Lab - 4: Connecting mycobot pro 600 with its digital twin

Devika Shaj Kumar Nair[1], Darsh Patel[2], Aman Milind Dhale[3]
[1]Ira Fulton School of Engineering, Arizona State University, Arizona, United States of America
[2]Ira Fulton School of Engineering, Arizona State University, Arizona, United States of America
[3]Ira Fulton School of Engineering, Arizona State University, Arizona, United States of America

Article Info

Article history:

Received November 24, 2024

Keywords:

ArUco Markers
Inverse Kinematics
TCP Socket Connection
MyCobot 600 Pro
ROS
Homogeneous Transformation
Digital Twin

ABSTRACT

This experiment focuses on implementing a vision-guided robotic arm navigation system using ArUco markers for localization and control. The process begins with detecting ArUco markers in a given workspace and extracting their pixel coordinates from a camera feed. These coordinates are then transformed into physical coordinates using calibration equations derived from camera parameters. The physical positions, along with predefined orientations, are used as input to an inverse kinematics solver to compute the joint angles required for the robot's end effector to navigate to the specified positions. The computed joint angles are validated in a MATLAB-based digital twin simulation and subsequently transmitted to a physical robotic arm via a TCP socket connection. The robotic arm replicated the motion observed in the digital twin, successfully navigating between marker positions with high accuracy. This experiment highlights the integration of computer vision, robotics kinematics, and control systems to achieve precise and reliable robotic motion in both simulated and real-world environments.

Corresponding Author:

Devika Shaj Kumar Nair
Ira Fulton School of Engineering, Arizona State University
Tempe, Arizona, United States of America
Email: dshajkum@asu.edu

1. INTRODUCTION

Mycobot 600 Pro is a 6-axis collaborative robot that was designed for both commercial and educational purposes. It has an operating radius of 600mm, with an effective load capability of 2kg. The robot weighs around 8.8kg and it utilizes a Raspberry Pi microprocessor embedded with RoboFlow visual programming software. It has 6 joints with the following joint angle limits[1].

Table 1: Joint angle range for Mycobot 600 Pro

Joint	Angle Range
J1	+/- 180°
J2	-270~90°
J3	+/- 150°
J4	-260~80°
J5	+/- 168°
J6	+/- 174°

The main aim of this lab report is to walk through the process followed for conducting the forward kinematic analysis of Mycobot 600 Pro and generate a digital twin of the same in MATLAB. We also implement the inverse kinematics solution to navigate between two points detected using the ArUco markers

2. KINEMATIC MODEL AND HOMOGENEOUS TRANSFORMATIONS

Homogeneous transformation matrices are derived from the configuration diagram of the robot model. Here, the homogeneous transformation matrices of the MyCobot 600 Pro robot are generated manually from the configuration diagram shown in figure 1 and the corresponding transformation matrices obtained are as follows:

```
H01 = [cos(t1)      0    sin(t1)    0;
       sin(t1)      0   -cos(t1)    0;
       0            1      0        210;
       0            0      0        1];

H12 = [-cos(t2)  -sin(t2)    0      -250*cos(t2);
       -sin(t2)   cos(t2)    0      -250*sin(t2);
        0           0       -1      76.2;
        0           0        0      1];

H23 = [-cos(t3) -sin(t3) 0       250*cos(t3);
       -sin(t3)  cos(t3) 0       250*sin(t3);
       0        0       -1      76.2;
       0        0        0      1];
```

```
H34 = [-cos(t4)   0  sin(t4)    0;
       -sin(t4)   0  -cos(t4)   0;
       0         -1        0    76.2;
       0          0        0    1];

H45 = [cos(t5)   0   sin(t5)    0;
       sin(t5)   0  -cos(t5)    0;
       0         1        0     107;
       0         0        0     1];

H56 = [-cos(t6)   sin(t6) 0     0;
       -sin(t6)  -cos(t6) 0     0;
       0         0        1     109.5;
       0         0        0     1];
```

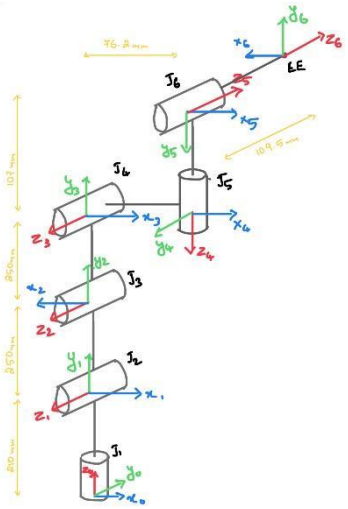


Figure 1: Configuration Diagram for Mycobot 600 Pro

The above homogeneous transformations yielded the following coordinates in MATLAB.

```
>> Homogeneous_Transformation
H06 =
   -1.0000         0         0         0
         0         0   -1.0000 -185.7000
         0   -1.0000         0  817.0000
         0         0         0    1.0000

X =
     0

Y =
  185.7000

Z =
   817
```

```
>> Homogeneous_Transformation
H06 =
         0         0    1.0000  185.7000
   -0.2588    0.9659         0 -521.6122
   -0.9659   -0.2588         0  479.1751
         0         0         0    1.0000

X =
 -185.7000

Y =
  521.6122

Z =
  479.1751
```

```
>> Homogeneous_Transformation
H06 =
    1.0000         0         0 -500.0000
         0         0   -1.0000 -185.7000
         0    1.0000         0  103.0000
         0         0         0    1.0000

X =
   500

Y =
  185.7000

Z =
   103
```

Figure 2: Homogeneous transformation results for Joint angles [0, -90, 0, -90, 0, 0], [90, -45, 30, -90, 0, 0] and [0,0,0,0,0,0] respectively

3. ArUCo MARKER DETECTION

In this task, we utilized Python and OpenCV[4] to map an arena with ArUCo markers [3] using the AI Kit camera. Our approach involved creating a Python script with three key functions to implement pixel-to-physical coordinate mapping. The process began with setting up the robot and calibrating the camera with the arena. For calibration, we manually moved the robot's end-effector to predefined points (top-left and bottom-right corners) visible in the camera feed, recording both their pixel and physical coordinates.

A linear mapping function, based on the equation of a line **y = mx + c**, was implemented to transform pixel coordinates into physical coordinates. The calibration function calculated slopes (mx, my) and intercepts (cx, cy) using the formula:

$$m_x = \frac{x_{\text{top\_physical}} - x_{\text{bottom\_physical}}}{x_{\text{top\_pixel}} - x_{\text{bottom\_pixel}}} \qquad m_y = \frac{y_{\text{top\_physical}} - y_{\text{bottom\_physical}}}{y_{\text{top\_pixel}} - y_{\text{bottom\_pixel}}}$$

$$c_x = x_{\text{top\_physical}} - m_x \cdot x_{\text{top\_pixel}} \qquad c_y = y_{\text{top\_physical}} - m_y \cdot y_{\text{top\_pixel}}$$

ArUCo markers in the camera's field of view were detected using the OpenCV aruco module, which identified marker corners. The center coordinates of each marker were calculated by averaging the corner coordinates. The function **getRobotCoordinates()** transformed these pixel coordinates into the robot's physical coordinates using the previously computed mapping parameters.

The script demonstrated real-time detection of ArUCo markers, drawing markers and their IDs on the video feed while converting marker center positions into robot coordinates. The successful calibration and mapping process ensured precise alignment between the camera feed and the robot's physical workspace. This setup enabled accurate end-effector positioning for subsequent robotic tasks.

4. INVERSE KINEMATICS

To calculate the joint angles for the MyCobot 600 Pro robotic arm, we utilized an Inverse Kinematics (IK) approach implemented in MATLAB. The process begins by importing the robot's URDF model and defining its data format and gravity. The end effector, specified as link6, represents the target point for IK calculations. The user is prompted to input the start and end positions and orientations of the end effector. These positions are provided as [x,y,z] coordinates, and the orientations as roll, pitch, and yaw angles in degrees. The orientation values are then converted to radians for computation.

An IK solver is initialized using the robot's rigid body tree model. The solver settings, including the algorithm (BFGSGradientProjection[5]) and parameters like maximum iterations and tolerances, are configured to ensure precision and convergence. Position and orientation weights are set to prioritize accurate placement over orientation adjustments, and the robot's home configuration is used as the initial guess for joint positions. The end-effector's position and orientation are combined into a transformation matrix using the trvec2tform and eul2tform functions. This transformation is passed to the IK solver along with the defined weights and initial guess. The solver computes the joint angles that achieve the desired end-effector pose while adhering to the robot's kinematic constraints.

The process is repeated for both the start and end positions. The joint angles for these configurations are extracted and converted from radians to degrees for readability. These angles represent the robot's motion path in joint space. This approach ensures precise control of the robot's motion based on user-defined positions and orientations.

5. IMPLEMENTATION IN PHYSICAL ROBOT

To establish communication with the MyCobot 600 Pro robotic arm and control its movements, we implemented a TCP socket[6] connection using Python. This allows joint angle commands, calculated through inverse kinematics, to be transmitted directly to the physical robot for execution. The script initializes a TCP socket and connects to the robot's server, specified by its IP address (192.168.1.159) and port

number (5001). Upon successful connection, a command string, such as set_angles(0, 0, 0, 0, 0, 0, 500), is sent to the robot. This command specifies the joint angles for all six joints of the robot along with a specified speed parameter.

The socket setup ensures bi-directional communication, allowing the robot to optionally send responses back to the client. This feedback mechanism aids in monitoring the success or failure of the transmitted commands. In the event of an error, the program safely handles socket exceptions, ensuring the connection is properly closed after use. This implementation demonstrates an efficient and robust way to transmit real-time control commands to the physical robotic arm, facilitating seamless integration between computed joint angles and the robot's mechanical execution.

6. RESULTS

This section presents the detailed outcomes of detecting ArUco markers, calculating joint angles via inverse kinematics, and verifying the movement of the robotic arm in both simulation and the physical setup. The results based on three sets of positions for ArUco markers, as described below:

6.1. ArUco Marker Detection

In the initial phase, the camera detected two ArUco markers for each set, and their pixel coordinates were displayed in the captured image. Using these pixel coordinates, the physical coordinates were calculated using the derived calibration equations. The three sets of detected physical coordinates for the markers are as follows (all positions in meters):

1. Set 1
   - Marker 2: (−0.4489,−0.2694)
   - Marker 1: (−0.3361,−0.3788)
2. Set 2
   - Marker 2: (−0.4732,−0.3383)
   - Marker 1: (−0.3210,−0.3335)
3. Set 3
   - Marker 2: (−0.4761,−0.3392)
   - Marker 1: (−0.3240,−0.3420)

For all three sets, the z-coordinate of the end effector was fixed at 0.04 m and the orientation was specified as 178°, 0°, 0° for roll, pitch, and yaw, respectively.

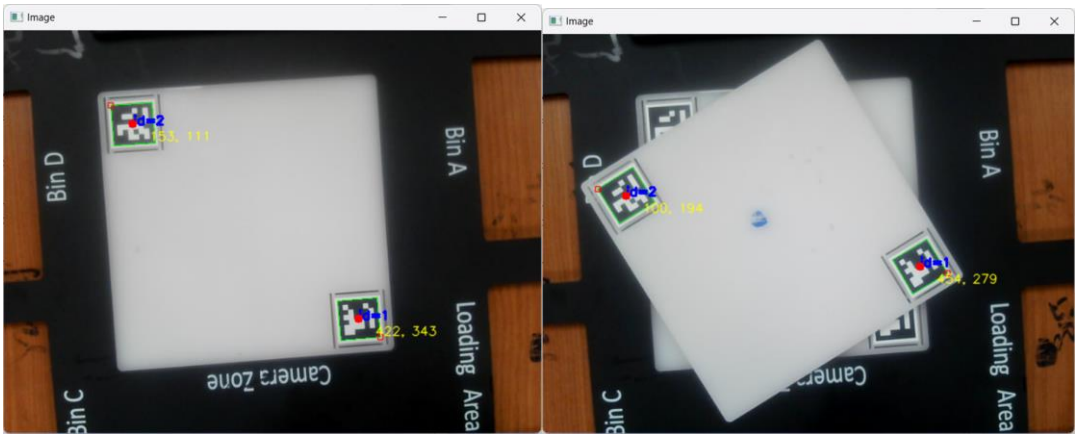


Figure 3: ArUco Marker detection and pixel positions

6.2. Inverse Kinematics

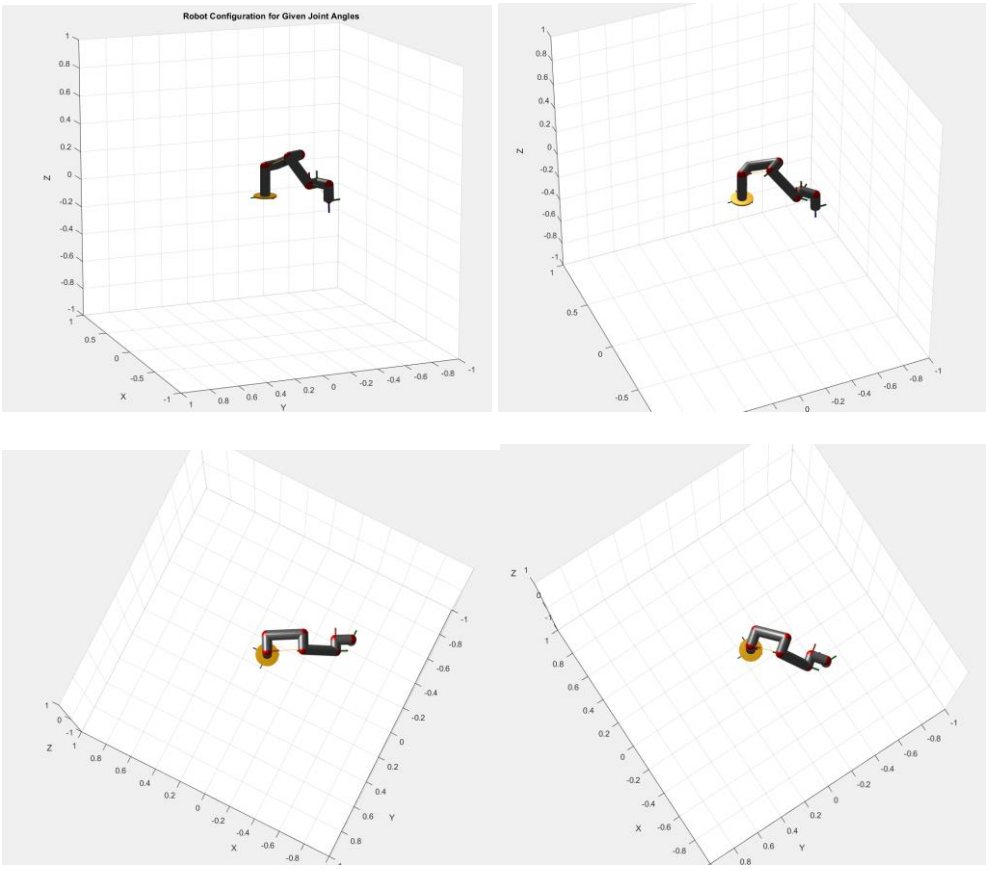
The physical coordinates and orientations were used as input to the inverse kinematics code. For each set, joint angles for both markers were calculated as follows:

1. Set 1
    - Marker 2: [-157.63º, -24.22º ,63.72º, -130.00º, -88.16º, -67.62º]
    - Marker 1: [-140.18º, -25.90º ,65.23º, -130.00º, -88.49º, -50.18º]
2. Set 2
    - Marker 2: [-152.30º, -11.83º ,38.84º, -117.94º, -88.23º, -62.29º]
    - Marker 1: [-142.90º, -32.34º ,72.50º, -130.00º, -88.45º, -52.92º]
3. Set 3
    - Marker 2: [-152.36º, -10.83º ,36.76º, -116.85º, -88.23º, -62.34º]
    - Marker 1: [-142.35º, -30.97º ,70.97º, -130.00º, -88.46º, -52.37º]

These joint angles were displayed in MATLAB for verification and were subsequently passed to the robotic arm.

6.3. MATLAB Visualization

For each set, the MATLAB script generated 3D visualizations showing the robotic arm configurations at the start and end positions for the respective marker locations. These figures illustrated the arm's ability to achieve the desired poses with high accuracy.



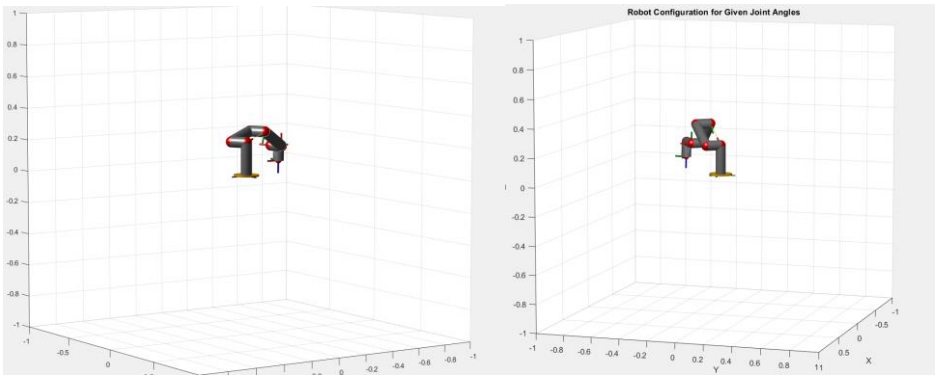


Figure 4: Joint angle validation in MATLAB

6.4. Physical Robot Execution

The calculated joint angles were transmitted to the physical robotic arm via TCP socket communication. The robotic arm successfully navigated between the marker positions, replicating the configurations observed in MATLAB. The operations were smooth and closely matched the simulated behavior, validating the effectiveness of the workflow.

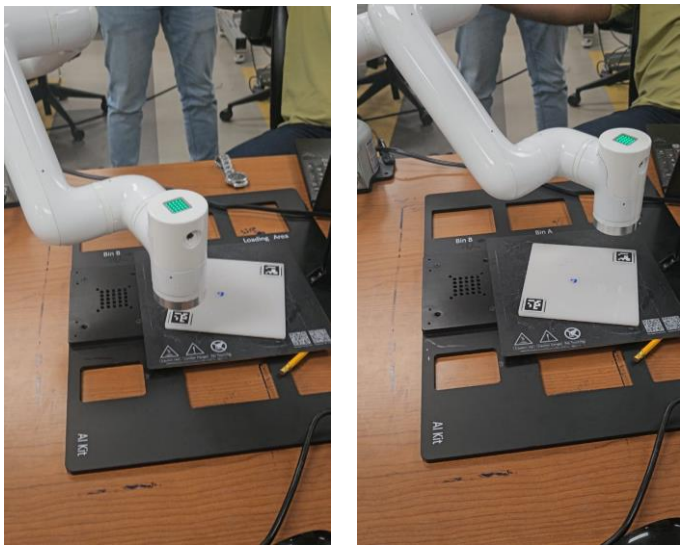


Figure 5: Physical robot navigation between two ArUco markers

The system successfully detected ArUco markers, computed the joint angles using inverse kinematics, and controlled the robotic arm to reach the specified positions and orientations. The results confirm the robustness and reliability of the implemented solution for marker-based robotic navigation.

7. CONCLUSION

In this lab, we successfully demonstrated the integration of ArUco marker detection, inverse kinematics, and robotic control to achieve precise motion in both a digital twin simulation and a physical robotic arm. Starting with the detection of ArUco markers and their pixel coordinates, we accurately transformed them into physical coordinates using calibration equations. These coordinates, along with predefined orientations, were fed into an inverse kinematics solver, which computed the joint angles required for the robotic arm to reach the desired positions.

The computed joint angles were validated through visualization in MATLAB, confirming the arm's capability to achieve the specified configurations. These results were replicated in the digital twin of the robotic arm.

Finally, the same joint angles were transmitted to the physical robot using a TCP socket connection, and the arm's motion mirrored the simulation results with high precision.

This workflow showcases a robust approach to integrating vision-based localization, kinematics computation, and real-world robotic control. The ability to seamlessly transition from simulation to physical implementation highlights the effectiveness of this method for tasks requiring accuracy and reliability in robotics applications. This success lays a solid foundation for further enhancements, such as dynamic obstacle avoidance or real-time adjustments based on marker detection.

REFERENCES

[1] Elephant Robotics, "MyCobot Pro 600 Documentation," Available: https://docs.elephantrobotics.com/docs/gitbook-en/2-serialproduct/2.3-myCobot_Pro_600/2.3-myCobot_Pro_600.html. Accessed: Sep. 14, 2024.

[2] Elephant Robotics, "Environment Building for ROS1," Available: https://docs.elephantrobotics.com/docs/pro600-en/12-ApplicationBaseROS/12.1-ROS1/12.1.2-EnvironmentBuilding.html. Accessed: Oct. 12, 2024.

[3] OpenCV, "ArUco: Table of Contents," Available: https://docs.opencv.org/3.4/d9/d6d/tutorial_table_of_content_aruco.html. Accessed: Nov. 23, 2024.

[4] OpenCV, "OpenCV Tutorials," Available: https://docs.opencv.org/4.x/d9/df8/tutorial_root.html. Accessed: Nov. 23, 2024.

[5] MathWorks, "Inverse Kinematics Algorithms," Available: https://www.mathworks.com/help/robotics/ug/inverse-kinematics-algorithms.html. Accessed: Nov. 23, 2024.

[6] Elephant Robotics, "Socket API Interface Description for myCobot Pro 600," Available: https://docs.elephantrobotics.com/docs/gitbook-en/2-serialproduct/2.3-myCobot_Pro_600/2.3.5%20socket%20API%20interface%20description.html. Accessed: Nov. 23, 2024.