Computer Vision and Pattern Analysis Final Project Report

Ziwen Guo (1W22CF84-9)

Prof. Hiroshi Ishikawa

10^{th} August, 2023

1 Introductin

Stereovision is a critical aspect of computer vision that empowers various applications, from robotics to autonomous driving. It involves the utilization of two or more cameras positioned at different angles to capture the same scene, a principle analogous to human binocular vision. Through comparing these multiple images, algorithms can compute disparities and generate depth information about the objects within the scene. This depth estimation process has enabled numerous technological advancements and has significant implications in fields such as augmented reality, navigation, and 3D modeling. This report will delve into the fundamental principles of stereovision, the methodologies utilized in depth estimation, my specific implementation, its limitations, and future potentials.

2 Advantages and Disadvantages of Stereovision

Stereovision systems offer several advantages over singular vision systems. Unlike monocular systems, which rely on a single camera, stereovision employs two or more cameras to capture the depth information of a scene. This configuration allows for a more precise 3D representation of the environment. While monocular systems require additional data such as object size, prior scene information, or complicated neural network and trainings to estimate depth, stereovision systems inherently extract depth information through the comparison of the disparity between the captured images. This capability enhances the performance in tasks like navigation, object recognition, and collision avoidance, especially in real-time applications where accurate depth perception is critical.

Despite these advantages, stereovision also has some limitations. The complexity of implementing multiple cameras can lead to increased costs and potential alignment issues.

Calibration and synchronization between the cameras are crucial for accurate depth estimation and may require extensive tuning. Additionally, situations with low texture or repetitive patterns can cause ambiguity in matching points between the cameras, leading to errors in depth estimation. Furthermore, the computational cost of a stereo vision system is higher due to the necessity of processing two images and performing more complex computations, made worse by the fact that such system will often be deployed on less powerful machines.

3 Camera Setup

The choice of equipment encompasses a pair of Logitech's HD Pro C920 webcams. These webcams were selected on account of several advantageous characteristics. Notably, they are relatively accessible, being among the most prevalent webcam models with appreciable performance metrics. Additionally, their affiliation with a distinguished company such as Logitech ensures well-documented specifications, which can be directly leveraged in the project.

The physical configuration of the cameras includes a baseline, defined as the inter-lens distance, set at 9 centimeters, and an alignment leveled meticulously. However, it is important to note that this setup is not without its challenges. Specifically, the adjustability inherent in the camera positions complicates the task of achieving a consistent pointing angle between the two lenses without introducing horizontal tilt.

It is precisely these geometric intricacies that underscore the vital role of the calibration and rectification algorithms within the project's overall framework. These computational procedures are not mere mathematical abstractions but serve as tangible tools designed to mitigate the aforementioned inaccuracies. The reliance on these algorithms exemplifies an integration of theoretical principles and practical implementation, aimed at achieving the highest level of accuracy possible within the constraints of the hardware setup. The choice of cameras, combined with the careful alignment and calibration strategies, illustrates a comprehensive approach to the complex task of stereovision.

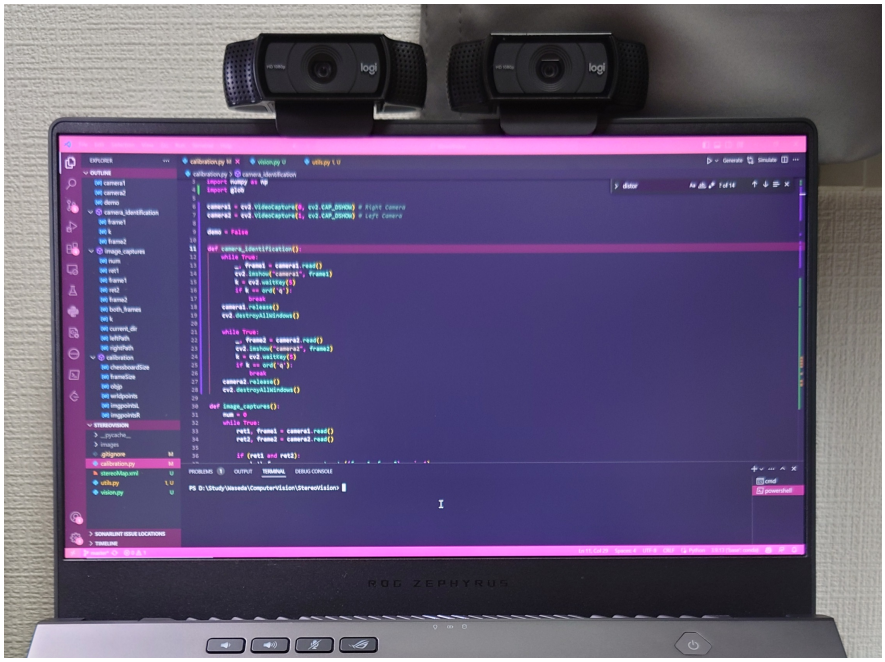

Figure 1: The Camera Setup

4 Project Implementations

4.1 Camera Initialization and Image Capturing

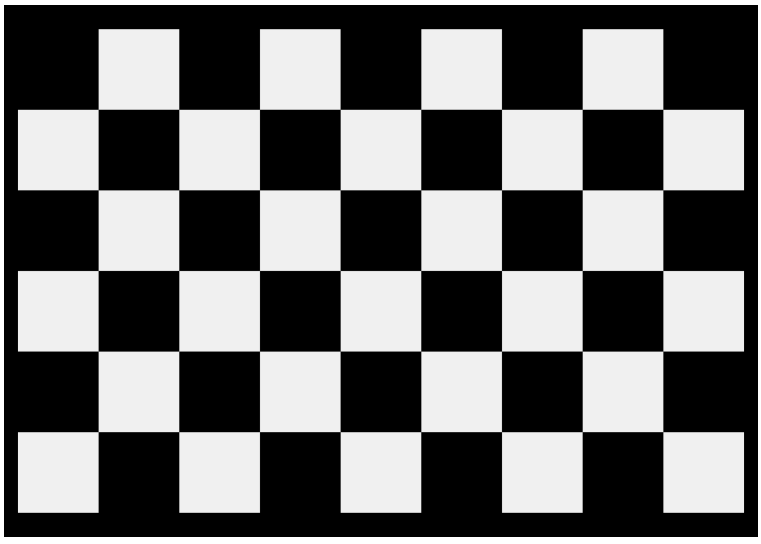

Figure 2: Chessboard example

The calibration process in my implementation comprises several steps, utilizing key functions available within the OpenCV library. Initially, the calibration commences with the acquisition of a set of reference images. These images capture a standard 9x6 chessboard grid, with an inner corner dimension of 8x5, as seen through both cameras, designated as

left and right. Subsequent to this acquisition, the images undergo a transformation into grey scale representations.

```
camera1 = cv2.VideoCapture(0, cv2.CAP_DSHOW)
camera2 = cv2.VideoCapture(1, cv2.CAP_DSHOW)

def camera_identification():
‘   # Code to identify the correct camera position and store them
    # assign the global camera variables the correct camera index
    ...

def image_captures():
    num = 0
    while True:
        ret1, frame1 = camera1.read()
        ret2, frame2 = camera2.read()

        if (ret1 and ret2):
            both_frames = np.concatenate((frame1, frame2), axis=1)
            cv2.imshow('Capturing Stage', both_frames)
        else:
            print('Error reading cameras')
            break

        k = cv2.waitKey(5)
        if k == ord('q'):
            break
        elif k  == ord('s'):
            # get the current directory
            current_dir = os.getcwd()

            # build the paths for the left and right images
            os.makedirs('images/left', exist_ok=True)
            os.makedirs('images/right', exist_ok=True)
            leftPath = os.path.join(current_dir,
                \ 'images', 'left', 'imageL' + str(num) + '.png')
            rightPath = os.path.join(current_dir,
                \ 'images', 'right', 'imageR' + str(num) + '.png')

            # save the images
```

```
cv2.imwrite(leftPath, frame1)
cv2.imwrite(rightPath, frame2)

print ('Images saved')
num += 1
```

Listing 1: Initialization and Reference Image Capture

Following this transformation, the process invokes a specialized chessboard corner matching function provided by OpenCV. This function is critical in detecting the corners and edges of the chessboard pattern. It is noteworthy that these detected corners are not mere geometric features but serve as vial image points within the calibration procedure. These image points hold a one-to-one correspondence to the real-world object points, which are defined explicitly concerning the layout of the chessboard.

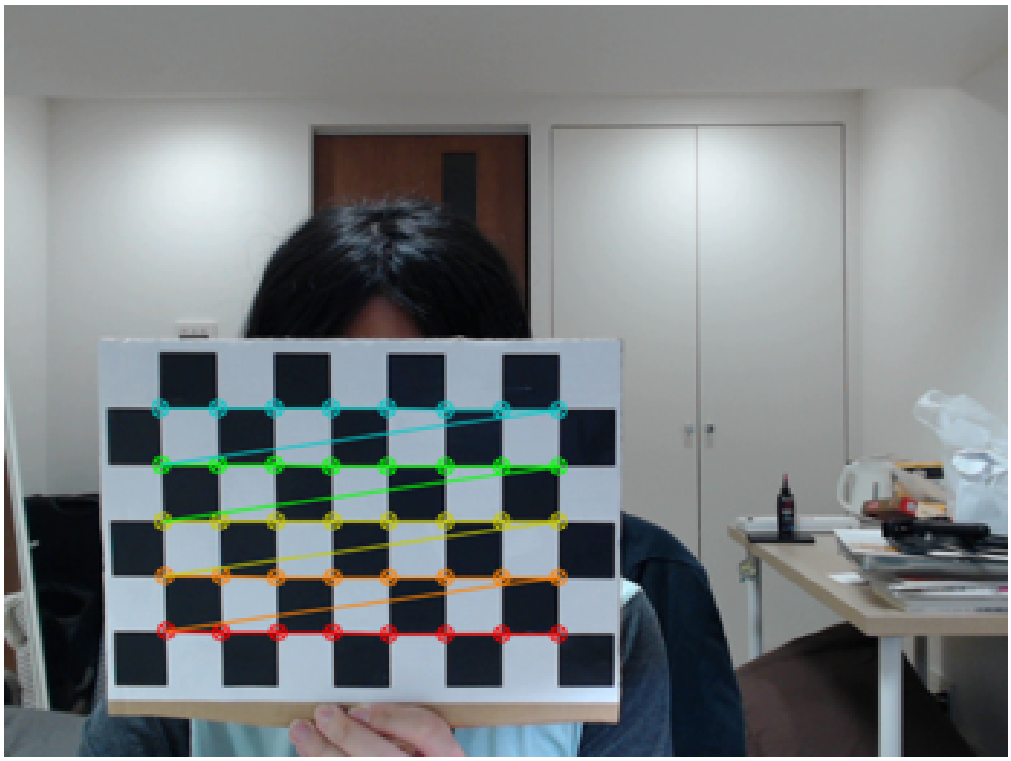

Figure 3: Found Corners

```
def calibration():
    chessboardSize = (8,5)
    frameSize = (640,480)

    objp = np.zeros((chessboardSize[0] *
        \ chessboardSize[1], 3), np.float32)
    objp[:,:2] = np.mgrid[0:chessboardSize[0],
        \ 0:chessboardSize[1]].T.reshape(-1,2)

    wrldpoints = []
    imgpointsL = []
    imgpointsR = []

    current_dir = os.getcwd()
    leftPath = os.path.join(current_dir, 'images', 'left', '*.png')
    rightPath = os.path.join(current_dir, 'images', 'right', '*.png')
    imagesLeft = glob.glob(leftPath)
    imagesRight = glob.glob(rightPath)

    ctr = (cv2.TERM_CRITERIA_EPS +
        \ cv2.TERM_CRITERIA_MAX_ITER, 50, 0.0005)

    num = 0
    for imgLeft, imgRight in zip(imagesLeft, imagesRight):

        imgL = cv2.imread(imgLeft)
        imgR = cv2.imread(imgRight)
        grayL = cv2.cvtColor(imgL, cv2.COLOR_BGR2GRAY)
        grayR = cv2.cvtColor(imgR, cv2.COLOR_BGR2GRAY)

        # Find the chess board corners
        retL, cornersL =
            \ cv2.findChessboardCorners(grayL, chessboardSize, None)
        retR, cornersR =
            \ cv2.findChessboardCorners(grayR, chessboardSize, None)

        # refining the image points with more searaching steps
        if (retL and retR):
```

```
        wrldpoints.append(objp)

        cornersL =
            \ cv2.cornerSubPix(grayL, cornersL,
            \ (15, 15), (-1, -1), ctr)
        imgpointsL.append(cornersL)

        cornersR =
            \ cv2.cornerSubPix(grayR, cornersR,
            \ (15, 15), (-1, -1), ctr)
        imgpointsR.append(cornersR)


        # Draw and display the corners
        ...
```

Listing 2: Checkess Board Pattern Corner Matching

The calibration process continues with the independent calibration of each camera, carried out before the subsequent stereo calibration phase. Through this method, rotation and translation matrices are meticulously derived. These matrices play a pivotal role in defining the spatial relationship between the two cameras, providing the foundational geometry upon which further stereovision analysis relies.

```
_, cameraMatrixL, distortionL, _, _ =
    \ cv2.calibrateCamera(wrldpoints,
    \ imgpointsL, frameSize, None, None)
heightL, widthL, _ = imgL.shape
newCameraMatrixL, _ =
    \ cv2.getOptimalNewCameraMatrix(cameraMatrixL,
    \ distortionL, (widthL, heightL), 1, (widthL, heightL))

_, cameraMatrixR, distortionR, _, _ =
    \ cv2.calibrateCamera(wrldpoints,
    \ imgpointsR, frameSize, None, None)
heightR, widthR, _ = imgR.shape
newCameraMatrixR, _ =
    \ cv2.getOptimalNewCameraMatrix(cameraMatrixR,
    \ distortionR, (widthR, heightR), 1, (widthR, heightR))
```

```
flags = 0
flags = cv2.CALIB_FIX_INTRINSIC

_, newCameraMatrixL, distortionL,
    \ newCameraMatrixR, distortionR, rot, trans, _, _ =
    \ cv2.stereoCalibrate(wrldpoints, imgpointsL,
    \ imgpointsR, newCameraMatrixL, distortionL,
    \ newCameraMatrixR, distortionR, grayL.shape[::-1], ctr, flags)
```

Listing 3: Calibration

4.2 Rectification

Subsequent to the calibration phase, the code integrates a rectification procedure. This rectification is predicated on the utilization of the camera matrices and distortion coefficients, the derivation of which emanates from the aforementioned calibration process. The essential purpose of the rectification transform lies in the alignment of the stereo image pair, such that corresponding points converge onto the same horizontal line, known in the domain of stereovision as the epipolar line. This alignment induces a transformation effect on the images, rendering them as if they emanated from a singular geometric plane.

```
rectifyScale = 1
rectL, rectR, projMatrixL, projMatrixR, _, _, _
    \ = cv2.stereoRectify(newCameraMatrixL, distortionL,
    \ newCameraMatrixR, distortionR,
    \ grayL.shape[::-1], rot, trans, rectifyScale,(0,0))

stereoMapL =
    \ cv2.initUndistortRectifyMap(newCameraMatrixL,
    \ distortionL, rectL, projMatrixL,
    \ grayL.shape[::-1], cv2.CV_16SC2)
stereoMapR =
    \ cv2.initUndistortRectifyMap(newCameraMatrixR,
    \ distortionR, rectR, projMatrixR,
    \ grayR.shape[::-1], cv2.CV_16SC2)

print ("Saving parameters!")
cv_file = cv2.FileStorage('stereoMap.xml',
    \ cv2.FILE_STORAGE_WRITE)
```

```
cv_file.write('stereoMapL_x',stereoMapL[0])
cv_file.write('stereoMapL_y',stereoMapL[1])
cv_file.write('stereoMapR_x',stereoMapR[0])
cv_file.write('stereoMapR_y',stereoMapR[1])

cv_file.release()
print ("Saving completed!")
```

Listing 4: Rectification

Following the application of the rectification transforms, the procedure advances to the generation of the undistortion and rectification maps for each respective camera within the stereo setup. These generated maps hold a subsequent application to each frame captured by the stereo camera arrangement. Such application ensures that each pair of frames undergoes an appropriate and methodical undistortion and rectification. This alignment process enhances the feasibility and precision of subsequent depth estimation efforts. While the output images may manifest a warped appearance, such an artifact is to be understood as a natural consequence of the rectification procedure.

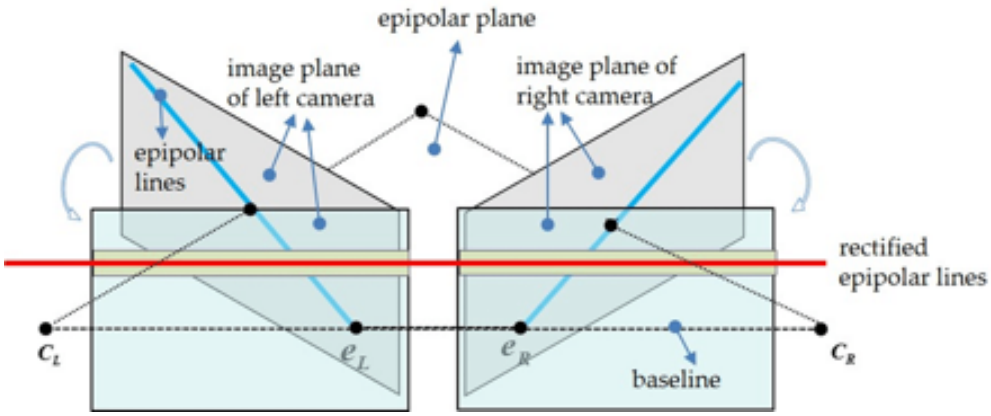

Figure 4: The rectification process

As a final step in this phase, the undistortion and rectification maps are meticulously archived within an XML file. This archival serves a practical purpose, ensuring the accessibility and reusability of these maps within the main pipeline of the project. It emphasizes a systematic approach, reinforcing the interconnection between various stages of the project, and underlining the significance of the rectification process within the broader framework of stereovision analysis.

4.3 Util Library

4.3.1 Undistort Rectify

```
def undistort_rectify(frameR, frameL):
    cv_file = cv2.FileStorage()
    cv_file.open('stereoMap.xml', cv2.FileStorage_READ)

    stereoMapL_x = cv_file.getNode('stereoMapL_x').mat()
    stereoMapL_y = cv_file.getNode('stereoMapL_y').mat()
    stereoMapR_x = cv_file.getNode('stereoMapR_x').mat()
    stereoMapR_y = cv_file.getNode('stereoMapR_y').mat()
    # Undistort and rectify images
    undistortedL= cv2.remap(frameL, stereoMapL_x,
        \ stereoMapL_y, cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
    undistortedR= cv2.remap(frameR, stereoMapR_x,
        \ stereoMapR_y, cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)

    return undistortedR, undistortedL
```

Listing 5: Undistort Rectify

The first function, `undistort_rectify`, takes as input two frames from the right and left cameras. It reads the stereo map (containing undistortion and rectification mappings) that was previously saved during calibration from an XML file. These stereo maps are applied to the input frames using OpenCV's remap function. The undistorted and rectified images for both left and right cameras are returned. This function is critical in preparing the stereo images for subsequent analysis, ensuring that the epipolar lines are horizontal and at the same y-coordinate, thus simplifying disparity computation and other stereo vision tasks.

4.3.2 Triangulation

```
def find_depth(right_point, left_point,
    / frame_right, frame_left, baseline, fov):
    _, width_right, _ = frame_right.shape
    _, width_left, _ = frame_left.shape

    f_pixel = (width_right * 0.5) / np.tan(fov * 0.5 * np.pi/180)

    x_right = right_point[0]
    x_left = left_point[0]
```

```
    # CALCULATE THE DISPARITY between left and right pixel
    disparity = x_left-x_right

    # CALCULATE DEPTH z:
    zDepth = (baseline*f_pixel)/disparity

    return zDepth
```

Listing 6: Undistort Rectify

The second function, `find_depth`, calculates the depth of a specific point in space by using its corresponding points in the right and left camera frames, the frames themselves, the baseline distance between the cameras, and the field of view (fov). The focal length in pixels is computed from the fov and image width. Then, the disparity between the right and left points is calculated as the difference in their x-coordinates. Finally, the depth is derived from the disparity, baseline, and focal length, following the basic principles of stereo triangulation. The depth value represents the distance from the cameras to the point in space and is returned by the function.

4.4 Vision

4.4.1 Face Tracking

A main loop starts by reading frames from both cameras and utilizing the previously defined `undistort_rectify` function from the `utils` module to undistort and rectify the frames. The images are then converted from BGR to RGB for processing with mediapipe's face detection model.

```
while(camera1.isOpened() and camera2.isOpened()):
    with mp_facedetector.
        \ FaceDetection(min_detection_confidence=0.7)
        \ as face_detection:

    ret1, frame_right = camera1.read()
    ret2, frame_left = camera2.read()

    frame_right, frame_left =
        \ utils.undistort_rectify(frame_right, frame_left)

    if not ret1 or not ret2:
        break
```

```
else:

    # Convert the BGR image to RGB
    frame_right = cv2.cvtColor(frame_right, cv2.COLOR_BGR2RGB)
    frame_left = cv2.cvtColor(frame_left, cv2.COLOR_BGR2RGB)

    # Process the image and find faces
    results_right = face_detection.process(frame_right)
    results_left = face_detection.process(frame_left)

    # Convert the RGB image to BGR
    frame_right = cv2.cvtColor(frame_right, cv2.COLOR_RGB2BGR)
    frame_left = cv2.cvtColor(frame_left, cv2.COLOR_RGB2BGR)
```

Listing 7: Face Tracker

Using mediapipe's face detection function, the code detects faces in both the left and right frames. If faces are detected, their bounding boxes and confidence scores are drawn on the frames. If a face is not detected in one or both frames, the "TRACKING LOST" message is displayed on the frames.

4.4.2 Depth Estimation

After detecting the faces, the code calculates the depth or distance to the detected faces using stereo vision principles. This part is tightly interwoven with face tracking. If faces are detected in both frames, the center point of the bounding boxes for the faces in both the right and left frames is determined. These points are then used in the `find_depth` function from the utils module to calculate the depth.

```
if results_right.detections:
    for id, detection in enumerate(results_right.detections):
        mp_draw.draw_detection(frame_right, detection)
        bBox = detection.location_data.relative_bounding_box
        h, w, c = frame_right.shape
        boundBox = int(bBox.xmin * w),
            \ int(bBox.ymin * h), int(bBox.width * w),
            \ int(bBox.height * h)
        center_point_right =
            \ (boundBox[0] + boundBox[2] / 2,
            \ boundBox[1] + boundBox[3] / 2)
```

```
            cv2.putText(frame_right, f'{int(detection.score[0]*100)}%',
                \ (boundBox[0], boundBox[1] - 20),
                \ cv2.FONT_HERSHEY_SIMPLEX, 2, (0,255,0), 2)

if results_left.detections:
    # Same Principles
    ...

if not results_right.detections or not results_left.detections:
    # If no face can be caught on one camera, display "TRACKING LOST"
    ...

else:
    depth = utils.find_depth(center_point_right,
    \ center_point_left, frame_right,
    \ frame_left, baseline, fov)

    #display the depth information and putting them on the frames
    print("Depth: ", str(round(depth,1)))
    ...

# Show the frames
```

Listing 8: Undistort Rectify

Again, the depth is computed by calculating the disparity between the corresponding points in the left and right images and then using the baseline and field of view to compute the distance. The calculated depth is then displayed on both frames and printed in the console.

The processed frames, including the bounding boxes around the detected faces and the calculated depth, are concatenated and displayed side by side. The loop continues to process and display the frames until the "q" key is pressed, at which point the cameras are released, and all windows are destroyed.

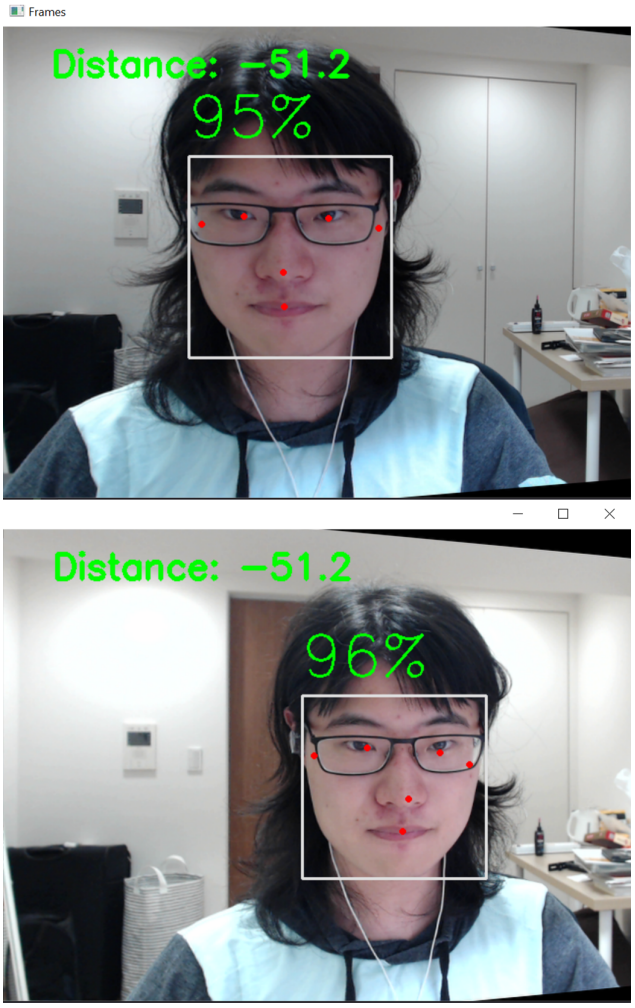

Figure 5: Tracking example

5 Difficulties and Limitations

Despite leveraging the well-established OpenCV libraries, known for their robust functionalities, the calibration process was fraught with complexities and nuances that necessitated careful consideration and troubleshooting.

One difficulty arose from the behavior of the chessboard corner detector, a powerful function within the OpenCV toolkit. The detector's ability to recognize corners, even in images that were out of focus or blurry, was simultaneously impressive and problematic. While the function's sensitivity might seem advantageous, the inability to pinpoint the exact positions of the corners led to calibration results that were markedly affected.

Furthermore, an intuitive strategy of capturing more photographs to enrich the calibration dataset also presented challenges. Though the rationale for acquiring more images was theoretically sound, the actual practice introduced additional complications. Outliers

such as blurry images from an out-of-focus camera, instability from shaky hands, or variances in brightness due to exposure adjustment emerged as deleterious factors. These seemingly minor inconsistencies coalesced to negatively impact the overall accuracy of the stereo setup.

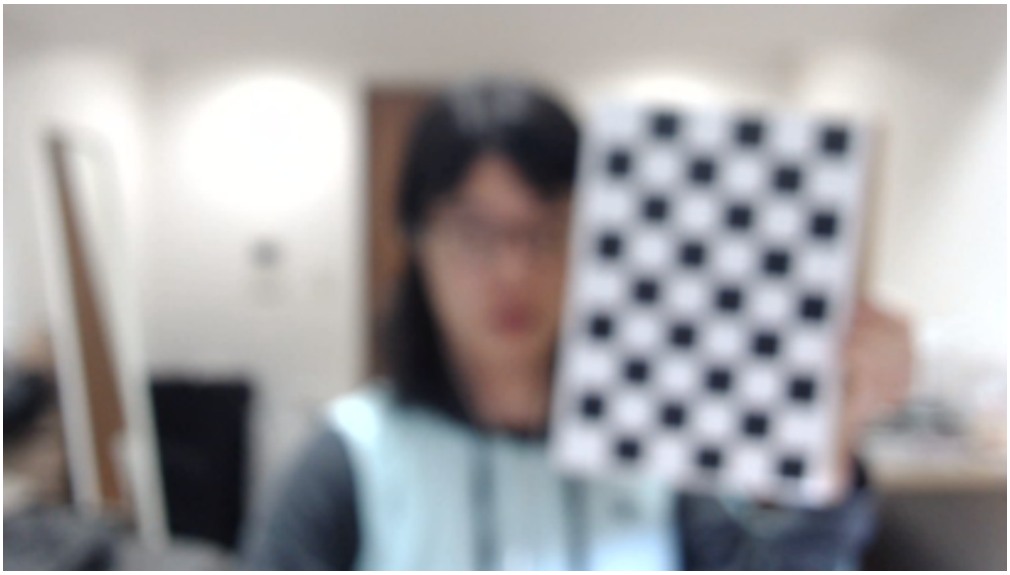

Figure 6: An example of a blurry image

Finally, with the current setup and calibration method, the program is only accurate within a range of approximately twenty centimeters to one meter, which coincides with the typical range for which the reference images are taken during calibration. Within this range, the accuracy can be as high as 98%, but once the tracked face leaves this optimal distance, the accuracy of the program immediately decreases. It is possible to achieve better ranges without adjusting the baseline or the camera orientations by taking more reference images from different distances. However, in addition to the problems described above, the chessboard dimension used in this project is not conducive to the pattern matching function provided by OpenCV for correct identification. A chessboard pattern with fewer edges, or larger squares, would be required. This adjustment would also affect the code implementation, as a multi-stage calibration code would be necessary.

6 Future Potentials

The exploration of stereovision within this project opens modest but interesting possibilities for further investigation that was not explored due to time limitations. While the work conducted is foundational and exploratory in nature, it points towards two practical ap-

plications where the principles employed could find relevance: robotics and 3D reconstruction.

In the domain of robotics, the stereovision techniques developed within this project might be aligned with existing control systems, such as Proportional-Integral-Derivative (PID) controllers. Integration with PID control loops could potentially enhance the feedback mechanisms in robotic navigation. By providing accurate depth information, stereovision could help to refine error correction within the control system, allowing for more precise movement and alignment. Though the current setup may have limitations, it represents a starting point for understanding how vision and control systems might be melded in small-scale robotic applications. This combination could be explored further in academic research or in the development of specialized robotic systems that require nuanced spatial awareness.

On the 3D reconstruction front, the principles employed in this project could be seen as a modest step towards the kind of technology embodied by solutions like Microsoft's Kinect Fusion. Kinect Fusion represents a more advanced application of 3D imaging and reconstruction, utilizing specialized cameras to capture the shape and appearance of objects in real-time. While the work conducted here is not directly comparable to such sophisticated systems, it may provide insights into the underlying concepts of 3D imaging. Understanding the basics of stereovision could lead to a greater appreciation of how complex systems like Kinect Fusion operate, and may serve as an educational tool or a foundation for more advanced studies in the field.

7 References

- https://www.clearview-imaging.com/en/blog/stereo-vision-for-3d-machine-vision-applications
- https://www.researchgate.net/figure/Stereo-vision-disparity_fig1_236455177
- https://markhedleyjones.com/projects/calibration-checkerboard-collection
- https://learnopencv.com/making-a-low-cost-stereo-camera-using-opencv/