routes within Flask, such as a home page route, an image upload route, and a processing route for uploaded images.

For the user interface, I designed HTML templates using Jinja2 templating, ensuring each template matched a specific route and laid out the web pages' structure.

To enable image upload, I integrated functionality using an HTML form with a file input field, allowing users to submit images for processing.

Upon image upload, I implemented logic to process the images, such as extracting object points of interest either through user interaction or image processing techniques.

Utilizing the identified points of interest, I applied perspective projection equations to accurately compute the real-world dimensions of the objects.

I dynamically displayed the computed dimensions alongside the uploaded image on the web page using JavaScript to update the DOM in real-time.

To enhance the user experience, I added CSS styling to the HTML templates, improving the web pages' appearance and ensuring a user-friendly interface.

By following these steps, I successfully developed a Flask application that empowers to upload images, compute real-world dimensions, and view results interactively on the web.

A screenshot of a computer

Description automatically generated

A screenshot of a circle

Description automatically generated

[**https://github.com/krishnadv97/CV-Assignments**](https://github.com/krishnadv97/CV-Assignments)