Department of Electronic & Telecommunication Engineering,
University of Moratuwa, Sri Lanka.

Depth Camera

Design Details

Group Members:

Abewickrema A.S.J.	210003B
Botheju W.G.W	210071E
Adikari A.A.S.D.	210022G
Lokudadalla L.D.S.D	210346D
Siriwardane C.	210612P
Dassanayaka D.M.T.K.R.	210095F

Submitted in partial fulfillment of the requirements for the module
EN2160 Electronic Design Realization

Date : 07/08/2024

Abstract

This document provides a comprehensive overview of the design details for our Depth Camera project. The project aims to create a portable and cost-effective solution for accurate depth measurement by leveraging computer vision techniques, object detection, and feature matching. This document outlines the project's comprehensive design details, key design considerations, and the daily log entries of the design process highlighting design methodologies employed in the development process.

This design project encompasses several crucial aspects, including hardware design, software integration, and system optimization. The hardware section details the selection and configuration of components, circuit design, enclosure design and PCB layout. The software integration section discusses the development of algorithms and the programming implementation of the depth measuring system. System optimization covers power management and overall system performance enhancement.

Furthermore this document includes the daily log entries that chronicle the design process. These entries provide a detailed account of the day-to-day progress, decisions made, issues faced, and milestones achieved throughout the project. By documenting these daily activities, the report offers insights into the development workflow and helps track the project's evolution over time.

By documenting the design details and daily logs, this report serves as a valuable reference for understanding the intricacies of the Depth Camera project and ensures transparency and reproducibility for future development and enhancements.

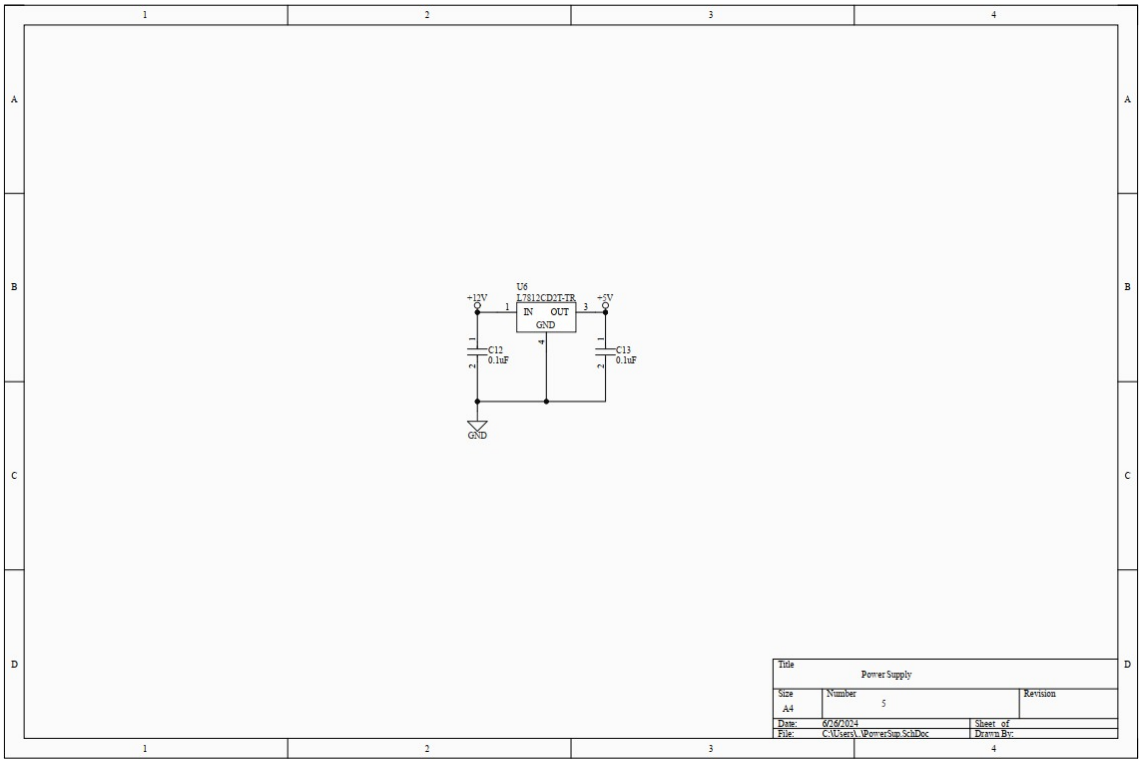
Contents

1 Comprehensive Design Details

1.1 Circuit Analysis

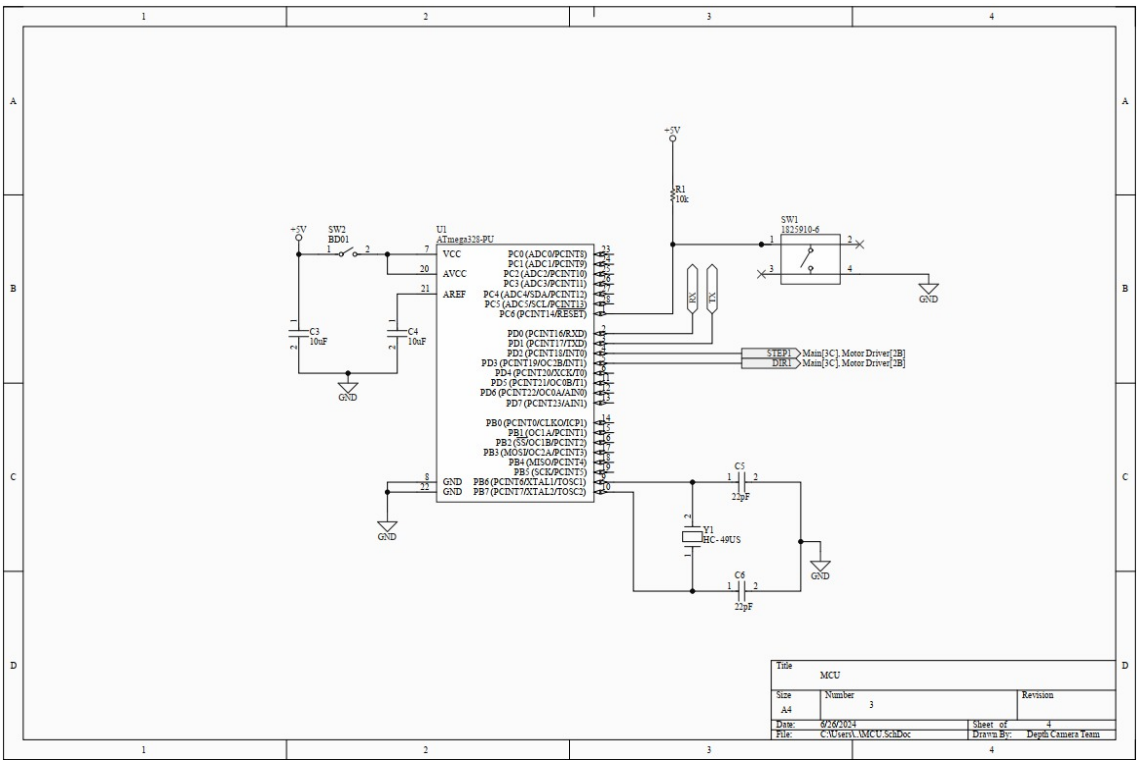
1.1.1 Power supply

Our power supply is a fairly simple setup. As the input, we use a 12v power supply (provided through a power-regulated power supply) and then inside the device by using a L7812 voltage regulator we also make our own 5v power supply inside the circuit. Then we use this to power up the atmega328 chip and servo motor driver. The primary components include a voltage regulator, capacitors for filtering, and connectors for input and output.



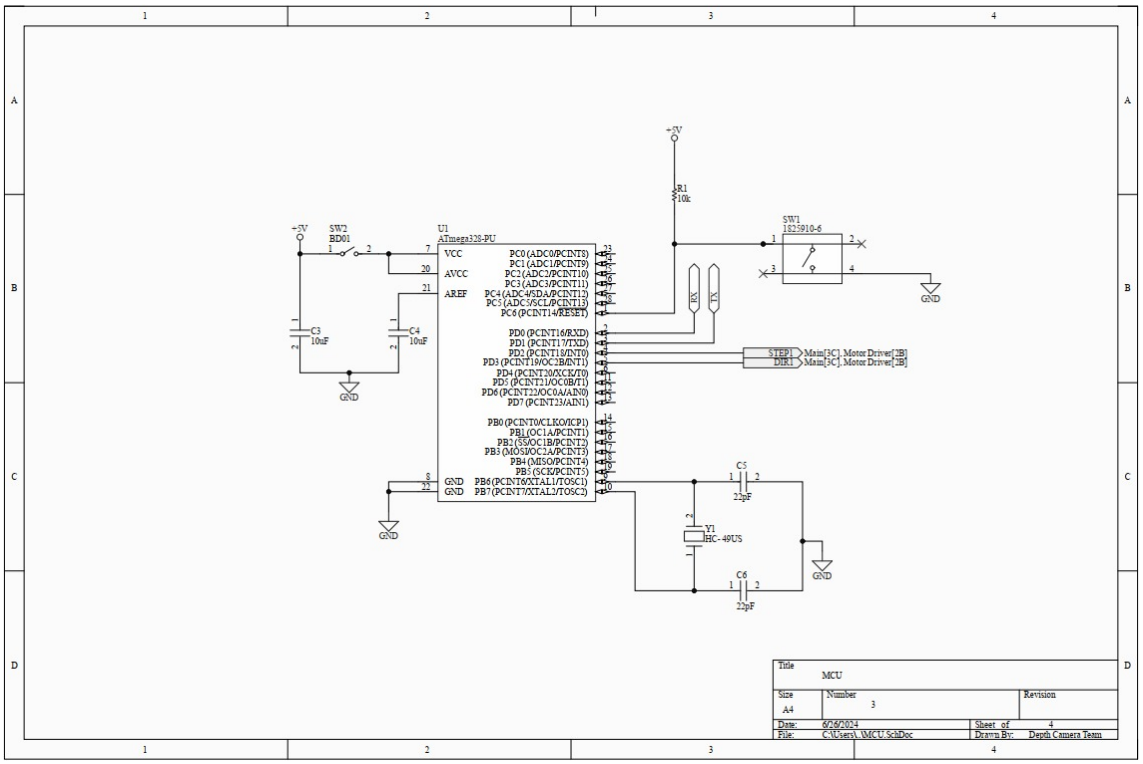
1.1.2 Stepper motor driver carrier

The stepper motor driver drives the stepper motor, converting digital signals from the microcontroller into precise movements. It features the A4988 stepper motor driver, which controls the current flow to the motor windings, enabling accurate positioning.



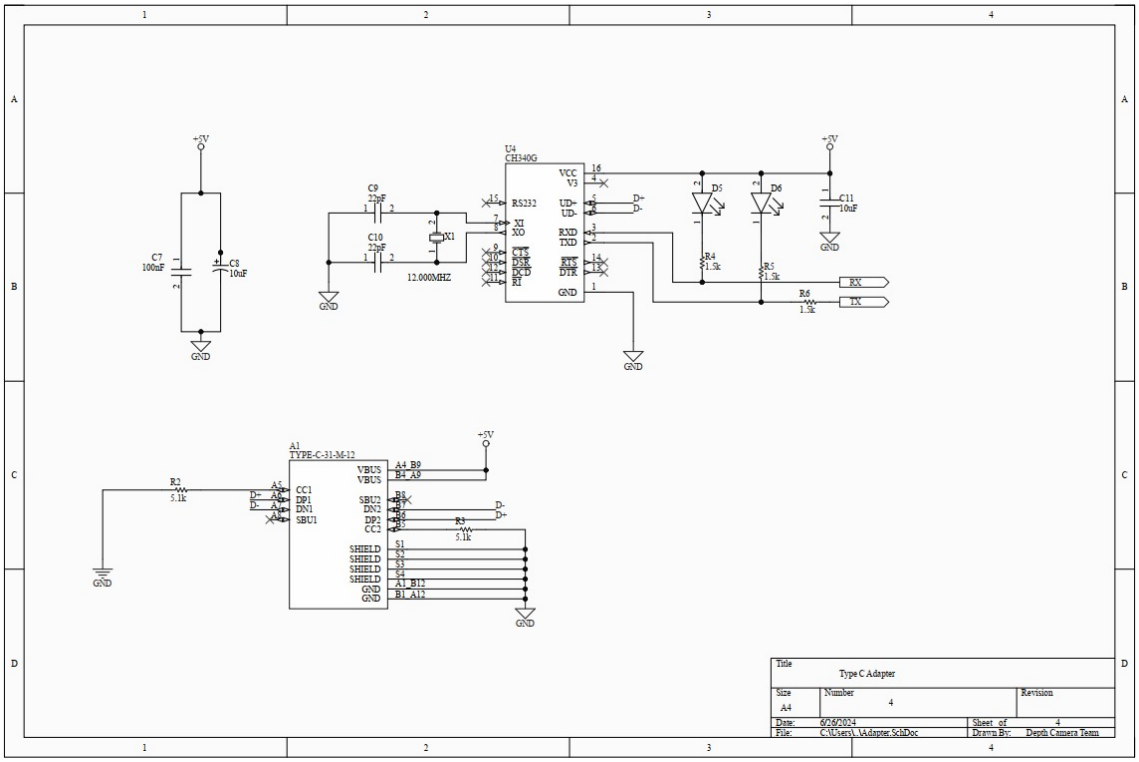
1.1.3 Microcontroller Unit

This module incorporates the ATmega328P microcontroller, which serves as the central processing unit for the circuit. It receives power from the power supply, controls the motor driver, and interfaces with other peripherals. The tactile switch SW2 could be used to start or stop the motor, change modes of operation, or initiate specific functions in the program running on the microcontroller. Here, we use it as a reset button. Other key components include the ATmega328P, crystal oscillator for clock generation, and decoupling capacitors.



1.1.4 Type C adapter

The Type C adapter circuit facilitates communication between the microcontroller and external devices through a USB Type C connector. It includes an RS232 transceiver for serial communication, capacitors for stabilization, and connectors for the USB interface. In this adapter circuit, the CH340 serves as an interface for serial communication. It converts USB signals from a host device (such as a computer) into UART signals that the microcontroller can understand and vice versa.



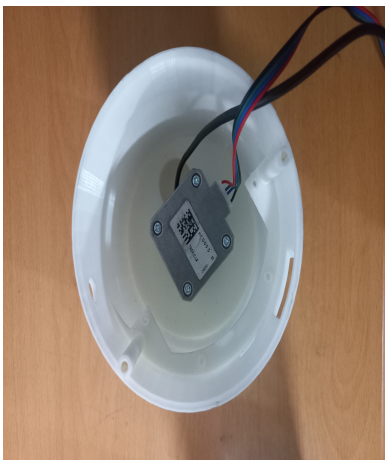
1.2 Enclosure analysis

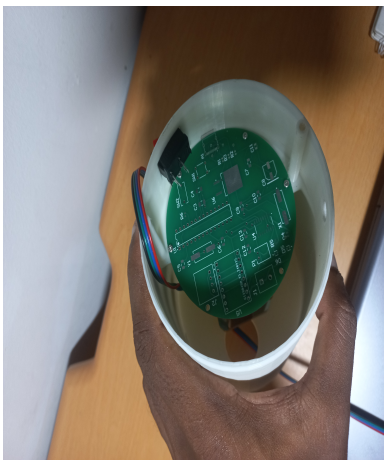
1.2.1 Main Body of the Enclosure

The 3D printed main body of the enclosure is cylindrical, chosen for its compatibility with the camera's rotational mechanism, ensuring smooth and efficient movement. This shape maximizes space utilization, allowing for the effective integration of the PCB and other internal components. Additionally, the cylindrical design enhances structural integrity and aesthetic appeal. The model includes a compartment for the stepper motor holder and a circular cutout for wire management, further improving functionality and ease of assembly.

1.2.2 Opposite Side View of the Main Body

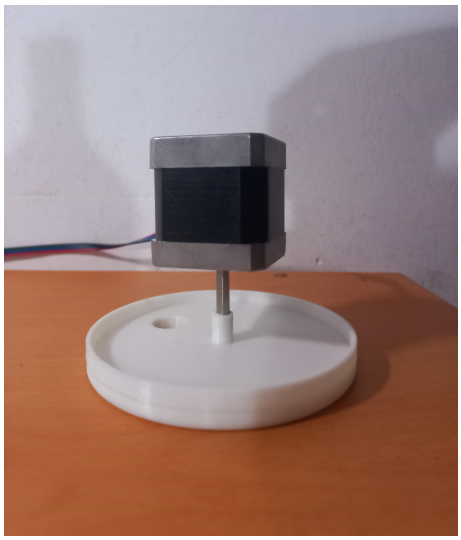
From the opposite side view of the 3D printed main body, the back lid attachment point is visible, providing a secure closure for the enclosure. The PCB holder is also shown, demonstrating the well-designed placement and support structure for the internal electronics. This setup ensures that the PCB remains securely in place, facilitating effective integration and protection of the electronic components.





1.2.3 Rotating Component

The rotating component, designed to interface with the stepper motor, is a critical part of the 3D printed model. This part ensures precise rotational movement, which is essential for accurate camera positioning and operation. The design includes secure attachment points for both the stepper motor and the camera, ensuring stability and reliability during use.

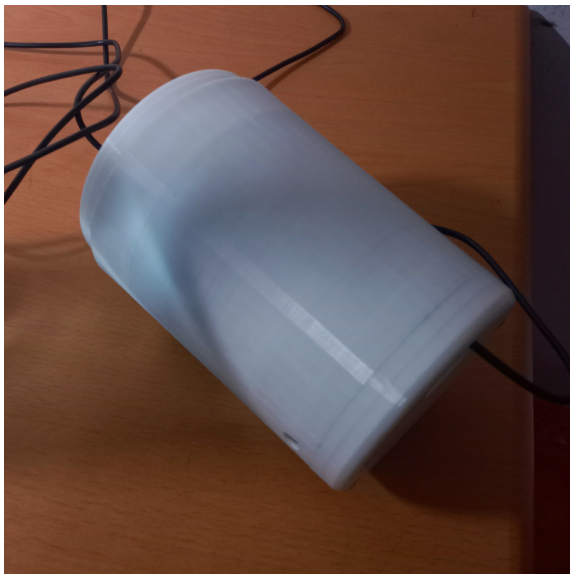


1.2.4 Back Lid of the Enclosure

The back lid of the 3D printed enclosure features a circular hole for wire management, allowing the camera wires to be routed externally and connected to the PC, which serves as the project's processor. This design ensures organized cable routing and secure connections, enhancing the overall efficiency and functionality of the system.

1.2.5 Complete Integrated Design

The complete 3D printed design integrates the main body, rotating component, and back lid seamlessly. The assembly clearly shows the PCB and stepper motor, highlighting their well-considered placement and integration within the enclosure. This comprehensive design ensures that all components work together cohesively, providing a robust and efficient solution for the depth camera system.



1.3 Programming Analysis

This section provides a comprehensive overview of the programming aspects of our depth camera project, detailing the implementation and testing processes involved in capturing images, calibrating cameras, detecting objects, matching features, and calculating depth.

1.3.1 Camera Calibration

- Objective: Calibrate the camera to obtain intrinsic and extrinsic parameters.

In the calibration.py code calibrate_camera() function is used to calibrate the camera and obtain the intrinsic parameters which are the camera matrix and distortion matrix. In this process we have used images of a 9,6 chessboard. OpenCV library is used to identify the chessboard corners and do the calibration process. frameSize is set to the resolution of images taken from the camera.

```
1
2  def calibrate_camera(images):
3      ####### FIND CHESSBOARD CORNERS - OBJECT POINTS AND IMAGE POINTS ######
4
5      chessboardSize = (9,6)  # Define the size of the chessboard
6      frameSize = (1920,1080)  # Define the size of the frames used for calibration
7
8      # Termination criteria for corner refinement
9      criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
10
11     # Prepare object points (3D points in real world space)
12     objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
13     objp[:,:2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]].T.reshape(-1,2)
14     size_of_chessboard_squares_mm = 20  # Size of a square in mm
15     objp = objp * size_of_chessboard_squares_mm
16
17     # Arrays to store object points and image points from all the images
18     objpoints = []  # 3D points in real world space
19     imgpoints = []  # 2D points in image plane
20
21     # Loop through all images for calibration
22     for image in images:
23         img = cv.imread(image)
24         gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
25
26         # Find the chessboard corners
27         ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)
28
29         # If found, refine and add object and image points
30         if ret == True:
31             objpoints.append(objp)
32             corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
33             imgpoints.append(corners)
34
35             # Draw and display the corners
36             cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
37             cv.imshow('img', img)
38             cv.waitKey(1000)
39
40     cv.destroyAllWindows()
```

Listing 1: calibrate_camera Function

- Stereo Camera Calibration

The stereo_calibrate() function is used obtain the extrinsic parameters. The camera matrix and the distortion matrix obtained from the calibrate_camera() function is used to obtain extrinsic parameters alongside the chessboard images from the two camera positions. This function returns the Rotational matrix and the transational matrix between the two camera postions. OpenCV is library is used to get the extrinsic parameters.

```

def stereo_calibrate(mtx, dist, c1_images1, c2_images1):
    # Read synchronized frames from both cameras
    c1_images = []
    c2_images = []
    for im1 in c1_images1:
        _im = cv.imread(im1)
        if _im is None:
            print(f"Failed to load image: {im1}")
            continue
        c1_images.append(_im)

    for im2 in c2_images1:
        _im = cv.imread(im2)
        if _im is None:
            print(f"Failed to load image: {im2}")
            continue
        c2_images.append(_im)

    # Termination criteria for corner refinement
    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    rows = 9  # Number of checkerboard rows
    columns = 6  # Number of checkerboard columns
    world_scaling = 20.  # Square size of a square in mm

    # Coordinates of squares in the checkerboard space
    objp = np.zeros((rows*columns,3), np.float32)
    objp[:,:2] = np.mgrid[0:rows,0:columns].T.reshape(-1,2)
    objp = world_scaling * objp

    # Frame dimensions
    frameSize = (1920,1080)

    # Pixel coordinates of checkerboards
    imgpoints_left = []  # 2D points in image plane (left camera)
    imgpoints_right = []  # 2D points in image plane (right camera)

    # Coordinates of the checkerboard in checkerboard space
    objpoints = []  # 3D points in real space

    # Loop through image pairs
    for frame1, frame2 in zip(c1_images, c2_images):
        gray1 = cv.cvtColor(frame1, cv.COLOR_BGR2GRAY)
        gray2 = cv.cvtColor(frame2, cv.COLOR_BGR2GRAY)
        c_ret1, corners1 = cv.findChessboardCorners(gray1, (9, 6), None)
        c_ret2, corners2 = cv.findChessboardCorners(gray2, (9, 6), None)

        # If corners are found in both images, refine and add points
        if c_ret1 == True and c_ret2 == True:
            corners1 = cv.cornerSubPix(gray1, corners1, (11, 11), (-1, -1), criteria)
            corners2 = cv.cornerSubPix(gray2, corners2, (11, 11), (-1, -1), criteria)

            cv.drawChessboardCorners(frame1, (9,6), corners1, c_ret1)
            cv.imshow('img', frame1)

            cv.drawChessboardCorners(frame2, (9,6), corners2, c_ret2)
            cv.imshow('img2', frame2)
```

```
            cv.waitKey(1000)

            objpoints.append(objp)
            imgpoints_left.append(corners1)
            imgpoints_right.append(corners2)

    print(f"Number of valid calibration pairs: {len(objpoints)}")
    if len(objpoints) == 0:
        raise Exception("No valid image points found for stereo calibration. Check image paths and checkerboard detection.")

    # Stereo calibration flags
    stereocalibration_flags = cv.CALIB_FIX_INTRINSIC
    ret, CM1, dist1, CM2, dist2, R, T, E, F = cv.stereoCalibrate(objpoints, imgpoints_left, imgpoints_right, mtx, dist,mtx, dist, frameSize, criteria=criteria, flags=stereocalibration_flags)

    print(ret)
    return R, T
```

Listing 2: stereo_calibrate Function

Two functions are called with images paths as input arguments to obtain Camera Matrix, Rotational Matrix and Translational Matrix which are needed to calculate the depth to a given object.

```

import numpy as np
import cv2 as cv
import glob

def calibrate_camera(images):
    ################ FIND CHESSBOARD CORNERS - OBJECT POINTS AND IMAGE POINTS #############################

    chessboardSize = (9,6)  # Define the size of the chessboard
    frameSize = (1920,1080)  # Define the size of the frames used for calibration

    # Termination criteria for corner refinement
    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    # Prepare object points (3D points in real world space)
    objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
    objp[:,:2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]].T.reshape(-1,2)
    size_of_chessboard_squares_mm = 20  # Size of a square in mm
    objp = objp * size_of_chessboard_squares_mm

    # Arrays to store object points and image points from all the images
    objpoints = []  # 3D points in real world space
    imgpoints = []  # 2D points in image plane

    # Loop through all images for calibration
    for image in images:
        img = cv.imread(image)
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

        # Find the chessboard corners
        ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)

        # If found, refine and add object and image points
        if ret == True:
            objpoints.append(objp)
            corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
            imgpoints.append(corners)

            # Draw and display the corners
            cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
```

```
            cv.imshow('img', img)
            cv.waitKey(1000)

    cv.destroyAllWindows()

    ############### CALIBRATION PROCESS ################

    ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints,
    frameSize, None, None)
    print(ret)
    print(cameraMatrix)
    return cameraMatrix, dist

def stereo_calibrate(mtx, dist, c1_images1, c2_images1):
    # Read synchronized frames from both cameras
    c1_images = []
    c2_images = []
    for im1 in c1_images1:
        _im = cv.imread(im1)
        if _im is None:
            print(f"Failed to load image: {im1}")
            continue
        c1_images.append(_im)

    for im2 in c2_images1:
        _im = cv.imread(im2)
        if _im is None:
            print(f"Failed to load image: {im2}")
            continue
        c2_images.append(_im)

    # Termination criteria for corner refinement
    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    rows = 9  # Number of checkerboard rows
    columns = 6  # Number of checkerboard columns
    world_scaling = 20.  # Square size of a square in mm

    # Coordinates of squares in the checkerboard space
    objp = np.zeros((rows*columns,3), np.float32)
    objp[:,:2] = np.mgrid[0:rows,0:columns].T.reshape(-1,2)
    objp = world_scaling * objp

    # Frame dimensions
    frameSize = (1920,1080)

    # Pixel coordinates of checkerboards
    imgpoints_left = []  # 2D points in image plane (left camera)
    imgpoints_right = []  # 2D points in image plane (right camera)

    # Coordinates of the checkerboard in checkerboard space
    objpoints = []  # 3D points in real space

    # Loop through image pairs
    for frame1, frame2 in zip(c1_images, c2_images):
        gray1 = cv.cvtColor(frame1, cv.COLOR_BGR2GRAY)
        gray2 = cv.cvtColor(frame2, cv.COLOR_BGR2GRAY)
        c_ret1, corners1 = cv.findChessboardCorners(gray1, (9, 6), None)
        c_ret2, corners2 = cv.findChessboardCorners(gray2, (9, 6), None)

        # If corners are found in both images, refine and add points
        if c_ret1 == True and c_ret2 == True:
            corners1 = cv.cornerSubPix(gray1, corners1, (11, 11), (-1, -1), criteria)
            corners2 = cv.cornerSubPix(gray2, corners2, (11, 11), (-1, -1), criteria)

            cv.drawChessboardCorners(frame1, (9,6), corners1, c_ret1)
            cv.imshow('img', frame1)

```

```
            cv.drawChessboardCorners(frame2, (9,6), corners2, c_ret2)
            cv.imshow('img2', frame2)
            cv.waitKey(1000)

            objpoints.append(objp)
            imgpoints_left.append(corners1)
            imgpoints_right.append(corners2)

    print(f"Number of valid calibration pairs: {len(objpoints)}")
    if len(objpoints) == 0:
        raise Exception("No valid image points found for stereo calibration. Check image paths and checkerboard detection.")

    # Stereo calibration flags
    stereocalibration_flags = cv.CALIB_FIX_INTRINSIC
    ret, CM1, dist1, CM2, dist2, R, T, E, F = cv.stereoCalibrate(objpoints, imgpoints_left, imgpoints_right, mtx, dist,mtx, dist, frameSize, criteria=criteria, flags=stereocalibration_flags)

    print(ret)
    return R, T

# Calibrate single camera
images = glob.glob('cameraCalibration/images/*.png')
mtx, dist = calibrate_camera(images)

# Read synchronized frames from both positions
c1_images = glob.glob("CameraCalibration/images2/*.png")
c1_images = sorted(c1_images)
c2_images = glob.glob("CameraCalibration/images1/*.png")
c2_images = sorted(c2_images)

# Calibrate the stereo camera system
R, T = stereo_calibrate(mtx, dist, mtx, dist, c1_images, c2_images)

print("R =", R)
print("T =", T)
```

Listing 3: calibration.py

- **Results**

  - The calibration process provided the camera matrix and distortion coefficients for the camera.
  - The stereo calibration provided the rotation and translation matrices between the two camera positions.

1.3.2 Training the Object Detection with YOLOv8

- Objective: Implement object detection using the YOLOv8 model, custom-trained to detect the object and obtain the bounding box before measuring the depth.

Overview of YOLOv8:

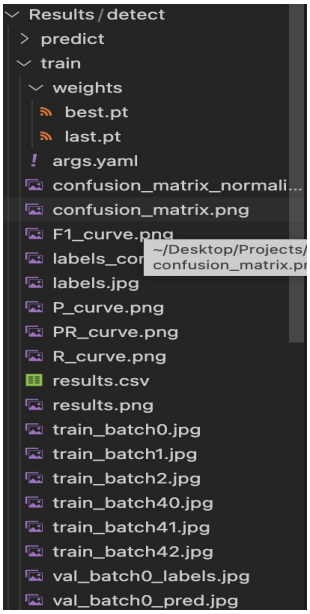
YOLOv8, developed by Ultralytics, is a state-of-the-art object detection model known for its speed and accuracy. It builds upon previous YOLO versions with improved backbone architectures, an anchor-free detection head, and better handling of small objects. YOLOv8 supports tasks like object detection, instance segmentation, and image classification, making it versatile for various applications.

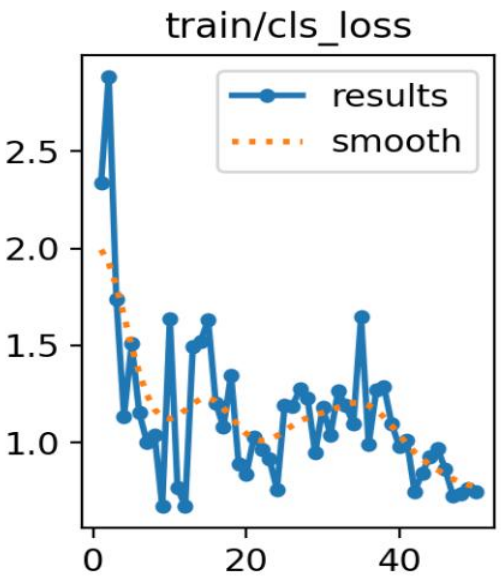
- Model Training
  - Used a custom dataset created using RoboFlow.
  - Trained the YOLOv8 model on this dataset, adjusting parameters like learning rate and batch size for optimal performance.

```
1
2 from roboflow import Roboflow
3 from ultralytics import YOLO
4
5 # Set the path for images and model weights
6 sample_image_path = "/Users/sasindu/Desktop/Depth Camera/CameraCalibration/left.jpg"
7 new_test_image_path = "/Users/sasindu/Desktop/Depth Camera/CameraCalibration/right.
      jpg"
8
9 # Initialize Roboflow
10 rf = Roboflow(api_key="3IDDv08wLwsxDOg32ia9")
11 project = rf.workspace("sandaru-o5zia").project("bottle_detector-lnu6c")
12 version = project.version(1)
13 dataset = version.download("yolov8-obb")
14
15 # Train the model
16 train_model = YOLO('yolov8m.pt')
17 results= train_model.train(data="/Users/sasindu/Desktop/DC/bottle_detector-1/data.
      yaml", epochs=20, imgsz=640)
```

Listing 4: Model_Training.py

Weights of the model is saved in best.pt to later detect obejcts with the model.





1.3.3 Measuring the depth

- Objective: To detect the depth to an given object after obtaining the Camera matrix, Rotational Matrix and Translational Matrix from calibration.py and getting the trained model to detect a given object from Training_Model.py

1. Capturing the Image
   - Objective: Implement a code to capture a photo from the USB webcam when the 'S' key is pressed.
   - Implementation: We used OpenCV to interact with the USB webcam and capture images. The code allows capturing images by pressing the 'S' key and exits on pressing 'Q' or interrupt.

```
def ImageCapture(input_string):
    key = cv. waitKey(1)
    webcam = cv.VideoCapture(0)
    while True:
        try:
            check, frame = webcam.read()
            #print(check) #prints true as long as the webcam is running
            #print(frame) #prints matrix values of each framecd
            cv.imshow("Capturing", frame)
            key = cv.waitKey(1)
            if key == ord('s'):
                cv.imwrite(filename= input_string, img=frame)
                webcam.release()
                cv.waitKey(1650)
                cv.destroyAllWindows()

                break
            elif key == ord('q'):
                print("Turning off camera.")
                webcam.release()
                print("Camera off.")
```

```
                print("Program ended.")
                cv.destroyAllWindows()
                break

        except(KeyboardInterrupt):
            print("Turning off camera.")
            webcam.release()
            print("Camera off.")
            print("Program ended.")
            cv.destroyAllWindows()
            break  def ImageCapture(input_string):
    key = cv. waitKey(1)
    webcam = cv.VideoCapture(0)
    while True:
        try:
            check, frame = webcam.read()
            #print(check) #prints true as long as the webcam is running
            #print(frame) #prints matrix values of each framecd
            cv.imshow("Capturing", frame)
            key = cv.waitKey(1)
            if key == ord('s'):
                cv.imwrite(filename= input_string, img=frame)
                webcam.release()
                cv.waitKey(1650)
                cv.destroyAllWindows()

                break
            elif key == ord('q'):
                print("Turning off camera.")
                webcam.release()
                print("Camera off.")
                print("Program ended.")
                cv.destroyAllWindows()
                break

        except(KeyboardInterrupt):
            print("Turning off camera.")
            webcam.release()
            print("Camera off.")
            print("Program ended.")
            cv.destroyAllWindows()
            break

ImageCapture('left1.jpg')

```

Listing 5: Image Capturing

2. Code to Send a signal to the atmega328p via USB C port.

    - Objective: To make a pin to high state in ATmega328P to let the ATmega328P know now its time to rotate the camera to the other position.

```
# Function to list available serial ports
def list_ports():
    ports = serial.tools.list_ports.comports()
    ports_list = []
    for one_port in ports:
        ports_list.append(str(one_port))
        print(str(one_port))
    return ports_list

# Function to select a port
def select_port(ports_list):
```

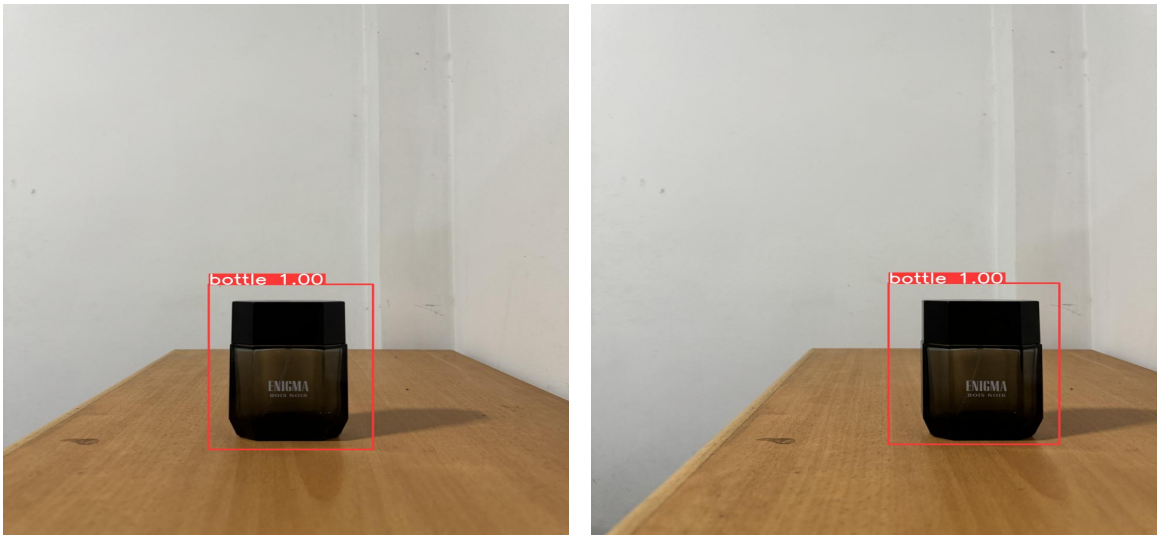
```
    val = input("Select Port: COM")
    for x in range(0, len(ports_list)):
        if ports_list[x].startswith("COM" + str(val)):
            port_var = "COM" + str(val)
            print(port_var)
            return port_var
    return None

# Function to send a command to the Arduino
def send_command(serial_instance, command):
    serial_instance.write(command.encode('utf-8'))

# Main script
ports_list = list_ports()
port_var = select_port(ports_list)

if port_var:
    serial_inst = serial.Serial()
    serial_inst.baudrate = 9600
    serial_inst.port = port_var
    serial_inst.open()

    # Set pin to high state
    send_command(serial_inst, 'H')  # Assuming 'H' sets the pin high
    print("Pin set to HIGH state.")
    time.sleep(1)  # Wait for 1 second or the duration you need

    serial_inst.close()
else:
    print("No valid port selected.")
```

3. Detecting the object

   - Objective: To detect the object in left and right images and obtain the bounding boxes of the two images to do feature matching
   - Implementation: Obtained the bounding box coordinates from .boxes.xywh.tolist() function and since this returns centre coordinates of the bounding box converted the coordinates to be the top left coordinates.

```
# Predict using the trained model
trained_model_path = "/Users/sasindu/Desktop/DC/Results/detect/train/weights/
    best.pt"
trained_model = YOLO(trained_model_path)
results = trained_model.predict(["/Users/sasindu/Desktop/DC/left.jpg","/Users/
    sasindu/Desktop/DC/right.jpg"],save=True, imgsz=640, conf=0.98)


# Extract bounding boxes, classes, names, and confidences
boxes1 = results[0].boxes.xywh.tolist()
boxes2 = results[1].boxes.xywh.tolist()


def center_to_topleft(cx, cy, w, h):

    x = cx - w / 2
    y = cy - h / 2
    bbx1 = [int(x),int(y),int(w),int(h)]
    return bbx1

#Bounding Boxes

bbox1= center_to_topleft(boxes1[0][0], boxes1[0][1], boxes1[0][2], boxes1[0][3])
bbox2= center_to_topleft(boxes2[0][0], boxes2[0][1], boxes2[0][2], boxes2[0][3])
```

- **Results**



4. Feature Matching

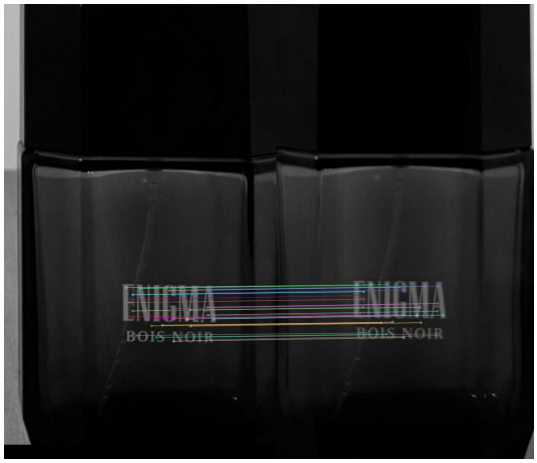
- Objective: Match features within the detected bounding boxes using various algorithms.
- Steps
  (a) Feature Detection Algorithms
    – Experimented with SIFT, ORB, and SURF for feature detection.
    – Chose SIFT for its robustness and accuracy.
  (b) Feature Matching
    – Implemented feature matching using the BFMatcher with NORM_L2 norm.

```
def match_features_in_bboxes(img1, img2, bbox1, bbox2,
    distance_threshold=0.4):
    """
    Match SIFT features within specified bounding boxes in two images,
    filter by match quality,
    and prepare for triangulation using the BFMatcher with NORM_L2.

    Args:
    img1, img2: Input images.
    bbox1, bbox2: Bounding boxes in each image as (x, y, width, height).
    distance_threshold: Ratio threshold for filtering good matches based
     on Lowe's ratio test.

    Returns:
    pts1, pts2: Arrays of the matched keypoints from each image, ready
    for triangulation.
    """
    # Extract regions of interest
    roi1 = img1[bbox1[1]:bbox1[1] + bbox1[3], bbox1[0]:bbox1[0] + bbox1
    [2]]
    roi2 = img2[bbox2[1]:bbox2[1] + bbox2[3], bbox2[0]:bbox2[0] + bbox2
    [2]]
```

```
17
18      # Initialize SIFT detector
19      sift = cv.SIFT_create()
20
21      # Detect and compute SIFT features in ROIs
22      kp1, des1 = sift.detectAndCompute(roi1, None)
23      kp2, des2 = sift.detectAndCompute(roi2, None)
24
25      # Create BFMatcher object with distance norm L2
26      bf = cv.BFMatcher(cv.NORM_L2, crossCheck=False)
27
28      # Match descriptors and apply Lowe's ratio test
29      matches = bf.knnMatch(des1, des2, k=2)
30      good_matches = []
31      for m, n in matches:
32          if m.distance < distance_threshold * n.distance:
33              good_matches.append(m)
34
35      # Extract coordinates of matched keypoints
36      pts1 = np.float64([kp1[m.queryIdx].pt for m in good_matches]).
        reshape(-1, 2)
37      pts2 = np.float64([kp2[m.trainIdx].pt for m in good_matches]).
        reshape(-1, 2)
38
39      # Adjust points to the full image coordinates
40      pts1 += np.array([bbox1[0], bbox1[1]])
41      pts2 += np.array([bbox2[0], bbox2[1]])
42
43      # Draw matches
44      match_img = cv.drawMatches(roi1, kp1, roi2, kp2, good_matches, None,
         flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
45
46      # Save the match image
47      match_img_path = "/Users/sasindu/Desktop/DC/match_features.jpg"
48      cv.imwrite(match_img_path, match_img)
49
50      return pts1, pts2
51
52  img1 = cv.imread("/Users/sasindu/Desktop/DC/left.jpg", cv.
        IMREAD_GRAYSCALE)  # Left image
53  img2 = cv.imread("/Users/sasindu/Desktop/DC/right.jpg", cv.
        IMREAD_GRAYSCALE)  # Right image
54
55  pts1, pts2 = match_features_in_bboxes(img1, img2, bbox1, bbox2)
56
57
```

Listing 6: Feature Matching

- **Results**
  - Successfully matched features within the detected bounding boxes using SIFT and BFMatcher.



5. Depth Measuring using Triangulation

- Objective: To calculate the depth using the matched points in the two images
- Implementation: Calculated depth using the matched points in the two images, Rotaional Matrix, Translational Matrix and Camera Matrix using the triangulation method.

```
img1 = cv.imread("/Users/sasindu/Desktop/DC/left.jpg", cv.IMREAD_GRAYSCALE)  # Left image
img2 = cv.imread("/Users/sasindu/Desktop/DC/right.jpg", cv.IMREAD_GRAYSCALE)  # Right image

pts1, pts2 = match_features_in_bboxes(img1, img2, bbox1, bbox2)

R = np.array([[ 1, 0, 0],
 [ 0,  1, 0],
 [0,  0,  1]])

T = np.array([[-60],
 [ 0],
 [ 0]])

mtx = np.array([[4.06850113e+03, 0.00000000e+00, 2.11799463e+03],
 [0.00000000e+00, 4.07285720e+03, 2.84485951e+03],
 [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])

P1 = np.hstack((mtx, np.zeros((3, 1))))  # Projection matrix for the first camera
P2 = mtx @ np.hstack((R, T.reshape(-1, 1)))  # Projection matrix for the second Position

pts1 = pts1.T
pts2 = pts2.T

# Triangulate points
points4D = cv.triangulatePoints(P1, P2, pts1, pts2)
points3D = points4D[:3] / points4D[3]
depths = points3D[2, :]  # Extract Z (depth information)

#Printing Depth
print("Depth :", np.mean(depths))
```

Complete Final Code

```
1
2  import cv2 as cv
3  import time
4  import numpy as np
5  import serial.tools.list_ports
6
7  from ultralytics import YOLO
8
9  def ImageCapture(input_string):
10     key = cv. waitKey(1)
11     webcam = cv.VideoCapture(0)
12     while True:
13         try:
14             check, frame = webcam.read()
15             #print(check) #prints true as long as the webcam is running
16             #print(frame) #prints matrix values of each framecd
17             cv.imshow("Capturing", frame)
18             key = cv.waitKey(1)
19             if key == ord('s'):
20                 cv.imwrite(filename= input_string, img=frame)
21                 webcam.release()
22                 cv.waitKey(1650)
23                 cv.destroyAllWindows()
24
25                 break
26             elif key == ord('q'):
27                 print("Turning off camera.")
28                 webcam.release()
29                 print("Camera off.")
30                 print("Program ended.")
31                 cv.destroyAllWindows()
32                 break
33
34         except(KeyboardInterrupt):
35             print("Turning off camera.")
36             webcam.release()
37             print("Camera off.")
38             print("Program ended.")
39             cv.destroyAllWindows()
40             break
41
42 ImageCapture('left1.jpg')
43
44 # Function to list available serial ports
45 def list_ports():
46     ports = serial.tools.list_ports.comports()
47     ports_list = []
48     for one_port in ports:
49         ports_list.append(str(one_port))
50         print(str(one_port))
51     return ports_list
52
53 # Function to select a port
54 def select_port(ports_list):
55     val = input("Select Port: COM")
56     for x in range(0, len(ports_list)):
57         if ports_list[x].startswith("COM" + str(val)):
58             port_var = "COM" + str(val)
59             print(port_var)
60             return port_var
61     return None
62
63 # Function to send a command to the Arduino
64 def send_command(serial_instance, command):
```

```
    serial_instance.write(command.encode('utf-8'))

# Main script
ports_list = list_ports()
port_var = select_port(ports_list)

if port_var:
    serial_inst = serial.Serial()
    serial_inst.baudrate = 9600
    serial_inst.port = port_var
    serial_inst.open()

    # Set pin to high state
    send_command(serial_inst, 'H')  # Assuming 'H' sets the pin high
    print("Pin set to HIGH state.")
    time.sleep(1)  # Wait for 1 second or the duration you need

    serial_inst.close()
else:
    print("No valid port selected.")

time.sleep(10) #waiting time to let the camera rotate to the other position

ImageCapture('right2.jpg')     #taking the right picture once the camera is done rotating

# Predict using the trained model
trained_model_path = "/Users/sasindu/Desktop/DC/Results/detect/train/weights/best.pt"
trained_model = YOLO(trained_model_path)
results = trained_model.predict(["/Users/sasindu/Desktop/DC/left.jpg","/Users/sasindu/Desktop/DC/right.jpg"],save=True, imgsz=640, conf=0.98)


# Extract bounding boxes, classes, names, and confidences
boxes1 = results[0].boxes.xywh.tolist()
boxes2 = results[1].boxes.xywh.tolist()


def center_to_topleft(cx, cy, w, h):

    x = cx - w / 2
    y = cy - h / 2
    bbx1 = [int(x),int(y),int(w),int(h)]
    return bbx1

#Bounding Boxes

bbox1= center_to_topleft(boxes1[0][0], boxes1[0][1], boxes1[0][2], boxes1[0][3])
bbox2= center_to_topleft(boxes2[0][0], boxes2[0][1], boxes2[0][2], boxes2[0][3])

def match_features_in_bboxes(img1, img2, bbox1, bbox2, distance_threshold=0.4):
    """
    Match SIFT features within specified bounding boxes in two images, filter by match quality,
    and prepare for triangulation using the BFMatcher with NORM_L2.

    Args:
    img1, img2: Input images.
    bbox1, bbox2: Bounding boxes in each image as (x, y, width, height).
    distance_threshold: Ratio threshold for filtering good matches based on Lowe's ratio test.

    Returns:
    pts1, pts2: Arrays of the matched keypoints from each image, ready for triangulation.
    """
    # Extract regions of interest
    roi1 = img1[bbox1[1]:bbox1[1] + bbox1[3], bbox1[0]:bbox1[0] + bbox1[2]]
```

```
    roi2 = img2[bbox2[1]:bbox2[1] + bbox2[3], bbox2[0]:bbox2[0] + bbox2[2]]

    # Initialize SIFT detector
    sift = cv.SIFT_create()

    # Detect and compute SIFT features in ROIs
    kp1, des1 = sift.detectAndCompute(roi1, None)
    kp2, des2 = sift.detectAndCompute(roi2, None)

    # Create BFMatcher object with distance norm L2
    bf = cv.BFMatcher(cv.NORM_L2, crossCheck=False)

    # Match descriptors and apply Lowe's ratio test
    matches = bf.knnMatch(des1, des2, k=2)
    good_matches = []
    for m, n in matches:
        if m.distance < distance_threshold * n.distance:
            good_matches.append(m)

    # Extract coordinates of matched keypoints
    pts1 = np.float64([kp1[m.queryIdx].pt for m in good_matches]).reshape(-1, 2)
    pts2 = np.float64([kp2[m.trainIdx].pt for m in good_matches]).reshape(-1, 2)

    # Adjust points to the full image coordinates
    pts1 += np.array([bbox1[0], bbox1[1]])
    pts2 += np.array([bbox2[0], bbox2[1]])

    # Draw matches
    match_img = cv.drawMatches(roi1, kp1, roi2, kp2, good_matches, None, flags=cv.
    DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

    # Save the match image
    match_img_path = "/Users/sasindu/Desktop/DC/match_features.jpg"
    cv.imwrite(match_img_path, match_img)

    return pts1, pts2

img1 = cv.imread("/Users/sasindu/Desktop/DC/left.jpg", cv.IMREAD_GRAYSCALE)  # Left
    image
img2 = cv.imread("/Users/sasindu/Desktop/DC/right.jpg", cv.IMREAD_GRAYSCALE)  # Right
     image

pts1, pts2 = match_features_in_bboxes(img1, img2, bbox1, bbox2)

R = np.array([[ 1, 0, 0],
 [ 0,  1, 0],
 [0,  0,  1]])

T = np.array([[-60],
 [ 0],
 [ 0]])

mtx = np.array([[4.06850113e+03, 0.00000000e+00, 2.11799463e+03],
 [0.00000000e+00, 4.07285720e+03, 2.84485951e+03],
 [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])

P1 = np.hstack((mtx, np.zeros((3, 1))))  # Projection matrix for the first camera
P2 = mtx @ np.hstack((R, T.reshape(-1, 1)))  # Projection matrix for the second
    Position

pts1 = pts1.T
pts2 = pts2.T

# Triangulate points
points4D = cv.triangulatePoints(P1, P2, pts1, pts2)
points3D = points4D[:3] / points4D[3]
depths = points3D[2, :]  # Extract Z (depth information)

```

```python
#Printing Depth
print("Depth :", np.mean(depths))
```

Listing 7: finalcode.py

1.3.4 Stepper Motor Integration

- Objective: Rotate the camera to two positions using a stepper motor controlled by an ATmega328P microcontroller.

- Implementation

  1. Stepper Motor Control

     – Used an ATmega microcontroller to control the stepper motor.

```

#include <avr/io.h>
#include <util/delay.h>

#define F_CPU 16000000UL  // Define CPU frequency for delay

void init_timer1() {
    // Set timer1 in Fast PWM mode with ICR1 as TOP
    TCCR1A |= (1 << WGM11) | (1 << COM1A1);  // Fast PWM mode, Clear OC1A on compare match
    TCCR1B |= (1 << WGM12) | (1 << WGM13) | (1 << CS11);  // Fast PWM mode with prescaler 8
    ICR1 = 39999;  // Set TOP value for 20ms period (50Hz)
}

void set_servo_angle(uint8_t angle) {
    OCR1A = 1000 + (angle * 1000 / 180);  // Calculate and set duty cycle for the desired angle
}

int main(void) {
    // Set PD3 as input (indicator pin)
    DDRD &= ~(1 << PD3);

    // Set PD5 as output (servo control signal)
    DDRD |= (1 << PD5);

    // Initialize Timer1
    init_timer1();

    while (1) {
        // Check the state of PD3
        if (PIND & (1 << PD3)) {
            set_servo_angle(180); // Rotate servo to 180 degrees if PD3 is HIGH
        } else {
            set_servo_angle(0);   // Rotate servo to 0 degrees if PD3 is LOW
        }

        _delay_ms(50);  // Small delay to allow the servo to move
    }
```

```
38 }
```

Listing 8: code for stepper motor control

The ATmega328P microcontroller has built-in timers that can be used to generate PWM signals. Timer1 is a 16-bit timer that can be configured to operate in various modes, including Fast PWM mode.The code above initializes Timer1 in Fast PWM mode and controls the servo motor based on the input from a push button.

Explanation of the code:

- **Defining the CPU Frequency**:

  define F_CPU 16000000UL: This defines the clock frequency of the microcontroller as 16 MHz, necessary for delay calculations

- **Initializing Timer1**:

  init_timer1(): This function sets up Timer1 in Fast PWM mode with a prescaler of 8. The ICR1 register is set to 39999 to achieve a 20ms period (50Hz PWM frequency).

- **Setting the Servo** : set_servo_angle(uint8_t angle): This function calculates the duty cycle corresponding to the desired servo angle and sets the OCR1A register. For example, an angle of 0 degrees corresponds to a 1ms pulse, while 180 degrees corresponds to a 2ms pulse.

- **Main Function**: The main loop continuously checks the state of a push button connected to PD3. If the button is pressed (PD3 is HIGH), the servo is set to 180 degrees. If the button is not pressed (PD3 is LOW), the servo returns to 0 degrees. A small delay is added to allow the servo to move smoothly.

- **Results**
  - Successfully rotated the camera to capture two images from different positions.

1.3.5 Testing and Evaluation

- Objective: Ensure the robustness and accuracy of the implemented system through extensive testing and evaluation.

- Steps

  1. Calibration Testing
     - Tested the calibration process with various chessboard images under different lighting conditions.
     - Evaluated the reprojection error to ensure accurate calibration results.

  2. Object Detection Testing

- Trained the YOLOv8 model on a custom dataset and evaluated its performance using metrics such as precision, recall, and mean Average Precision (mAP).
- Fine-tuned the model by adjusting hyperparameters to improve detection accuracy.

3. Feature Matching Testing
   - Compared different feature detection algorithms (SIFT, ORB, SURF) to find the most suitable one for our application.
   - Evaluated the feature matching accuracy using various image pairs and analyzed the matching results.

4. Depth Calculation Testing
   - Verified the depth calculation results by comparing the triangulated depths with ground truth measurements.
   - Conducted experiments to test the system's performance under different conditions and refined the algorithms accordingly.

5. Motor Control Testing
   - Tested the stepper motor control mechanism to ensure precise rotation and image capture.
   - Evaluated the integration of motor control with the image capture process to achieve consistent results.

The depth camera project successfully combined various computer vision techniques, including camera calibration, object detection, feature matching, and depth calculation. By experimenting with different algorithms and models, we achieved accurate and reliable results.

In our testings, we managed to get the depth measurements withing an accuracy range of 5 milimeters for an object with an actual depth of around 30 centimeters.

The detailed coding process and integration of multiple components demonstrate the robustness and effectiveness of the implemented methods. The YOLOv8 model's advanced capabilities significantly enhanced our object detection performance, making it a key component of our system.

2 Daily Log Entries

2.1 Week 1: Initial Research and Planning

This week focused on foundational research, planning, and initial setup. We explored various depth measurement techniques and documented the key findings. This laid the groundwork for our conceptual designs and SolidWorks setup.

- **Day 1-3: Exploring Research Publications and Other Resources (March 1-3)**
  - Delve into various research papers and industry resources on depth camera technologies.
  - Understand different depth measurement techniques, including stereo vision, structured light, and time-of-flight (ToF) methods.
- **Day 4-5: Initial Findings Documentation (March 4-5)**
  - Document findings on depth measurement techniques, focusing on their pros and cons.
  - Create an outline of design requirements and features based on the research.
- **Day 6-7: Conceptual Design Brainstorming (March 6-7)**
  - Brainstorm and develop initial conceptual designs for the depth camera system.
  - Consider factors like environmental robustness, precision, and response time in the designs.

2.2 Week 2: Conceptual Design Development and Initial Coding

We focused on developing and evaluating conceptual designs, while simultaneously beginning the coding aspects by setting up the development environment and starting the calibration code.

- **Day 8-10: Developing Conceptual Designs (March 8-10)**
  - Continue developing conceptual designs for the depth camera system.
  - Explore methods to integrate deep learning models for depth estimation with robust hardware setups.
- **Day 11-12: Finalizing Conceptual Designs (March 11-12)**
  - Finalize the conceptual designs and prepare for evaluation.
  - Ensure all design requirements and features are well-documented.
- **Day 13-14: Camera Calibration Code Implementation (March 13-14)**
  - Implement and explain the camera_calibrate function.
  - Capture and save chessboard images for calibration.

2.3 Week 3: SolidWorks Enclosure Design and Calibration

This week involved setting up SolidWorks for the enclosure design and advancing the camera calibration code. We ensured proper integration between the physical design and the calibration process.

- **Day 15-17: Initial SolidWorks Setup (March 15-17)**
  - Set up the SolidWorks environment and project files.
  - Create initial sketches and basic shapes for the enclosure.
- **Day 18-19: Calibration Code Refinement (March 18-19)**
  - Continue running and refining the calibration code.
  - Document and analyze the intrinsic parameters obtained.
- **Day 20-21: Refining Enclosure Design (March 20-21)**
  - Review and refine the enclosure sketches based on feedback.
  - Ensure all dimensions are fully defined and accurate.

2.4 Week 4: PCB Design Initiation and Object Detection Setup

We began designing the PCB layout while setting up YOLOV8 for object detection. This ensured a balanced focus on both hardware and software components.

- **Day 22-24: PCB Design Research (March 22-24)**
  - Study components required for the PCB: Atmega 328p, A4988 stepper motor driver, USB Type-C adapter.
  - Document findings and key considerations for the PCB design.
- **Day 25-26: Initial PCB Schematic Design (March 25-26)**
  - Set up the design environment and project files in LTSpice.
  - Create initial circuit designs and basic layouts.
- **Day 27-28: Setting Up YOLOV8 (March 27-28)**
  - Introduce YOLOV8 and its capabilities.
  - Collect and preprocess the custom dataset using RoboFlow.

2.5 Week 5: SolidWorks and PCB Development

This week focused on refining the SolidWorks enclosure design and advancing the PCB schematic. Simultaneously, initial steps in object detection were taken.

- **Day 29-31: SolidWorks Enclosure Design Refinement (March 29-31)**
  - Start designing the main body of the enclosure.
  - Focus on ensuring a smooth and moldable shape.
- **Day 32-33: PCB Design Refinement (April 1-2)**
  - Make necessary adjustments to the schematic design based on simulation results.
  - Finalize the circuit design in LTSpice.
- **Day 34-35: Object Detection Model Training (April 3-4)**
  - Train the YOLOV8 model on the custom dataset.
  - Validate the trained model.

2.6 Week 6: Finalizing Enclosure and PCB Designs

We finalized the enclosure and PCB designs, ensuring they were ready for integration. The object detection code was implemented and tested.

- **Day 36-38: Enclosure and PCB Integration Planning (April 5-7)**
  - Discuss PCB size and layout with the team.
  - Ensure proper fit for the PCB and other components in the enclosure.
- **Day 39-40: Enclosure Design Refinement (April 8-9)**
  - Review and refine the cylindrical shape for the enclosure.
  - Incorporate feedback and make necessary adjustments.
- **Day 41-42: Object Detection Code Implementation (April 10-11)**
  - Implement the object detection code using YOLOV8.
  - Test the object detection and display bounding boxes.

2.7 Week 7: Feature Matching and Motor Holder Design

This week focused on developing the feature matching code and designing the motor holder in SolidWorks. We also planned the integration of the PCB and other components.

- **Day 43-45: Feature Matching Implementation (April 12-14)**
  - Implement the match_features_in_bboxes function.
  - Test feature matching on bounding boxes.
- **Day 46-47: Integrating Enclosure and PCB Designs (April 15-16)**
  - Plan for the integration of the PCB and other components.
  - Create mounting points and support structures for the PCB.
- **Day 48-49: Motor Holder Design (April 17-18)**
  - Research methods to securely hold a stepper motor.
  - Create initial sketches and models for the motor holder.

2.8 Week 8: Triangulation and Ensuring Ruggedness

We worked on implementing the triangulation code and researched methods to enhance the ruggedness of the enclosure. Final adjustments to designs were also made.

- **Day 50-52: Triangulation Code Implementation (April 19-21)**
  - Explain the triangulation process using the camera, rotation, and translation matrices.
  - Implement and test the triangulation code.
- **Day 53-54: Ruggedness Research (April 22-23)**
  - Research methods to increase the ruggedness of the enclosure.
  - Consider materials and design features that enhance durability.
- **Day 55-56: Final Adjustments to Designs (April 24-25)**
  - Review and refine the final designs for the enclosure and PCB.
  - Conduct thorough checks for any issues or areas for improvement.

2.9 Week 9: Stepper Motor Integration and Final Touches

We focused on integrating the stepper motor and making final touches to the design. This included finalizing the PCB layout and ensuring secure connections.

- **Day 57-59: Stepper Motor Setup and Integration (April 26-28)**
  - Demonstrate stepper motor control using an ATmega microcontroller.
  - Integrate the stepper motor control with photo capture.
- **Day 60-61: Finalizing PCB and Enclosure (April 29-30)**
  - Ensure all components and connections are secure in the PCB layout.
  - Finalize the enclosure design for manufacturing.
- **Day 62-63: System Integration Testing (May 1-2)**
  - Test the complete system and capture two photos from different positions.
  - Ensure proper integration of all components.

2.10 Week 10: Testing and Debugging

This week was dedicated to thorough testing and debugging of the complete system. We identified and addressed any remaining issues.

- **Day 64-66: Initial System Testing (May 3-5)**
  - Conduct initial tests on the complete system.
  - Analyze the test results and identify any issues.
- **Day 67-68: Debugging (May 6-7)**
  - Debug and refine the system based on test results.
  - Conduct additional tests to ensure accuracy.
- **Day 69-70: Final Adjustments (May 8-9)**
  - Make final adjustments to the system.
  - Ensure all components are functioning correctly.

2.11 Week 11: Documentation and Final Preparations

We focused on documenting the entire design process and preparing the final report. This included detailed drawings and specifications for manufacturing.

- **Day 71-73: Documentation (May 10-12)**
  - Document the entire design process, including calibration, object detection, feature matching, triangulation, and stepper motor integration.
  - Prepare a final report, highlighting key findings and challenges faced.
- **Day 74-75: Preparation for Manufacturing (May 13-14)**
  - Prepare detailed drawings and specifications for manufacturing.
  - Submit the design for manufacturing and plan for the next steps, including assembly and further testing.

3 Appendix

3.1 Previously used Arduino type Coding

```
#include <AccelStepper.h>

#define stepPin 2
#define dirPin 3
#define controlPin 8

AccelStepper stepper(1, stepPin, dirPin);

void setup() {
  // Set the pin modes
  pinMode(controlPin, OUTPUT);

  // Set the maximum speed and acceleration:
  stepper.setMaxSpeed(1000);
  stepper.setAcceleration(500);
}

void loop() {
  // Check the state of the input pin
  if (digitalRead(controlPin) == HIGH) {

    // Move the stepper motor 180 degree
    stepper.move(100);

    // Continuously run the stepper motor until it reaches the target
    while (stepper.distanceToGo() != 0) {
      stepper.run();
    }
    delay(3000);  //Wait for the camera to capture

    stepper.move(-100);  // Move back to the starting position

    // Continuously run the stepper motor until it reaches the starting position
    while (stepper.distanceToGo() != 0) {
      stepper.run();
    }

    digitalWrite(controlPin,LOW);
    delay(1000);
  }
}
```

Listing 9: Code for Stepper Motor Integration