Final Report for Introduction to Robotics Lab

Huzaifah Tariq Ahmed*, Daniyal Rahim †, Syed Asghar Abbas Zaidi ‡
Habib University, Pakistan
Email: *ha07151@st.habib.edu.pk, †d07605@st.habib.edu.pk, ‡sz0701@st.habib.edu.pk

***Abstract*—The Aim of the this course's Lab was to build a 4R robotics arm from scratch so that it could successfully perform pick and place task autonomously using the camera. This task presents challenges across industries such as medical, automotive, power generation, and manufacturing, where it is carried out repetitively thousands of times throughout the day. Robotic arms in the medical industry aid in surgeries and pharmaceutical assembly, while in automotive manufacturing, they excel in assembly line tasks like placing components and welding. Additionally, they play a vital role in the power industry for maintenance and inspection tasks, ensuring safety and efficiency. We approached the problem discretely by breaking it into sub problems like of forward kinematics, inverse kinematics, perception, motion control and lastly integration of all units. At the end, we were able to pick and place the wooden blocks within any point of robot's workspace using intel Real Sense Camera and hence successfully tackled the problem.**

***Index Terms*—Phantom X Pincher, Robot Manipulator, Pick and Place, Inverse Kinematics, Forward Kinematics, Perception, Vision-based system, Pose Estimation, Motion Control**

I. Introduction

THIS paper presents a comprehensive study on the development of a vision-based pick-and-place robotic system on Phantom X Pincher arm with the help of Matlab. The team explored the camera used, developed an interpretation of the robotic arm system in terms of frames, and designed a vision pipeline for object pose estimation using depth images. The project also involved understanding point clouds as an alternate representation for scenes in robotics and developing a vision pipeline for pose estimation using point clouds. The team obtained forward kinematics mapping of the manipulator using the standard DH convention and determined the reachable workspace of the manipulator. They also obtained closed-form expressions for the inverse kinematics mapping of the manipulator.

The culmination of the project was the development of a functional motion control system that could accurately pick and place objects at specified locations. The team integrated different subsystems to build this complex robotic system, formulated a grasping strategy considering the abilities of the arm gripper, and designed experiments to determine the performance of the robotic system.

The report concludes with the integration of the motion control system and the vision pipeline to obtain a complete vision-based pick-and-place robotic system. All of our documentations have also been uploaded to github [1]

A. Significance

The development of a vision-based pick-and-place robotic system represents alot of potential in various industrial settings (automation), particularly assembly lines where precision and efficiency are paramount. Moreover, the project's innovative vision pipeline, leveraging depth images and point clouds for object pose estimation, promises to enhance robotic perception in tasks such as sorting, inspection, and assembly. Additionally, the project's forward and inverse kinematics mappings for the Phantom X Pincher arm offer valuable insights into robotic manipulator design and control, with computational efficiency crucial for real-time applications. Furthermore, the determination of the manipulator's reachable workspace provides essential insights for task planning and optimization in robotics. The successful integration of motion control and vision systems exemplifies a holistic approach to robotic system development, yielding *robots capable of handling complex tasks with heightened efficiency and precision.* Moreover, the formulated grasping strategy lays the groundwork for adaptive gripping mechanisms, crucial for versatile and reliable object manipulation. Lastly, the designed experiments for evaluating system performance establish a standardized framework for benchmarking efficiency and accuracy, driving advancements in the field. Together, these contributions highlight the project's potential to drive innovations and advancements in robotics, shaping the future of automation and intelligent systems.

II. Developmental Details

We didn't knew how to develop vision-based pick-and-place robotic system on Phantom X Pincher arm from the get-go. Instead we slowly made progress on learning theory and implemented each subsystem seperately throughout each week until we integrated everything together those in Lab 7 and 8 respectively. We will be explaining what aspect we worked on first, what did we achieve. The performance of our system will be discussed in III. section

A. Getting Familiar with Matlab

As MATLAB was used extensively throughout the course, we were expected to familiarize ourselves with its' environment. As we all were computer engineering students, we already had some kind of familiarity, but just to be safe, Asghar undertook Object-Oriented Programming Onramp [3], Huzaifah undertook Simulink Onramp [4], Daniyal undertook Introduction to Symbolic Math with MATLAB [5] to broaden their skill-set to use throughout the course. Daniyal's skills really came in-handy in mathematically modelling kinematics on Matlab.

B. Getting Familiar with arm hardware

We identified kinematic chain, and DOFs (Degree of Freedoms) of our arm using Gruber's Formula. Understood how

some tasks are better suited in different co-ordinate systems (Chess in Cartesian and Painting in Cylindrical) and calibrated Armlink's software with actual hand's movement, and did pick/drop giving commands manually through GUI (without any automation). Familiarized ourselves with Dynamixel's reference manual, identifying actuators and sensors as well as their limitatations in terms of resolution and joint limits. We also designed block-diagram for what a automated pick-and-place operation using perception could look like.

All of these activities were important in understanding the hardware we are working with, what limitations we can expect and how it normally moves when we eventually start developing the model for automation on Mathlab

C. Perception

a) Theory: We worked with Intel RealSense SR305 which gave us Stereo images allowing us to measure the third-dimension (depth), further explanation can be found at [6]. Worked first with depth-image (before point-cloud) and understood how various lightening conditions introduces varying degrees of noises, and how could potential post-processing could help mitigate those (but also introduce delays). We learnt Segmentation, Colour Segmentation, K-cluster and learnt to identify Areas of Interest by setting Threshold manually (adaptive filter can be used to automate it) and understood the importance of line and edge-detection using Hough transform and RANSAC (random sample consensus) to determine the pose of detected objects.

Later on, we delved into the concept of point clouds, which serve as an alternative representation for scenes in robotics. we leveraged the capabilities of an RGB-D camera, which provides depth images where each pixel value corresponds to the distance between the camera and the nearest object. By combining this depth information with the camera's intrinsic parameters, such as focal length, we transformed the 2D depth image into a collection of 3D points known as a point cloud. Each point in this cloud is defined by its coordinates (x, y, z) in real-world space, allowing for a detailed representation of the environment for segmentation and pose estimation.

b) Implementation: When working with depth-images, we converted RGB images to binary format, inverted colors to isolate cubes, and then removed unwanted objects. Color identification was done based on RGB values, and upper faces of cubes were identified using depth data. We then refined the process to isolate top faces and determine their orientation. This involved creating binary masks for each block color, combining them, and using depth analysis to isolate top faces. Finally, we visualized the results, including identified top faces and their orientation. We learnt to do this by taking relevant modules from Image Processing with Matlab's course [7]

For point-cloud, we developed a vision pipeline for pose estimation using point clouds. we did this by extracting point clouds from the scene, followed by segmenting these clouds to isolate objects of interest, such as cubes in a workspace. Leveraging geometric models and point cloud registration techniques, we then estimated the pose of these objects relative to the camera frame. This involved fitting geometric models to point clouds and finding transformations that align model point clouds with scene point clouds. Through meticulous experimentation and coding, **we team crafted a robust pipeline capable of accurately determining the position and orientation of objects in the robot's environment.**

D. Forward Kinematics

In this, our primary goal was to understand and implement forward kinematics for a robotic manipulator We began by delving into the Denavit-Hartenberg (DH) convention, which provided a systematic framework for assigning frames and parameters to the manipulator's links and joints [2]. Utilizing DH parameters and MATLAB functions, we constructed the forward kinematics model to compute the end-effector position and orientation based on the joint angles. This involved developing MATLAB scripts to calculate homogeneous transformation matrices, mapping servo angles to DH joint angles, and visualizing the robot model. Additionally, we explored the reachable workspace of the manipulator through various simulations and verified our computations through physical testing.

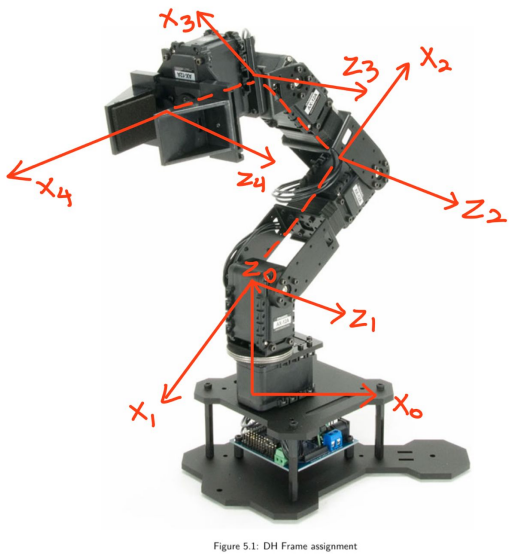

Figure 1: DH Frame Assignment

Link	a_i	α_i	d_i	θ_i
1	0	0	L1 (0.04495m)	θ_0
2	L2 (0.1035m)	0	0	θ_1
3	L3 (0.10375m)	0	0	θ_2
4	L4 (0.111m)	0	0	θ_3

TABLE I: DH Parameters

TABLE II: Linear mapping between servo angles and DH angles

Joint ID	DH Joint Angle θ_i	Servo angle Φ_i	Aligned Direction of rotation (Yes/No)
1	$0°$	$0°$	Yes
2	$0°$	$-90°$	Yes
3	$0°$	$0°$	Yes
4	$0°$	$0°$	Yes

TABLE III: Linear mapping between servo angles and DH angles

Joint ID	DH Joint Angle θ_i	Servo angle Φ_i	Aligned Direction of rotation (Yes/No)
1	$0°$	$0°$	Yes
2	$0°$	$-90°$	Yes
3	$0°$	$0°$	Yes
4	$0°$	$0°$	Yes

We later further inputted maximum angle every servo should ideally achieve, and through trial and error, we got "check

TABLE IV: Expected Joint Limits

Joint ID	Minimum Joint Angle		Maximum Joint Angle	
	Servo angle	DH joint angle	Servo angle	DH Joint angle
1	$-150°$	$-150°$	$150°$	$150°$
2	$-150°$	$-240°$	$150°$	$60°$
3	$-150°$	$-150°$	$150°$	$150°$
4	$-150°$	$-150°$	$150°$	$150°$

E. Inverse Kinematics

We now have to determine the joint variables of robot's while giving it the position of end effector. We studied solution of Inverse Kinematics equations of 2R robot in class. But, in our case, robot has 3 revolute joints whose z axis are parallel to each other. Therefore, we first find the equations to trace back from 4th revolute joint of robot to 3rd revolute joint.

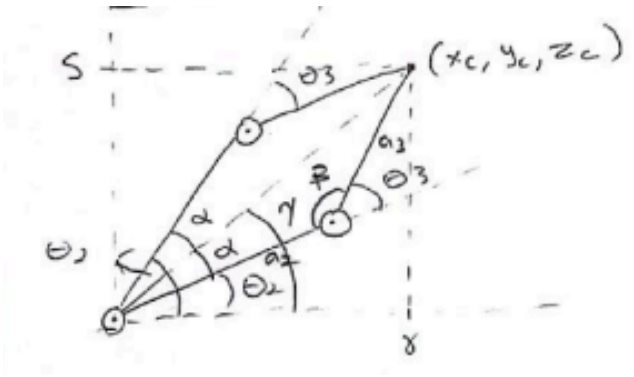

Figure 2: 3R robot image where xc,yc,zc are obtained by tracng back from last joint

ϕ will be input of the function.

$$
\begin{aligned}
t1 &= \text{atan2}(y, x);\\
xc &= x - (0.111\cos(\phi)\cdot\cos(t1));\\
yc &= y - (0.111\cos(\phi)\cdot\sin(t1));\\
s &= z - d1 - 0.111\sin(\phi);\\
r &= \sqrt{xc^2 + yc^2};
\end{aligned}
$$

Equations for $\theta_1, \theta_2, \theta_3, \theta_4$ were obtained in class slides.

We found over all 4 solutions to our IK equations namely 2 for elbow up and 2 for elbow down,

The current x, y, z coordinate of end effector can be achieved via getpose function and desired coordinates can be achieved by finding the inverse kinematics solution closest to current configuration of robot. i.e.

$$|\Delta\theta_1| + |\Delta\theta_2| + |\Delta\theta_3| + |\Delta\theta_4|$$

We also have to check whether the obtained joint variables are possible to attain in robot's workspace. Therefore we write a function ErrorCode (like in 7) to verify that the solution is within the joint limits.

F. Motion Control

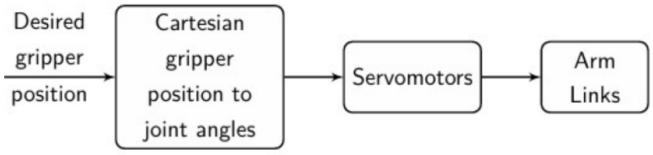

Figure 3: Block Diagram of our Motion Control System

This block diagram indicates that our system is not operating in a closed loop, i.e. there is no feedback being obtained from the camera about the positioning of the gripper.

We input the desired position to the system. Through which it calculates the joint variables such the gripper should be placed on desired coordinates. Servo motors attached to the joints then move Arm Links at the calculated angle.

There is no path planning or trajecgory planning algorithm specifically attached to our system. We just calculate the desired joint angle and set the joint variables accordingly to pick and place the wooden blocks.

G. Final Integration with Perception

Figure 4: Block Diagram of our Complete System

In the end we have to integrate the perception/ vision in our Motion Control System. For that we first fetch RGB and depth image from Intel RealSense Camera. We then process the RGB image to identify the blocks of our desired color and find x,y coordinate from the image. To find the z coordinate we use same coordinates in depth image to find the depth of that point. Hence we get all x,y,z coordinates to obtain IK solutions for our robot. We then provide these parameters and ϕ to the motion control system to set the joints at such an angle so that end effector can pick the block and place it at desired location within workspace.

We have to do unique mapping from a pixel(u,v) in image to a point (x,y,z) in the real world. Therefore, we need to calculate (u,v) using camera parameters (f_x, f_y, u_0, v_0), we can use following model:

$u = f_x \cdot x/z + u_0$

$v = f_y \cdot y/z + v_0$

All the systems of inverse kinematics, perception, motion control are part of this section. But, since we have described all parts individually in previous sections therefore It will be redundant to reiterate the them. Hence, instead, we have build the connection with previous parts.

III. Testing and Results

As we have gone through most of our design previously, we will now be talking about many hurdles that we had to go through. At times there were errors that told us that our system isn't working, while at times, those errors were correct.

A. Perception

We tested our Perception labs in various lightning conditions, realizing that it easily led to many false-positives and false-negatives due to how colours shift due to light colour itself or how it's hard to tell edges in less lighting. We realized the importance of consistent bright lighting especially by white light. We have later suggested in Section IV on how we could mitigate or lessen the error.

B. Forward Kinematics

1) Physical Joint Limits: As servos in pincher X have the capacity to rotate from $-150°$ to $150°$ you may expect Table IV when thinking about Joint limits. However, in actuality within our testing that wasn't the case. There were some *constraints* imposed. Either, the body of the arm itself didn't allow the joint to move in a certain angle, or servo couldn't rotate further cause it required arm to go *beneath* the defined workspace or Matlab wouldn't accept 150 degrees as a valid input, and only 149.

Whenever the robotic arm couldn't have his servo move any further, we would get **checksum fail** in our mathlab output and we knew that it failed that, after which we had to turn off and on the robot. That was cause there was a discrepancy between what the computer thinks "the robot should be at", and where "it's really at". Initially I thought the robotic arm itself was malfunctioning until I realized that it couldn't realize certain configurations due to our physical workspace limits.

TABLE V: Physical Joint Limits

Joint ID	Minimum Joint Angle		Maximum Joint Angle	
	Servo angle	DH joint angle	Servo angle	DH Joint angle
1	$-150°$	$-150°$	$149°$	$149°$
2	$-110°$	$-200°$	$150°$	$30°$
3	$-149°$	$-149°$	$148°$	$148°$
4	$-105.5°$	$-105.5°$	$103°$	$103°$

2) Identifying reachable Workspace: We plotted the position of the end-effector with 10,000 random configurations. One could notice the squished spherical shape of it which is **our workspace**. In reality though, it would be more flattened from beneath cause there is ground that needs to be considered as well. We have highlighted the maximum point at both ends of all three axies in the graph. **The maximum reach was 0.315m**.

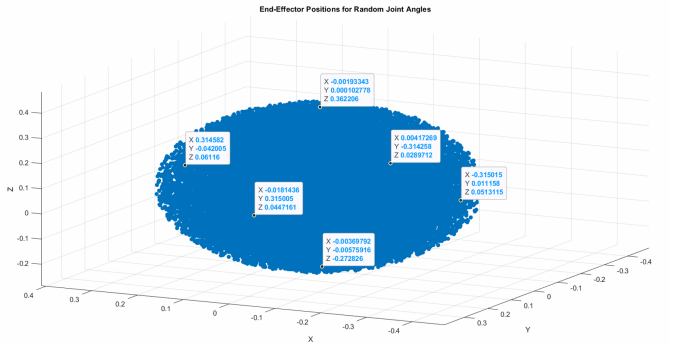

Figure 5: End-Effector Positions for Random Joint Angles

3) FindPincher: We positioned the arm such that $\theta_1 = \pi/2$ and the other angles were set to zero. We compared what we expected the end-effector position to be in, with what we actually got.

The experimental observation of the end-effector position yielded coordinates (0,0.314,-0.036) in metres which were as follows, when ideally it should have been Ideally, it should have been ($1.9845x10^{-7}$, 0.31825, 0.4495). The different in X,Y are near negligible but the differences do exist. We believe the inaccuracies in motor encoders or due to mechanical wear-and-tear can cause such issues. Within our scope, such a margin of error is acceptable.

However, what's interesting is 0.08m discrepancy in z-axis. As we had the position of our hand point straight-forward, we believe such a large error exist due to **gravity**, resulting in bending of mechanical parts. We have assumed the robot to be ideally rigid, when in reality it isn't.

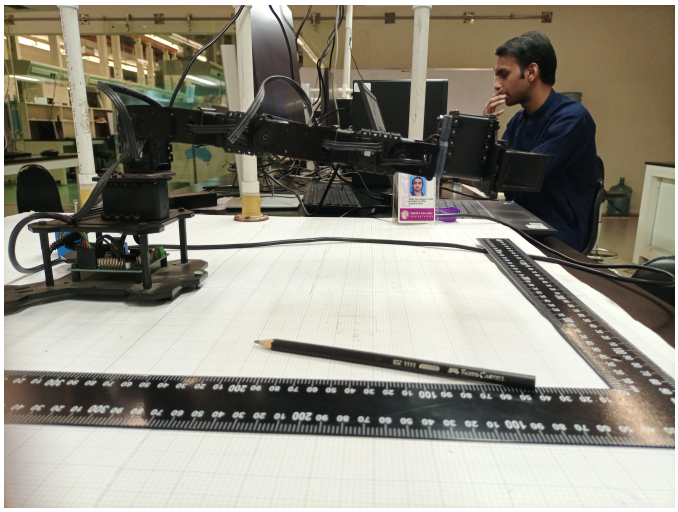

Figure 6: Gravity bending the arm

4) Verification of Forward Kinematics: We verified whether our Forward Kinematics was working or not by comparing pincherModel.m with PincherFK, we always used "random" to make sure that it verified with the most random of configurations. We did this 10 times, and verified in all of these, rotational matrixes were matching, thus verifying that it worked.

5) ErrorCode: We made sure that any expected abnormal inputs made the code exit if we knew it could potentially harm the robotic arm. We made use of it for debugging issues at times. We have attached one usage of it in our Figure.

When we give out-of-range values:

```
%Creating jointAngles array consisting of our DH angles.
DH_JointAngle1 = -151;
DH_JointAngle2 = -240;
DH_JointAngle3 = 150;
DH_JointAngle4 = 90;

% Prompt the user for joint angles
% DH_JointAngle1 = input('Enter DH Joint Angle 1 in degrees: ');
% DH_JointAngle2 = input('Enter DH Joint Angle 2 in degrees: ');
% DH_JointAngle3 = input('Enter DH Joint Angle 3 in degrees: ');
% DH_JointAngle4 = input('Enter DH Joint Angle 4 in degrees: ');

jointAngles = [DH_JointAngle1, DH_JointAngle2, DH_JointAngle3, DH_JointAngle4];
[servoJointAngles,errorCode] = dh2servo(jointAngles);
```

```
DH Joint angles are: -151 -240  150   90
Servo Joint angles are: -2.6354      2.0944      2.618      1.5708
In degrees without pi to pit transform, Servo Joint angles are: -151 -150  150   90
```

```
if (errorCode == 1)
    servoJointAngles = []; % Empty array for servo joint angles
    error('Joint angles are outside the allowable range of [-150, 150] degrees')
end
```

```
Joint angles are outside the allowable range of [-150, 150] degrees
```

As can be observed, error is given *"Joint angles are outside of the allowable range of [-150 150] degrees"* , and the code exits.

Figure 7: Example of ErrorCode working

C. Inverse Kinematics

When trying to understand if our Inverse is working or not, we coded and found Optimal Solutions. When any position was not reachable, it would give imaginery value, which resulted from a negative value being square-rooted. This obviously didn't go to our arm and it didn't do anything.

We had to revise our inverse kinematics' equations cause of that. There were positions that we felt that our elbow-down modelled arm should reach easily, but it failed to find optimal solutions.

D. Furthermore

Later labs were more integration based, so most of the errors were variants of previously mentioned examples one way or another. In areas we felt that the our assumptions didn't warrant such a huge error, we would try to understand if it was replicable, when did it exactly fail, does it meet our expectations? If not, re-visit and try to verify if we did it correct with RA, our fellow colleagues or from the Professor himself if we felt something was still intuitively wrong.

IV. Future Work

To elaborate furthermore, reference section 4, and you will notice that there is no real-time feedback taking place. We recommend implementing a complete visual feedback-based closed loop which will be measuring the end-effector's position continously, compare it with "where we expected it to be", find the error, and then adjust accordingly. It will also allow for system to retry or adjust its actions accordingly, for cases like failed grasps or misplacement.

For even more rigour control, we can continuously keep "finding out joint angles" using $get\_pos()$ command, compare it with where we expect the end effector to be at that given time, find the noticeable error, and then adjust accordingly. This will cater to mechanical wear and tear as well in the future as well,

Additionally, optimizing the pick-and-place trajectories and gripper parameters based on collected data can lead to smoother and more precise operations, reducing the chances of errors during grasping and placing tasks.

If there are no timing constrictions, we can use SURF algorithm to more consistently correctly identify objects, but if we want robot to act with more efficiently with as least image processing as possible, then we will go with Kalman Filter.

We also haven't modelled keeping gravity in mind if you refer to 6 and how it effects our robotic arm. We can relax our ideal "perfectly rigid" model of our arm, and have it so that it caters to that physical reality of our world.

V. Conclusion

In summary, we implemented automatic pick and place task using robotic arm with the use of computer vision. We started by identifying the real world coordinates of the desired block to be picked using perception. We then found the solution of arm's inverse kinematics equations and inputted the coordinates to get the optimal joint angles. Hence we picked the object and placed it onto desired location. Given more time we could work on correcting the error and improving our perception for various lighting conditions. We could also work on more complex problems, like stacking one block over another, organizing the blocks in form of a shape, sorting objects by color or size.

Acknowledgments

This entire project couldn't be done without the contribution of any single one of our member, be it Huzaifah, Asghar or Daniyal. But most notable thanks definitely goes to the following two;

Miss Sadaf for always being patient when the equipment of our lab weren't the best and failing. For being calm and always going from one place to another, continously guiding everyone through the lab, cause everybody had questions all the times throughout the 3 hours we spent each week (and more ouside the class timings).

Dr.Basit for always being available throughout the lab, and even outside it. Even if the labs were extremely difficult and we annoyed him with very late submissions, he was always supportive to the best of his capabilities, and we doubt that we will ever forget his kindness. And last but not the least, thank you the reader for reading it through all of this now

References

[1] AsgharBoi. (2024). *RoboticsSpring2024*. GitHub Repository. Retrieved from https://github.com/AsgharBoi/RoboticsSpring2024

[2] Tekkotsu Robotics. (2009, October 22). Denavit-Hartenberg Reference Frame Layout [Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=rA9tm0gTln8&ab_channel=TekkotsuRobotics

[3] MathWorks. (2024, January 18). *Object-Oriented Programming Onramp*. MATLAB Academy. Retrieved from https://matlabacademy.mathworks.com/details/object-oriented-programming-onramp/oroop

[4] MathWorks. (2024, January 18). *Simulink Onramp*. MATLAB Academy. Retrieved from https://matlabacademy.mathworks.com/details/simulink-onramp/simulink

[5] MathWorks. (2024, January 18). *Introduction to Symbolic Math with MATLAB*. MATLAB Academy. Retrieved from https://matlabacademy.mathworks.com/details/introduction-to-symbolic-math-with-matlab/symbolic
[6] First Principles of Computer Vision. (2021, Apr). Structured Light Range Finding — Active Illumination Methods [Online Video]. Available: https://www.youtube.com/watch?v=3S3xLUXAgHw
[7] MathWorks. (2023, February 23). *Image Processing with MATLAB*. MATLAB Academy. Retrieved from https://matlabacademy.mathworks.com/details/image-processing-with-matlab/mlip#module=7