# Lab 3: Path Planning & Visual Servoing

Shyam Varsani

Suryansh Ankur

Instructor: Martin Jagersand

Date: 25th Oct 2023

## Introduction

Within the field of robotics, the ability to navigate and interact with the physical world holds a position of paramount importance. Lab 3: Path Planning & Visual Servoing represents an endeavor to augment the capabilities of a 2 DOF robot arm, focusing on precision, autonomy, and adaptability. This assignment requires the expansion of the robot's capabilities through the integration of path planning functionalities, enabling it to perform tasks such as drawing lines between defined points that fall within the robot arm's working space.

In the second part of this assignment, we explore Uncalibrated Visual Servoing (UVS). Our goal is to enable our planar robot arm to move autonomously to a specific position while tracking both its own end-effector and a moving target. UVS is a significant step forward in making our robot adaptable to dynamic real-world scenarios and capable of monitoring objects of interest. By using OpenCV's tracking modules, we've harnessed the power of dynamic tracking, opening up new possibilities for real-time interaction and control.

In this report, we've gathered data, made observations, and tackled the challenges along the way. It serves as a guide to share what we've learned and accomplished during this journey.

## Section 1: Path Planning

In this section, we delve into the implementation of path planning for our 2DOF robot arm, a vital enhancement in our journey towards precision and versatility. Our objective is to empower the robot with the capability to draw a straight line defined by two points within its working space.

To accomplish this, we have developed a Python function named draw_straight_line. This function takes as input two points, specifically their coordinates (x, y), and the number of steps required to traverse the line. The function ensures that the number of steps is a positive integer to ensure the validity of the operation.

Next, the function calculates the step increments for both the x and y directions. These increments, dx and dy, determine how much the robot arm should move at each step to reach the next point.

The function then initializes the current position to the starting point (point1), and through a loop, iterates through the intermediate points along the straight line. At each step, it moves the robot arm to the calculated position (x, y), offering precision and repeatability in the robot's movement.

## Section 1.2: Measuring Accuracy and Repeatability of Arm Movements for Path Planning

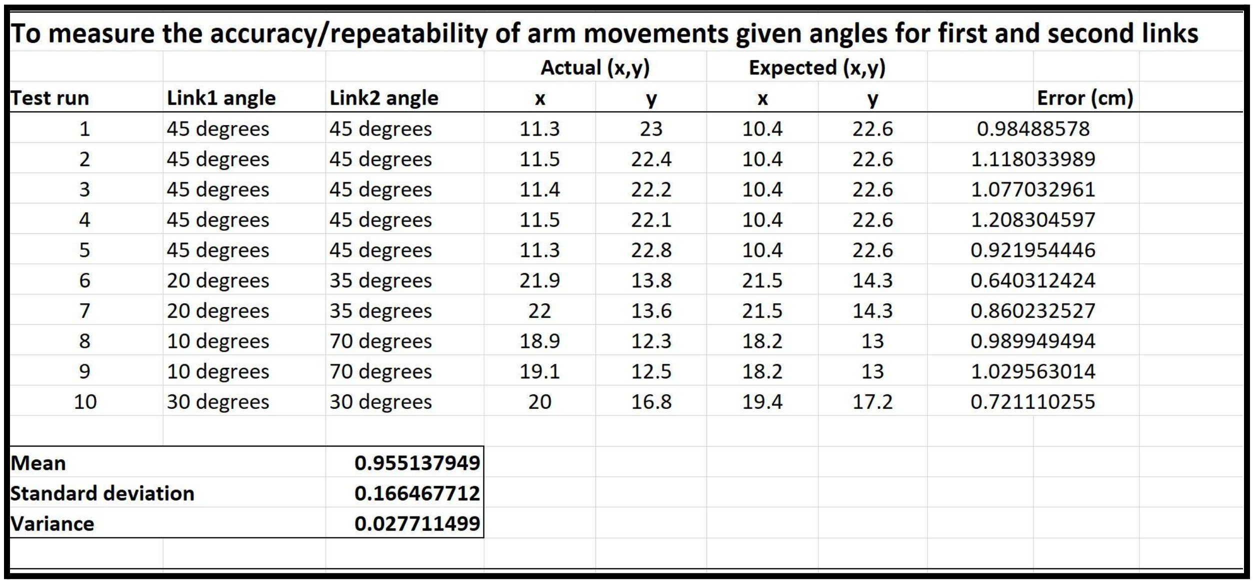
In this section, we delve into the critical task of assessing the accuracy and repeatability of our robot arm's movements. The path planning functionalities we've integrated demand precision and reliability, making this evaluation essential for our robot's performance.

**Experimental Setup:**

Our experimental setup closely follows the methodology employed in our previous laboratory assignment, despite changes in motor hardware. This decision stems from our need to address residual motor freedom and errors. The setup is as follows:

- Joint Angle Specification: We begin by specifying joint angles for both the first and second links of the robot arm. These angles simulate various target positions within the arm's operational space.
- Anticipating (x, y) Positions: For each experimental iteration, we record the desired joint angles and anticipate the corresponding (x, y) positions. These anticipations are based on our comprehensive understanding of the arm's kinematics.
- Robotic Execution: The robot then receives instructions to move to the specified joint angles, with our program orchestrating the precise movements.
- Error Analysis: Our program comes into play, meticulously calculating the actual (x, y) position of the end effector.

**Results and Analysis:**



**To measure the accuracy/repeatability of arm movements given angles for first and second links**

| | | | Actual (x,y) | | Expected (x,y) | | |
|---|---|---|---|---|---|---|---|
| Test run | Link1 angle | Link2 angle | x | y | x | y | Error (cm) |
| 1 | 45 degrees | 45 degrees | 11.3 | 23 | 10.4 | 22.6 | 0.98488578 |
| 2 | 45 degrees | 45 degrees | 11.5 | 22.4 | 10.4 | 22.6 | 1.118033989 |
| 3 | 45 degrees | 45 degrees | 11.4 | 22.2 | 10.4 | 22.6 | 1.077032961 |
| 4 | 45 degrees | 45 degrees | 11.5 | 22.1 | 10.4 | 22.6 | 1.208304597 |
| 5 | 45 degrees | 45 degrees | 11.3 | 22.8 | 10.4 | 22.6 | 0.921954446 |
| 6 | 20 degrees | 35 degrees | 21.9 | 13.8 | 21.5 | 14.3 | 0.640312424 |
| 7 | 20 degrees | 35 degrees | 22 | 13.6 | 21.5 | 14.3 | 0.860232527 |
| 8 | 10 degrees | 70 degrees | 18.9 | 12.3 | 18.2 | 13 | 0.989949494 |
| 9 | 10 degrees | 70 degrees | 19.1 | 12.5 | 18.2 | 13 | 1.029563014 |
| 10 | 30 degrees | 30 degrees | 20 | 16.8 | 19.4 | 17.2 | 0.721110255 |

| | |
|---|---|
| **Mean** | **0.955137949** |
| **Standard deviation** | **0.166467712** |
| **Variance** | **0.027711499** |

*Figure 1: Sample results from accuracy measurement experiments.*

Our pursuit of precision and repeatability yields a wealth of data, granting us invaluable insights into our robot's performance. Figure 1, exemplified in the table, offers a snapshot of the results garnered from our accuracy measurement experiments, mirroring our earlier lab work. These results encapsulate critical information, including the specified joint angles, the expected (x, y) positions, and the recorded actual (x, y) positions of the end effector. Furthermore, we rely on the Euclidean distance error to quantify the accuracy of the robot's movements.

**Interpreting Accuracy and Repeatability:**

To comprehend the accuracy and repeatability of our robotic arm's movements, it is essential to consider the factors contributing to the observed errors. A substantial portion of these errors can be attributed to two primary sources:

*Vertical Play in Motors:* The presence of slight but substantial vertical play in the motors can introduce variations in the arm's positioning.

*Motor Degrees of Freedom:* Another significant source of error arises from the motors' approximately 10 degrees of freedom of movement, both clockwise and anti-clockwise, without actually propelling the arm.

- Test Runs 1 to 5: All set with 45 degrees for both Link1 and Link2 angles. The expected (x, y) positions were consistently around (10.4, 22.6) cm. The actual positions varied slightly, resulting in an error ranging from 0.922 to 1.208 cm.

- Test Runs 6 and 7: For Test Run 6, the angles were 20 degrees for Link1 and 35 degrees for Link2. The expected (x, y) positions were (21.5, 14.3) cm. The actual positions exhibited an error of approximately 0.64 cm. Test Run 7 displayed similar results with a slightly higher error of around 0.86 cm.

- Test Runs 8 and 9: In Test Run 8, the angles were 10 degrees for Link1 and 70 degrees for Link2, resulting in an expected (x, y) position of (18.2, 13) cm. The actual positions showed an error of approximately 0.99 cm. Test Run 9 yielded a slightly larger error of about 1.03 cm.

- Test Run 10: In this test, both Link1 and Link2 were set to 30 degrees. The expected (x, y) position was around (19.4, 17.2) cm, with an error of about 0.72 cm.

The overall mean error across all test runs was approximately 0.955 cm, with a standard deviation of around 0.166 cm and a variance of roughly 0.028.

These results indicate that the robot arm's movements exhibit a level of accuracy, with errors typically less than 1.5 cm. While slight variations in the actual positions occur, the robot demonstrates a reasonably consistent performance in reaching the expected positions specified by the joint angles. This assessment provides valuable insights into the arm's precision and its capabilities for path planning tasks.

## Section 2: Uncalibrated Visual Servoing (UVS)

In this section, we explore the world of Uncalibrated Visual Servoing (UVS) and how we've integrated it into our robot system. Our core objective with UVS is to equip our 2-DOF planar robot arm with the capability to autonomously navigate to specified positions while dynamically tracking its end-effector and a moving target. This advancement enhances our robot's adaptability in responding to real-world situations and enables real-time object monitoring.

Our UVS implementation relies on a carefully crafted Python script. It acts as the central command center, orchestrating the robot's movements and vision-based tracking. Two key elements play a pivotal role in this operation: the *'tracker'* and the *'server.'*

> **The Tracker**: This component continuously monitors desired points and tracks the robot's end-effector and a moving target in real-time. It harnesses OpenCV's tracking modules, allowing our robot to adapt its path as tracked objects move.
>
> **The Server**: This element ensures seamless communication between the robot and the tracking components, facilitating precision and control.

### The UVS Class

#### 1. Jacobian Matrix: Guiding Precision

The Jacobian matrix is the lynchpin of our UVS implementation. This matrix serves as the critical bridge linking the robot's joint angles and its end-effector position. It is the key to ensuring that the robot's movements are executed with precision and accuracy. As the robot dynamically performs real-time path planning and tracking, the Jacobian matrix guides its motion with exceptional precision, making sure it reaches its desired locations with the utmost accuracy.

#### 2. Control System Gains: Kp and Kd

Our UVS implementation incorporates control system gains in the form of Kp (Proportional Gain) and Kd (Derivative Gain). These gains play a pivotal role in determining how the robot responds to error. Kp dictates the strength of the robot's response to discrepancies, ensuring it maintains precision. Kd comes into play by predicting changes in error and counteracting overshooting, fine-tuning the robot's movements. Together, Kp and Kd work harmoniously to guarantee that the robot's actions are finely balanced and accurate.

**3. Tracking Queue: Seamless Communication**

The inclusion of a tracking queue is pivotal in our UVS implementation. This queue serves as the conduit for seamless communication between the robot and the tracking system. It manages the efficient transmission of data, ensuring that information flows effortlessly and that the robot remains in perfect synchronization with tracked objects in real-time. This communication mechanism is the driving force behind the robot's dynamic tracking capabilities, allowing it to adapt swiftly to changes in its environment and maintain precision while monitoring objects.

**4. State Variables: Constant Adaptation**

In our UVS implementation, the 'UVS' class is equipped to provide constant updates to crucial state variables. These variables encompass the current position (point) and the desired goal position. These updates are essential for computing the error, which, in turn, guides the robot's movements. The continuous adaptation enabled by these updates ensures that the robot can respond promptly to changing environmental conditions and adapt to evolving tracking circumstances.

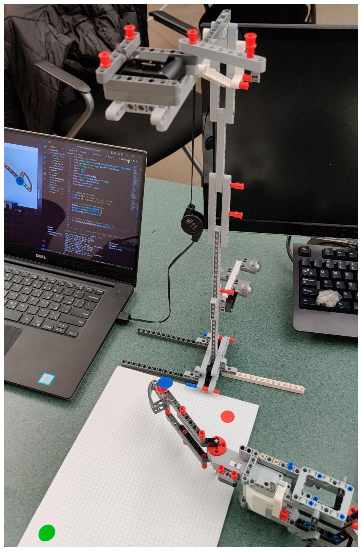
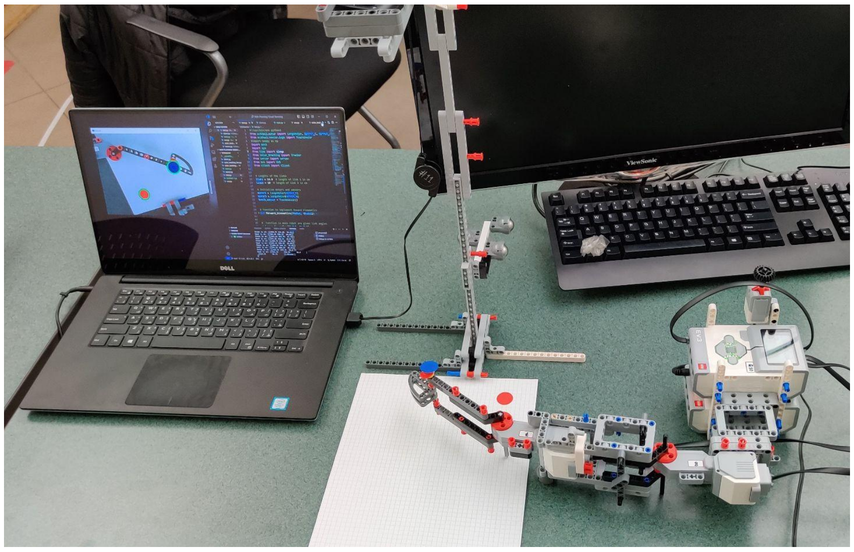
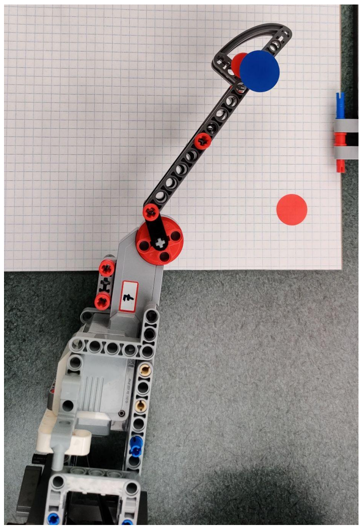
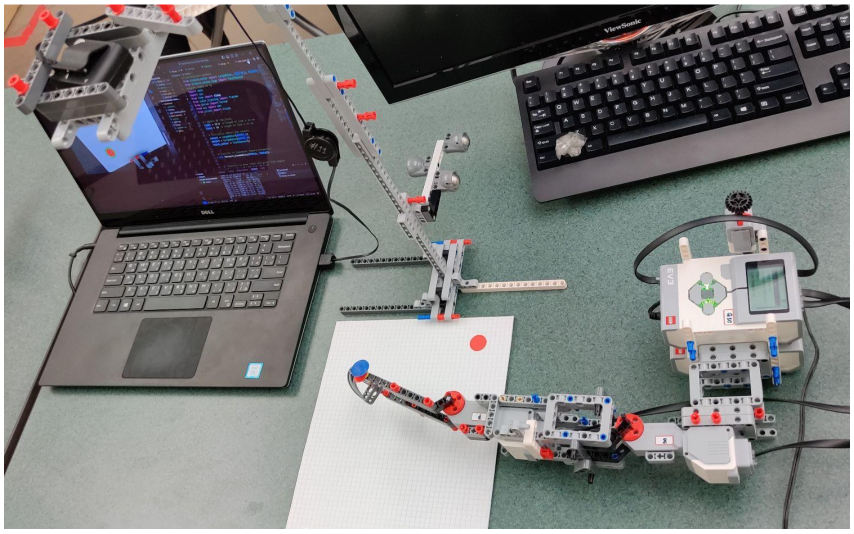
**5. Delta Angles: Dynamic Adjustments**

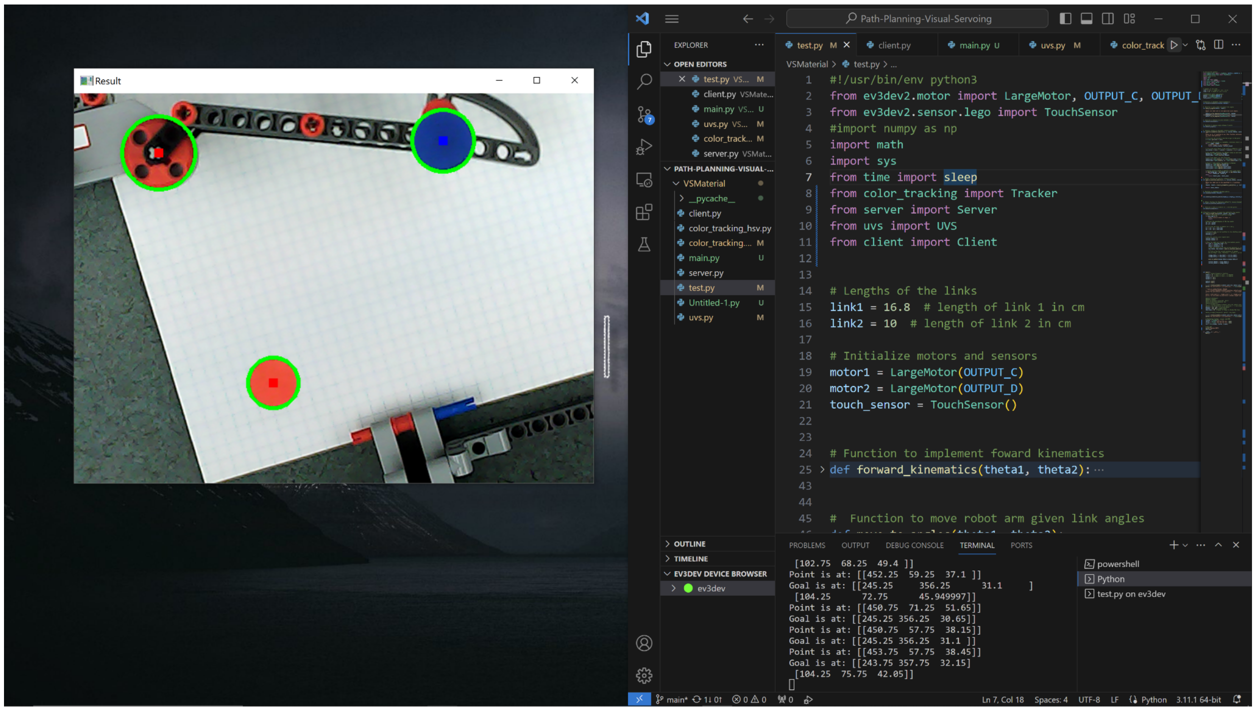
Our robot actively calculates delta angles, representing dynamic adjustments necessary to guide its path and minimize error. These calculated angles are instrumental in real-time control, enabling the robot to adapt swiftly to changing conditions and maintain precision while tracking moving targets.

The 'UVS' class serves as the backbone of our UVS implementation, harmoniously incorporating these components. This synergy empowers our 2-DOF planar robot arm with dynamic, precision-driven capabilities, enabling it to excel in navigating, tracking, and interacting effectively with the world. Through UVS, our robot takes a significant step forward in terms of adaptability and responsiveness, making it more than just a machine; it becomes an adaptable and precise entity.

## The setup:

The images below give a visual representation of our setup for our implementation of universal visual servoing. This particular orientation of the camera and the robotic arm helps to ensure that the camera is directly ontop of the arm hence allowing a bird's eye view for tracking the end effector and the target location which are depicted by the blue and red colors respectively.



```python
#!/usr/bin/env python3
from ev3dev2.motor import LargeMotor, OUTPUT_C, OUTPUT_
from ev3dev2.sensor.lego import TouchSensor
#import numpy as np
import math
import sys
from time import sleep
from color_tracking import Tracker
from server import Server
from uvs import UVS
from client import Client


# Lengths of the links
link1 = 16.8  # length of link 1 in cm
link2 = 10  # length of link 2 in cm

# Initialize motors and sensors
motor1 = LargeMotor(OUTPUT_C)
motor2 = LargeMotor(OUTPUT_D)
touch_sensor = TouchSensor()


# Function to implement foward kinematics
def forward_kinematics(theta1, theta2): ...


#  Function to move robot arm given link angles
```

```
 [102.75  68.25  49.4 ]]
Point is at: [[452.25  59.25  37.1 ]]
Goal is at: [[245.25     356.25       31.1      ]
 [104.25      72.75      45.949997]]
Point is at: [[450.75  71.25  51.65]]
Goal is at: [[245.25 356.25  30.65]]
Point is at: [[450.75  57.75  38.15]]
Goal is at: [[245.25 356.25  31.1 ]]
Point is at: [[453.75  57.75  38.45]]
Goal is at: [[243.75 357.75  32.15]
 [104.25  75.75  42.05]]
```

The image above shows the tracking of the end effector and the target location in effect as they have been circled by the green circles. This process takes place in real-time thus the robot arm can effectively calculate how far off it is from the target location at any given time. This can also be seen in the terminal output in the bottom right corner of the picture, where the (u,v) location of the end effector and the target is displayed during every iteration of the uvs algorithm. Keep in mind that these values are inform of raw pixels and not cm.

Our UVS journey commences with the vital task of initializing the Jacobian matrix through incremental orthogonal motions, enabling the robot to grasp its kinematic relationships. Error computation is central, continuously evaluating the disparity between the desired goal and the robot's current position, steering its movements.

Flexibility is key; the Jacobian matrix dynamically adapts through Broyden's update method, ensuring precision by responding to prior actions and error fluctuations. Calculated delta angles guide the robot with precision.

Operated within an iterative loop, our UVS script incrementally adjusts joint angles until the error falls below a defined threshold. Exception handling and safety measures are in place to uphold operational integrity.

## Section 3: Adapting UVS for 3D Coordinate System

Transitioning from controlling and moving a robot arm in a 2D coordinate system to a 3D coordinate system involves significant adaptations to our existing UVS program, equipment setup, and mathematical equations. These modifications are essential to enable our robot arm to navigate and interact effectively in three dimensions (x, y, z). Below, we detail the changes required:

### Program Modifications:

**1. Enhanced Dimensionality:** Our UVS program, currently limited to a two-dimensional plane, must be extended to operate in three dimensions (x, y, z). The program needs to support 3D coordinates for tracking, planning, and control. This expansion is vital for the robot arm to perform precise, adaptive, and coordinated motions in 3D space.

**2. End-Effector Control:** The existing mechanisms that control the robot's end-effector in a 2D plane must be redesigned to accommodate 3D control. This enhancement allows the robot to control the end-effector's position along all three dimensions (x, y, z). The robotic tooling and control interfaces should be updated to enable this expanded range of motion.

**3. 3D Error Computation:** Our error computation process, which presently calculates the difference between the desired 2D goal position (x, y) and the robot's real-time 2D position, needs to be extended to include the third dimension (z). The error calculation must consider the full 3D position (x, y, z) to guide the robot accurately in a 3D workspace. This extension enables the robot to respond effectively to variations in height and depth.

### Equipment Setup:

**1. 3D Vision System:** To support the transition to a 3D coordinate system, our existing vision system based on 2D cameras must be upgraded to incorporate 3D vision capabilities. This entails the introduction of depth-sensing cameras or stereo cameras, capable of capturing spatial depth information. These advanced cameras provide data for the third dimension (z-axis), enabling the robot to perceive the environment in full 3D, not just in the x and y dimensions.

**2. Positional Feedback:** In a 3D environment, the robot arm requires enhanced positional feedback systems. These systems must provide accurate tracking and reporting of movements not only in the horizontal plane but also along the vertical (z) axis. Incorporating 3D positional feedback mechanisms empowers the robot to operate effectively in a three-dimensional workspace. They provide real-time data about the robot's location in this expanded space, allowing for precision and adaptability.

## Mathematical Equations:

**1. Expanded Jacobian Matrix:** Our existing Jacobian matrix, which maps joint velocities to end-effector linear velocities in the x and y directions, needs to be expanded to accommodate 3D motion. The new Jacobian matrix should be a 3x2 matrix to effectively capture the robot's motion across all three dimensions (x, y, z). This expansion is crucial to guide the robot's motion with precision in a 3D environment.

**2. Advanced Kinematics:** The kinematic equations that currently govern the robot's movement and transformations in a 2D space must be updated to include three-dimensional kinematics. These revised kinematic equations will calculate 3D joint angles and positions, enabling the robot to traverse and interact effectively in a three-dimensional workspace.

**3. Modified Control Gains:** Our control system gains, specifically Kp (Proportional Gain) and Kd (Derivative Gain), need to be fine-tuned to suit 3D motion. These control gains influence how the robot responds to deviations and errors in its movements. Kp dictates the strength of the robot's response to discrepancies in all three dimensions, ensuring precision across the expanded workspace. Kd predicts changes in error and counteracts overshooting, fine-tuning the robot's movements. Adjusting Kp and Kd harmoniously ensures that the robot's actions are finely balanced and accurate in three-dimensional space.

**4. 3D Error Metrics:** The error calculation process should be updated to consider the Euclidean distance in 3D space. While previously focused on two dimensions (x, y), the error calculation now encompasses all three dimensions (x, y, z). This change enhances the robot's precision and adaptability in tracking and responding to the 3D environment. Clear error thresholds should be defined to guide the robot effectively in this expanded workspace.

The adaptations detailed above will empower our robot arm to move and interact in a three-dimensional coordinate system, significantly expanding its capabilities and allowing it to perform tasks with greater precision and adaptability in complex 3D environments.

## Conclusion

This report chronicles our quest to enhance the capabilities of a 2 DOF robot arm, emphasizing precision, autonomy, and adaptability. We ventured into two vital areas: path planning and Uncalibrated Visual Servoing (UVS).

In path planning, we developed a Python function enabling the robot to draw straight lines between defined points in its workspace. Despite minor motor limitations, our analysis revealed commendable precision, with errors typically under 1.5 cm.

Our exploration of UVS empowered our planar robot to autonomously navigate and track its end-effector and moving targets. Components like the Jacobian matrix, control gains (Kp and Kd), and a tracking queue created a dynamic, precision-driven system for real-time object monitoring.

We also prepared our robot for a transition from 2D to 3D coordinates. This entailed enhancing our UVS program, equipment setup, and mathematical equations to accommodate three-dimensional motion.

In conclusion, our journey signifies a significant stride toward making our robot adaptable and responsive in real-world scenarios. These advancements open new possibilities for automation, marking our robot as more than a machine – a precise and adaptable entity in the realm of robotics and beyond.