Robot Localization Using Stereovision
BME 499 - Final Report

Prepared For:
The University of Waterloo

Prepared By:
Cooper Cole - 20815895

Date:
April 19th, 2024

Table of Contents

List of Figures

1 Introduction

1.1 Background

A baseball pitching robot is used for athlete training. The robot can move in the X and Y directions, and is capable of throwing any type of pitch at any location in the strike zone. See Figure 1.

Figure 1: Pitching Robot [1], [2]

1.2 Project Objective

The project objective is to design and build a system to localize the baseball pitching robot in the horizontal plane with relation to home plate using a stereovision camera. This allows the robot to aim the pitches into the strike-zone with increased accuracy. See Figure 2.

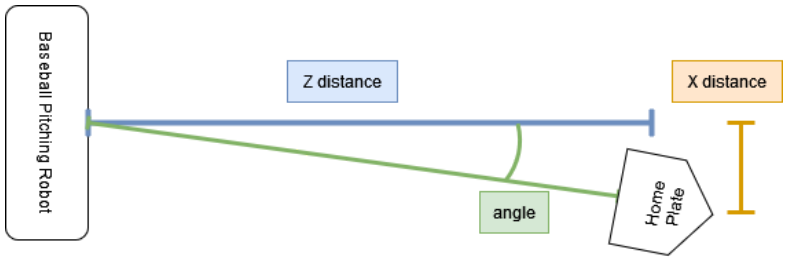


Figure 2: Environment Diagram

1.3 Needs Statement

To increase pitch accuracy and improve athlete training, there is a need to develop a precise localization system for the baseball pitching robot. Currently, the system lacks precise localization with respect to home plate, leading to a trial and error setup process for the robot.

1.4 System Requirements

The system must use stereovision to provide offline measurements of Z and X distance along with angle θ, interface seamlessly with a PC, and measure with a range of up to 20 meters.

1.5 System Component Options

The system will have a stereovision camera, a target to measure to, and custom written software to take the measurements. The options for each can be seen in Table 1 below.

Table 1: Design Options

Component	**Stereovision Camera**	**Target**	**Software**
Options	Basler	Home Plate	OpenCV in Python
	Intel RealSense	Aruco Marker	ROS
	ZED 2i	Reflective Markers	MATLAB

1.6 Chosen Options

The ZED 2i from Stereolabs was selected for its long range capabilities and simple hardware and software interfacing. The specifications include a range of up to 35 meters, USB C connectivity, and interfacing capabilities with Python and OpenCV. The Basler and Intel stereovision cameras have too short a range to use for this application. The target will use an Aruco marker for its visibility and easy identification capabilities. Home plate on its own is too hard to identify precisely and reflective markers are not visible enough at long distances. For the software interfacing, Stereolabs has an SDK to interface with the ZED 2i, and allows further interfacing using third party applications such as Python and OpenCV. See Figure 3.

Figure 3: ZED 2i Camera [3]

2 System Design

2.1 System Overview

The main components are the ZED 2i Stereovision camera and a localization target. Additional items include a baseball home plate to place the target on, a tripod for mounting the camera to mimic the pitching robot, and a laptop to run the stereo vision camera. The camera is placed at 15.7 m from the target, which is the distance the robot will be from home plate. See Figure 4.

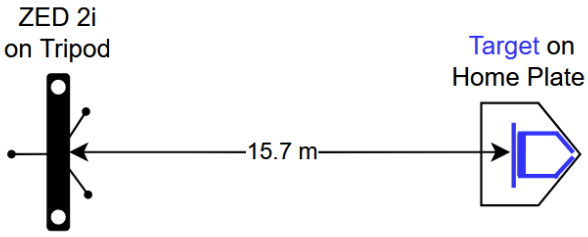


Figure 4: System Overview

2.2 Camera Parameters

The ZED 2i Stereovision camera is configured with specific parameters to optimize its performance for this application:

- **Frame Rate:** The camera is set to a frame rate of 15 frames per second. This relatively low frame rate improves the camera's performance in low light conditions, which is crucial for capturing clear images regardless of the lighting conditions in the training environment.
- **Resolution:** The camera's resolution is set to its maximum of 2K. This high resolution ensures that the camera captures the maximum amount of information in each image, which is vital for accurate depth perception and localization.
- **Distance Range:** The camera is configured to focus on a specific range of distances, from a minimum of 13.7m to a maximum of 16.7m. This range corresponds to the expected distance between the robot and the home plate during operation. By focusing on this specific range, the camera can optimize its depth perception and reduce errors in the Z measurements.
- **Target Surface:** The Aruco surface of the localization target is not "textureless" compared to a single coloured wall. This design choice reduces temporal instability, which can cause fluctuations in the camera's depth measurements over time. The textured surface of the target provides a consistent visual feature for the camera to lock onto, improving the stability and accuracy of the localization. [3]

2.3 Localization Target

The target is made from one square foot of 16 gauge cold rolled steel sheet metal. A locking hinge is used for the target to fold up for easy carrying and storage. The pattern is waterjet cut, the legs are bent and the hinge is riveted to the parts. The pop out from the handle is bent and riveted to the back of the legs to connect them. An 8”x8” Aruco marker is printed and taped to the front of the target. The target legs align to the back of home plate so it is square and an accurate point of reference. The sheet metal is $5 per square foot and the hinge is $5, bringing the total cost of the target to a very affordable $10.
See Figure 5.

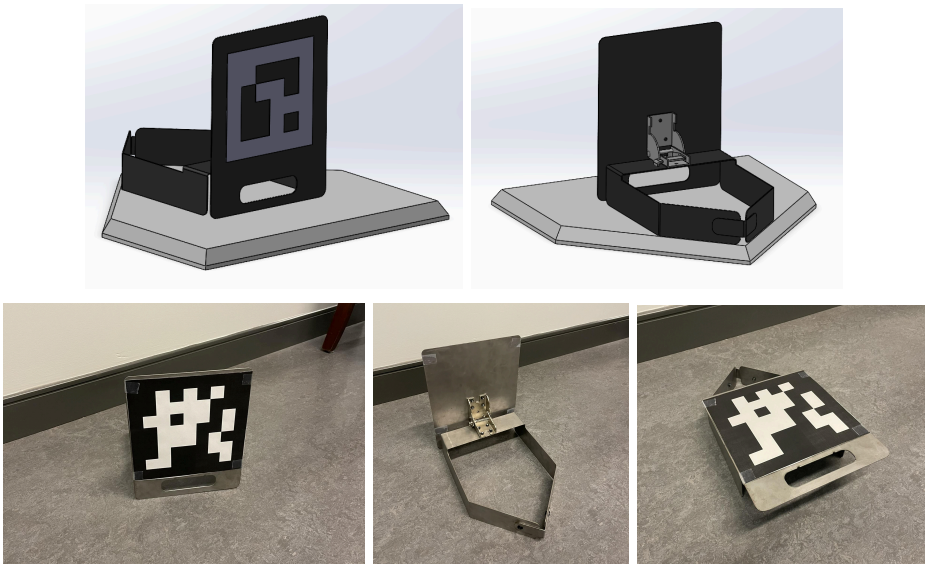
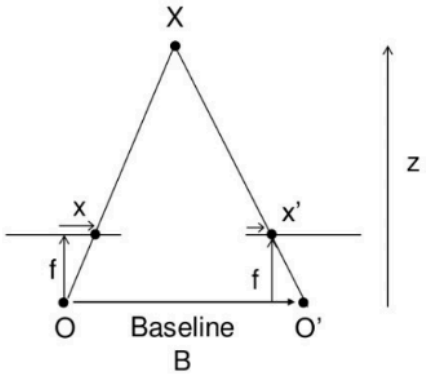


Figure 5: Localization Target

2.4 Localization Code

Stereovision uses a pair of cameras to capture multiple viewpoints of a scene, enabling the computation of depth information through triangulation. See Figure 6.



$$Z = \frac{fB}{x - x'}$$

Figure 6: Stereovision Calculation Diagram [4]

The left camera is the reference frame and the back of home plate is the desired point to measure to. The localization code is written in Python using OpenCV. The code automatically detects the Aruco marker on the target, then measures the depth distances to the center and right middle edge of the target to get the Z1 and Z2 measurements. Then uses the point cloud function call to get the lateral X distances to the center and middle edge. After making these required measurements, the code does the math indicated in Figure 7 to calculate the Z, X, and Yaw values. The code can be found in Appendix A.

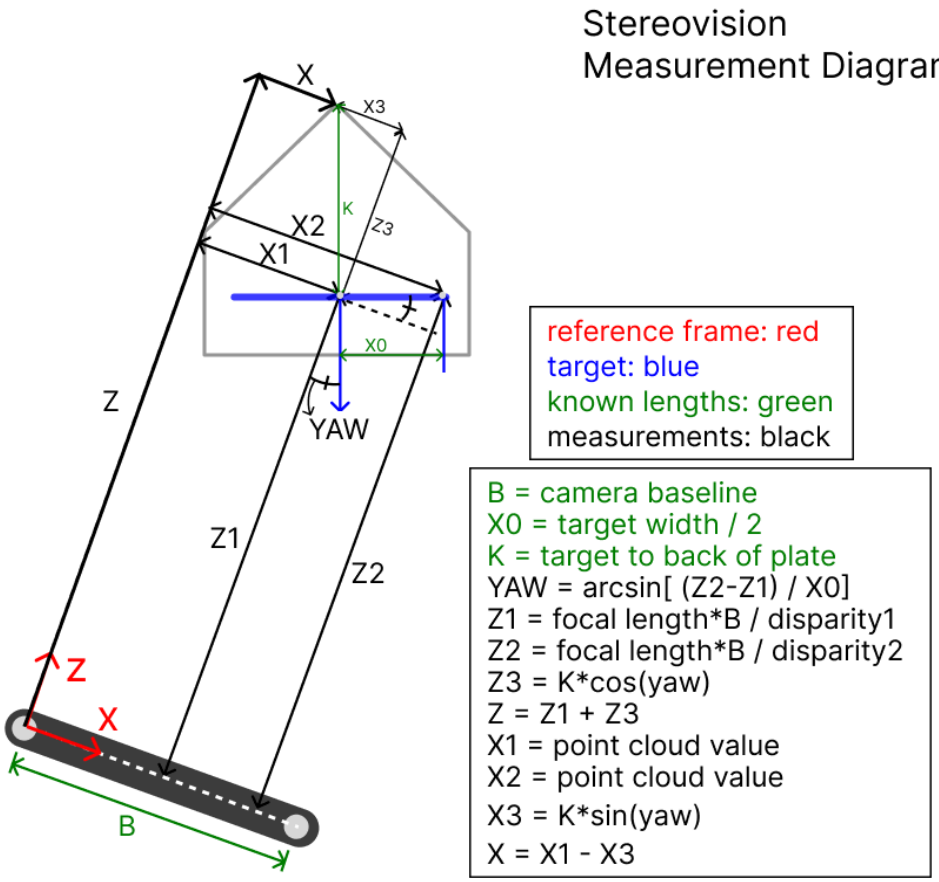


Figure 7: Measurement Diagram

3 Results

3.1 Data Collection

The ZED 2i camera was set up on the tripod, aligned with the center of the target, and pointed at the target, which was placed 51.5 ft away. Using a laser measurement tool, validation measurements were taken of the ground truth Z and X distances. For yaw, a compass was used to measure the relative angle between the camera and target. From here, 20 measurements were taken. See Appendix B for the data.

3.2 Analysis of Results

While there is no specific accuracy constraint for this project, the goal was to be as accurate as possible. Based on the expectations of this system, an acceptable limit is to have Z and X within 3” and the yaw within 5°. The results can be seen in Table 2.

Table 2: Results

	Z Error [in]	**X Error [in]**	**Yaw Error [°]**
Average	17.89”	2.26”	26.24
Median	18.67”	2.03”	18.44

The results reveal some interesting insights into the performance of the system. The X error is within an acceptable limit, which indicates that the system is able to accurately determine the lateral position of the robot relative to the home plate. This is very important, as it allows the robot to accurately aim its pitches across the width of the strike zone. However, the Z and Yaw errors are quite large, which shows that the system struggles to accurately determine the depth and orientation of the robot. This is a critical issue, as it affects the robot’s ability to accurately hit the strike zone and to adjust the pitches depending on its orientation to the home plate.

The poor accuracy of the Z measurement is due to the limitations of the stereovision camera. The ZED 2i camera, while boasting a range of up to 35 meters, has its limitations when it comes to depth perception. This is large in part due to the distance between the cameras (the baseline) and the focal length of the lenses.

The Yaw error, on the other hand, is a result of the inaccurate Z measurements. Since Yaw is calculated using Z1 and Z2 measurements, any error in these measurements will directly affect the accuracy of the Yaw calculation. This highlights the importance of accurate depth perception in determining the orientation of the robot.

4 Future Work and Conclusion

4.1 Future Work

4.1.1 Localization Accuracy

To increase the accuracy of the depth measurements, designing a custom stereovision setup using two identical cameras is the best way forward. The cameras can be set up with a much larger baseline and long focal length lenses will increase the accuracy of the depth measurements, as per the equation in Figure 8 from [5]. Appropriate baseline and focal length values can be chosen based on the desired maximum error.

$$\Delta z = \frac{z^2}{b \cdot f} \cdot \Delta D \quad \text{Where}$$

Δz	depth error
z	depth of point of interest
b	baseline
f	focal length
ΔD	disparity error

Figure 8: Depth Error Formula

4.1.2 Localization Target

For a future target, it should have larger dimensions of at least 12”x12”, and a checkerboard instead of an Aruco marker should be painted onto the target instead of printed paper. Using paint provides a more professional target with improved contrast, increased flatness, and reduced glare with the matte finish. A custom stereovision camera setup requires calibration, and the checkerboard can be used for both calibration and localization. The larger target allows for a larger checkerboard which is easier for the camera to identify for localization and improves the calibration performance.

4.2 Conclusion

In conclusion, this project has made significant strides towards achieving its goal of localizing a baseball pitching robot relative to home plate. Despite the challenges encountered, the system was able to measure the Z, X, and Yaw of the pitching robot with reasonable accuracy. However, the results also highlight the limitations of the current system and provide valuable insights into areas for future improvement.

Overall, while there is still work to be done, the project represents a significant step forward in the quest for a precise localization system for a baseball pitching robot. This project lays the groundwork and provides a clear way forward to improve the performance of the system.

References

[1] "Trajekt Sports," Trajekt Sports, 2022. [Online]. Available: https://www.trajektsports.com/. [Accessed: April 19, 2024].

[2] Hofstetter, F. (2020, April 13). Something feels off about the official MLB strike zone dimensions. Screwball Times. [Online]. Available: https://www.screwballtimes.com/texas-leaguers/mlb-strike-zone-dimensions/ [Accessed: April 19, 2024]

[3] Stereolabs. (2024). ZED 2 - Stereolabs. [Online]. Available: https://www.stereolabs.com/products/zed-2. [Accessed: April 4, 2024]

[4] OpenCV. "Depth Map from Stereo Images." OpenCV Documentation. [Online]. Available: https://docs.opencv.org/3.4/dd/d53/tutorial_py_depthmap.html. [Accessed: February 19, 2024]

[5] "Estimate Disparity Error for Depth Accuracy Estimation," Robotics Stack Exchange. [Online]. Available: https://robotics.stackexchange.com/questions/11749/estimate-disparity-error-for-depth-accuracy-estimation. [Accessed: March 25, 2024].

Appendix A

```
import pyzed.sl as sl
import math
import numpy as np
import sys
import cv2
import cv2.aruco as aruco
import fractions
import csv
import os
import datetime


MILLIMETERS_TO_INCHES = 0.0393701
FRACTIONAL_PRECISION = 16
ZED2_BASELINE = 120  # mm
MARKER_WIDTH = 300  # mm
k = 293 # mm
PRINT_DATA = False
WRITE_TO_CSV = True
SHOW_IMAGE = True
PROCESS_IMAGE = True
SET_MIN_DIST = False
# Define the checkerboard dimensions
CHECKERBOARD_SIZE = (3, 3)


def getMousePos(image):
    def onMouse(event, x, y, flags, param):
        if event == cv2.EVENT_LBUTTONDOWN:
            param['x'] = x
            param['y'] = y
            param['event'] = event



    y_start = 600
    y_end = 800
    x_start = 1000
    x_end = 1200
```

```
    param = {'x': -1, 'y': -1, 'event': -1}
    cv2.namedWindow("Image")
    cv2.setMouseCallback("Image", onMouse, param)
    cv2.imshow("Image", image[y_start:y_end, x_start:x_end])

    while param['event'] != cv2.EVENT_LBUTTONDOWN:
        cv2.waitKey(10)

    cv2.destroyWindow("Image")
    print(f"Chosen point: {{{param['x'] + x_start};{param['y'] + y_start}}}")

    return param['x'] + x_start, param['y'] + y_start

def mm_to_feet_inches_fractions(mm):
    # Convert mm to inches
    inches = mm * MILLIMETERS_TO_INCHES
    # Calculate feet and remaining inches
    feet = int(inches // 12)
    remaining_inches = inches % 12

    # Convert remaining inches to fractions
    remaining_distance = remaining_inches - int(remaining_inches)

    numerator = round((remaining_distance * FRACTIONAL_PRECISION))
    denominator = FRACTIONAL_PRECISION
    fraction = fractions.Fraction(numerator, denominator)

    # Format the result
    result = f"{feet}' {int(remaining_inches)}\" {fraction}"
    return result

def get_camera_matrix_and_distortion(camera):
    # Get the camera matrix and distortion coefficients from the ZED camera
    camera_info = camera.get_camera_information()
    # Get the camera matrix and distortion coefficients from the ZED camera
    cam_fx =
camera_info.camera_configuration.calibration_parameters.left_cam.fx
    cam_fy =
camera_info.camera_configuration.calibration_parameters.left_cam.fy
    cam_cx =
camera_info.camera_configuration.calibration_parameters.left_cam.cx
```

```
    cam_cy =
camera_info.camera_configuration.calibration_parameters.left_cam.cy
    cameraMatrix = np.array([[cam_fx, 0, cam_cx],
                             [0, cam_fy, cam_cy],
                             [0, 0, 1]])
    #  Get the distortion coefficients
    distCoeffs =
camera_info.camera_configuration.calibration_parameters.left_cam.disto
    # # Convert to numpy array, come back to check this array, it apprears to
cut off the last digit, expected values are [0.01, 0.1, 0.1, 0.1, 0.1]ish
    # distCoeffs =
np.array(camera_info.camera_configuration.calibration_parameters.left_cam.disto
)

    # print the camera matrix and distortion coefficients
    if(PRINT_DATA):
        print(f"Camera Matrix: {cameraMatrix}")
        print(f"Distortion Coefficients: {distCoeffs}")

    return cameraMatrix, distCoeffs

def find_aruco_marker(image, aruco_dict, parameters, cameraMatrix, distCoeffs):

    # Crop the image to the region of interest
    y_start = 625
    y_end = 725
    x_start = 1025
    x_end = 1125

    # Crop the image and convert it to grayscale
    gray = cv2.cvtColor(image[y_start:y_end, x_start:x_end],
cv2.COLOR_BGR2GRAY)

    processed_image = gray
    cv2.imshow("Processed Image", processed_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Detect the aruco markers in the cropped image, this also applies adaptive
thresholding
    corners, ids, rejected = aruco.detectMarkers(processed_image, aruco_dict,
parameters=parameters)
```

```
    # If any markers are detected
    if ids is not None:
        # Get the center coordinates of the first detected marker
        marker_center = np.mean(corners[0][0], axis=0)
        marker_right_middle = np.mean(corners[0][0][1:3], axis=0)
        print(f"Center of the aruco marker: {marker_center}")
        print(f"Right Middle of the aruco marker: {marker_right_middle}")
        x = int(marker_center[0]) + x_start # add on the x_start to get the
coordinates in the uncropped image
        y = int(marker_center[1]) + y_start # add on the y_start to get the
coordinates in the uncropped image
        x_right_middle = int(marker_right_middle[0]) + x_start # add on the
x_start to get the coordinates in the uncropped image

        if PRINT_DATA: print(f"Center of the aruco marker: {{{x};{y}}}")

        # Draw a rectangle around the detected marker in the image
        cv2.rectangle(image, (x-10, y-10), (x+10, y+10), (0, 255, 0), 2)

        # Display the image with the detected marker
        cv2.imshow("Raw Image with Detected Center", image[y-200:y+200,
x-200:x+200])
        cv2.waitKey(0)
        cv2.destroyAllWindows()

        # return the coordinates of the center of the aruco marker for depth
sensing, and yaw angle
        return x, y, x_right_middle
    else:
        print("---------NO ARUCO MARKER DETECTED---------")
        return None

def get_z_depth(depth, x, y):
    # Get the depth value at the detected point
    err, z = depth.get_value(x, y)
    return z

def get_x_distance(point_cloud, x, y):
    err, point_cloud_value = point_cloud.get_value(x, y)
```

```
    # get the lateral distance to the target using the point cloud value of the
center of the target
    if err == sl.ERROR_CODE.SUCCESS:
        x_distance = point_cloud_value[0]
        return round(x_distance, 2)

    return None

def get_yaw(z1, z2):
    x0 = MARKER_WIDTH / 2
    # print(f"X0: {x0}")
    # print(f"Z1: {z1}")
    # print(f"Z2: {z2}")
    # print(f"Z2-Z1: {z2-z1}")
    # print(f"Z2-Z1/x0: {(z2-z1)/x0}")
    yaw = math.asin( (z2-z1) / x0)
    yaw_degrees = np.round(math.degrees(yaw),2)
    return yaw_degrees

def write_to_csv(data):
    # Write the data to a CSV file
    if WRITE_TO_CSV:
        if not os.path.isfile('output.csv'):
            with open('output.csv', mode='a') as output_file:
                output_writer = csv.writer(output_file, delimiter=',',
quotechar='"', quoting=csv.QUOTE_MINIMAL)
                output_writer.writerow(['Z [IMP]', 'Z [mm]', 'X [IMP]', 'X
[mm]', 'Yaw [deg]'])
        with open('output.csv', mode='a') as output_file:
            output_writer = csv.writer(output_file, delimiter=',',
quotechar='"', quoting=csv.QUOTE_MINIMAL)
            output_writer.writerow(data)

def main():

    # Create the aruco dictionary, this is for the specfic aruco marker we are
using
    aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_7X7_250)
    parameters = aruco.DetectorParameters()

    # Create a Camera object
    zed = sl.Camera()
```

```
    # Create a InitParameters object and set configuration parameters
    init_params = sl.InitParameters()
    init_params.depth_mode = sl.DEPTH_MODE.ULTRA  # Use ULTRA depth mode
    init_params.camera_resolution = sl.RESOLUTION.HD2K # Use HD2K video mode
(4416x1242) @ 15fps
    # init_params.camera_resolution = sl.RESOLUTION.HD1080 # Use HD1080 video
mode (1920x1080) @ 15fps
    init_params.camera_fps = 15 # Set the camera to 15fps, I think this is
redundant as 15fps is max for HD2K
    init_params.coordinate_units = sl.UNIT.MILLIMETER  # Use meter units (for
depth measurements)
    init_params.depth_maximum_distance = 16764 # Set the maximum depth
perception distance to 55 ft
    init_params.depth_minimum_distance = 13716 # Set the minimum depth
perception distance to 45 ft
    # Set the positional tracking parameters
    track_params = sl.PositionalTrackingParameters()
    track_params.set_as_static = True # Set the camera to static mode
    # PositionalTrackingParameters::set_as_static

    # Create a matrix to store image, depth, point cloud
    image = sl.Mat()
    depth = sl.Mat()
    point_cloud = sl.Mat()

    # Create a reference to the mirror (IDK WHAT THIS MEANS)
    mirror_ref = sl.Transform()
    mirror_ref.set_translation(sl.Translation(2.75,4.0,0))

    # Open the camera
    status = zed.open(init_params)
    if status != sl.ERROR_CODE.SUCCESS: #Ensure the camera has opened
succesfully
        print("Camera Open : "+repr(status)+". Exit program.")
        exit()

    # Get the camera matrix and distortion coefficients
    cameraMatrix, distCoeffs = get_camera_matrix_and_distortion(zed)

    # Create and set RuntimeParameters after opening the camera
    runtime_parameters = sl.RuntimeParameters()
    # A new image is available if grab() returns SUCCESS
```

```
    if zed.grab(runtime_parameters) == sl.ERROR_CODE.SUCCESS:
        # Retrieve left image
        zed.retrieve_image(image, sl.VIEW.LEFT)
        # Retrieve depth map. Depth is aligned on the left image
        zed.retrieve_measure(depth, sl.MEASURE.DEPTH)
        # Retrieve colored point cloud. Point cloud is aligned on the left
image.
        zed.retrieve_measure(point_cloud, sl.MEASURE.XYZRGBA)

        # define the image data for future use
        image_data = image.get_data()

        # detect center of the aruco marker
        center_x, center_y, center_x_right = find_aruco_marker(image_data,
aruco_dict, parameters, cameraMatrix, distCoeffs)

        print(f"----------------DEPTH-------------------------")
        z1 = get_z_depth(depth, center_x, center_y)
        z2 = get_z_depth(depth, center_x_right, center_y)
        z1_inches = mm_to_feet_inches_fractions(z1)
        z2_inches = mm_to_feet_inches_fractions(z2)
        print(f"Z Depth to Target: {z1_inches}, {round(z1, 2)} mm")
        print(f"Z Depth to Target Right: {z2_inches}, {round(z2, 2)} mm")
        print(f"----------------X Distance-------------------------")
        x1 = get_x_distance(point_cloud, center_x, center_y)
        x2 = get_x_distance(point_cloud, center_x_right, center_y)
        x1_inches = mm_to_feet_inches_fractions(x1)
        x2_inches = mm_to_feet_inches_fractions(x2)
        print(f"X Distance to Target: {x1_inches}, {round(x1, 2)} mm")
        print(f"X Distance to Target Right: {x2_inches}, {round(x2, 2)} mm")
        print(f"----------------YAW-------------------------")
        yaw = get_yaw(z1, z2)
        print(f"Yaw: {yaw}°")

        print(f"----------------Final Values-------------------------")
        z3 = k * math.cos(math.radians(yaw))
        x3 = k * math.sin(math.radians(yaw))
        Z = np.round(z1 + z3, 2)
        X = np.round(x1 - x3, 2)
        print(f"Z: {Z} mm, {mm_to_feet_inches_fractions(Z)}")
        print(f"X: {X} mm, {mm_to_feet_inches_fractions(X)}")
        print(f"Yaw: {yaw}°")
```

```
        print(f"----------------END-----------------------")

        data = [Z, X, yaw]
        write_to_csv(data)

    # Close the camera
    zed.close()

if __name__ == "__main__":
    main()
```

Appendix B

Z [mm]	X [mm]	Yaw [deg]		Z diff [mm]	X diff [mm]	Yaw diff [deg]		Z Val [mm]	X Val [mm]	Yaw Val [deg]	
16162.42	-272.6	17.09		465.23	-42.41	37.09		15697.19	-230.19	-20	
16136.24	-184.71	-0.26		439.05	45.48	19.74					
16225.03	66.44	-60.6		527.84	296.63	-40.6					
16092.45	-300.24	22.99		395.26	-70.05	42.99		**avg Z diff**	**med Z diff**	**stdev Z diff**	
16214.46	-172.7	-2.79		517.27	57.49	17.21		454.52	474.32	82.88	**[mm]**
16222.29	-176.91	-1.97		525.10	53.28	18.03		17.89	18.67		**[in]**
16236.9	-88.69	-19.68		539.71	141.50	0.32					
16224.26	-104.24	-16.45		527.07	125.95	3.55		**avg X diff**	**med X diff**	**stdev X diff**	
16089.99	-340.71	31.87		392.80	-110.52	51.87		110.52	51.53	136.51	**[mm]**
16085.33	-329.14	29.27		388.14	-98.95	49.27		4.35	2.03		**[in]**
16067.6	-352.46	34.67		370.41	-122.27	54.67					
16160.13	-316.63	26.33		462.94	-86.44	46.33		**avg Yaw diff**	**med Yaw diff**	**stdev Yaw diff**	
16012.79	-418.16	52.43		315.60	-187.97	72.43		26.24	18.44	31.05	
16204.15	-105.18	-16.21		506.96	125.01	3.79					
16171.51	-89.54	-19.34		474.32	140.65	0.66					
15943.82	-466.2	72.9		246.63	-236.01	92.9					
16206.38	-102.91	-16.68		509.19	127.28	3.32					
16239.34	-118.99	-13.49		542.15	111.20	6.51					
16187.47	-178.66	-1.56		490.28	51.53	18.44					