Modeling and Control of a Pick and Place Robotic Arm

Waleed Atef, Omar Magdy,
Abdallah Mohamed, Fouad Elshaikh and Mohamed Hassan
German University in Cairo (GUC), Egypt
Emails: waleed.abdelrahman@student.guc.edu.eg, omar.abdelwaly@student.guc.edu.eg, fouad.elshaikh@student.edu.eg, mohamed.elattar@student.edu.eg, abdallah.falah@student.edu.eg

***Abstract*—This study focuses on creating and controlling joint-space trajectories for a robotic arm using MATLAB and Python. The main aim was to develop smooth paths that guide the arm's end-effector from an initial position, through an intermediate stage, to a final position. We defined the starting, middle, and end joint configurations, calculated the joint angles with inverse kinematics, and used MATLAB to generate the necessary trajectories. Python was then used for data analysis, visualization, and post-processing, adding flexibility and enhancing the control system.**

We also had to deal with issues like ensuring that the data was formatted correctly and that MATLAB and Python could communicate with each other without any problems. By resolving these problems, we improved the process' dependability and usability. This work shows that combining MATLAB and Python can be a powerful way to plan and control robotic motion. The methods we developed offer valuable insights into creating smooth, precise movement paths for robots and set the stage for future work in real-time control and more adaptive motion strategies.

Keywords-**Robotic arm; Modeling; numerical; control.**

I. Introduction

Robotic arms are widely used to carry out tasks demanding a high degree of accuracy and reliability in both academic and industrial contexts. This project aims to develop a five-DOF desktop robotic arm that is primarily suitable for pick-and-place operations. The goal is to create something small, economical, and effective that functions well in small spaces like workshops or labs. The project includes making the mechanical parts, picking proper sensors and actuators, and setting up a control system so it operates accurately. This report shows how the arm was designed, built, and tested, and talks about what it can do and where it can be used.

[1] talked about how robots are becoming preferred over human labor in tasks needing precision and accuracy, mainly because of their better performance and lower risks. An articulated robotic arm is made of links connected by rotary joints, where the number of joints decides its Degrees of Freedom (DOF). Servomotors, controlled by microcontrollers, provide the torque needed to move these joints. Proper design and simulation are very important to make sure the robotic arm can perform its tasks efficiently and accurately. The study focused on designing and analyzing a three-DOF robotic arm, looking at forward kinematics to find the position and orientation of the end effector, while inverse kinematics calculated the joint angles needed to reach a certain position. They also looked into static torques, link cross-sections, and workspace determination. Some challenges, like having multiple solutions and singularity points, were addressed using tools like SolidWorks and MATLAB, which helped validate the designs and improve motion accuracy.

[2] highlighted how manipulator-based robotic arms and small mobile robots are becoming more popular in educational setups worldwide. These platforms are great for learning about dynamics, kinematics, and control, all of which are crucial topics in modern robotics. Manipulator behavior is a key part of robotic system control, and understanding it requires good kinematic analysis. Forward kinematics is used to find the position of the end effector from joint parameters, while inverse kinematics works backward to find the joint angles. Small robotic arms like the AL5B, with four revolute joints, are often used for this purpose in education. Designing robotic arms with CAD tools and using small to medium-sized DC servo motors is common because these parts are easy to find and control with Arduino microcontrollers. However, programming them can be tedious, so researchers often use GUI-based software like LABVIEW to make it easier. By combining kinematic algorithms in MATLAB with LABVIEW, students can see how calculated positions compare to real positions, which helps them understand kinematics and errors better. This interactive approach has been shown to improve robotics education.

[3] introduced Mirobot, a low-cost, desktop-sized robotic arm with six degrees of freedom, made for educational use. Mirobot uses six stepper motors with reducers for the joints, and its structure is 3D-printed, which allows for easy design updates. An AVR MEGA2560 microcontroller is used to control the system, while inverse kinematics are solved using geometric methods and Euler angle transformations. The software also includes look-ahead control to keep the motion smooth. Mirobot can be controlled via computer or phone and even has a remote-control option, making it very user-friendly. Its affordable design, 3D-printed parts, and simple control system make it ideal for teaching robotics. It integrates forward and inverse kinematics, letting users try out complex motions in a hands-on way, making it perfect

for students and hobbyists.

From these studies, our project aims to build a desktop robotic arm specifically for pick-and-place tasks. This arm will be designed to support a drilling process carried out by another robot.

II. Hardware Design and Implementation

We started by designing the robotic arm on Solidworks, keeping in mind that it should fit the servos we found locally for maximum compatibility. After the design was finished, we 3D printed the parts using durable material for any application.

Table I: Hardware Components Table

Components	arduino	servo mg996r	servo sg90
Count	1	4	1

Then we moved on to the electronics part. We created a circuit to connect the servos and made sure the wires were put cleanly to prevent any altercations. The Arduino was chosen as our micro-controller connected to MATLAB to integrate our code with our model. The exact components are as shown in Table I. Figure 1 highlights connections of the hardware and electronics.

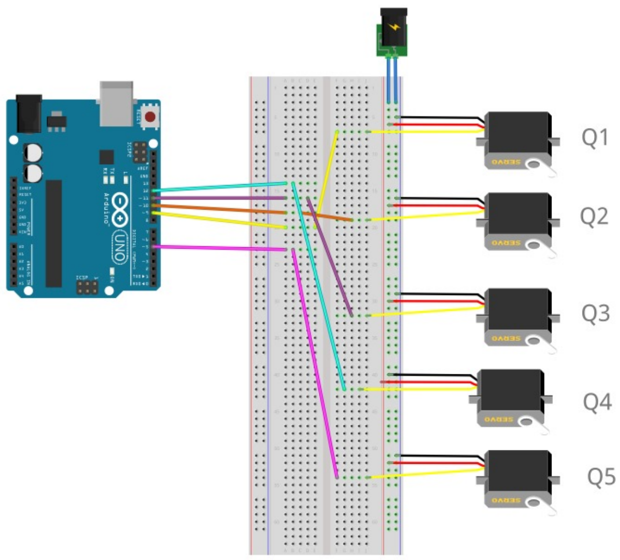

Figure 1: Circuit Diagram

III. Robot Kinematics

The robot's initial position, which is standing upright as shown in Figure 2, is defined as the zero point when we first set up the coordinate frames.

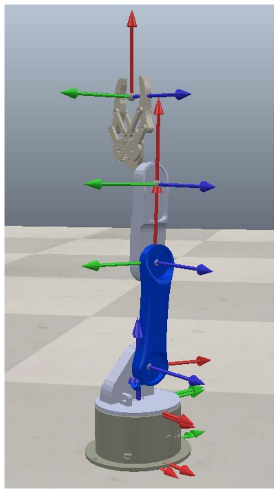

Figure 2: Frame Definition

Establishing the Denavit-Hartenberg (DH) protocol and allocating frames to each joint required this first step. Since the final two joints (q_4 and q_5) were mostly utilized for gripping and minor adjustments to move the end-effector (EEF) in place for object manipulation, we only concentrated on three of the five joints (q_1, q_2, and q_3) to control the location of the EEF. This setup allowed us to derive the transfer function.

$$T = \begin{bmatrix} \cos(q_2+q_3)\cos(q_1) & -\sin(q_2+q_3)\cos(q_1) & \sin(q_1) & P_1 \\ \cos(q_2+q_3)\sin(q_1) & -\sin(q_2+q_3)\sin(q_1) & -\cos(q_1) & P_2 \\ -\sin(q_2+q_3) & -\cos(q_2+q_3) & 0 & P_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

where:

$$\begin{aligned} P_X &= \frac{13\cos(q_1)}{1000} - \frac{3\sin(q_1)}{100} - \frac{3\cos(q_1)\cos(q_2)}{25} \\ &\quad + \frac{19\cos(q_1)\cos(q_2)\cos(q_3)}{100} - \frac{19\cos(q_1)\sin(q_2)\sin(q_3)}{100}, \\ P_Y &= \frac{3\cos(q_1)}{100} + \frac{13\sin(q_1)}{1000} - \frac{3\cos(q_2)\sin(q_1)}{25} \\ &\quad - \frac{19\sin(q_1)\sin(q_2)\sin(q_3)}{100} + \frac{19\cos(q_2)\cos(q_3)\sin(q_1)}{100}, \\ P_Z &= \frac{3\sin(q_2)}{25} - \frac{19\sin(q_2+q_3)}{100} + \frac{1}{25}. \end{aligned} \tag{2}$$

To calculate the EEF position for any given set of joint angles, we used Forward Position Kinematics. We wrote a MATLAB script that took the DH parameter found in table II and joint angles as input and calculated the EEF position using equations 2. The code then requires the user to enter joint angles to obtain the exact EEF coordinates .

Table II: DH-Parameters Table

Joint$_i$	θ_i (rad)	d_i(m)	a_i(m)	α_i (rad)
1	θ_1	0.04	0.013	$\pi/2$
2	θ_2	0	0.12	0
3	θ_3	0	0.09	0
4	0	-0.03	0.1	0

For the inverse position kinematics, we developed a code that takes the desired position of the EEF as input and calculates the joint angles needed to reach that position. We implemented this using the Newton-Raphson method. The Newton-Raphson equation (3)

$$\mathbf{q}^{(k+1)} = \mathbf{q}^{(k)} - J(\mathbf{q}^{(k)})^{-1}\mathbf{e}(\mathbf{q}^{(k)}) \tag{3}$$

We tested the output of this inverse kinematics code in CoppeliaSim. By entering the calculated joint angles into the simulation, we confirmed that the EEF's position matched our expectations, showing that the code was accurate.

To double-check, we ran the test in MATLAB's Simscape. We uploaded the model and compared the output with the results from our MATLAB code. Everything lined up perfectly, confirming that our approach was reliable.

Since CoppeliaSim works with Python scripting, we translated the code into Python to ensure compatibility and smooth operation.

Forward Velocity Kinematics was the next topic we covered. To compute the linear and rotational velocities of the EEF, we utilized the Jacobian matrix (4).

$$\begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} \tag{4}$$

where:

- $\mathbf{v}$ is the linear velocity of the EEF,
- $\boldsymbol{\omega}$ is the angular velocity of the EEF,
- J_v and J_ω are the linear and angular components of the Jacobian matrix, respectively,
- $\dot{q}_1, \dot{q}_2, \dot{q}_3$ are the angular velocities of the three joints.

We calculated the linear and rotational velocities of the EEF.

In order to confirm these findings, we calculated the forward velocities in MATLAB to compare them with the Simscape model outputs. The correctness of our forward velocity calculations was confirmed by the comparison, which revealed that the findings matched exactly.

Additionally, we created an inverse velocity kinematics code that determines the joint angular velocities required to reach the appropriate linear and rotational velocities of the EEF. We used the forward velocity kinematics output as the inverse code's input to confirm accuracy. The match confirmed that our implementation was accurate.

Finally, we focused on trajectory tracking in joint space. we made 4 trajectories one for each state of our pick and place operation. We computed the trajectories using equations (5) and (6).

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{5}$$

where:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & T_f & T_f^2 & T_f^3 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2T_f & 3T_f^2 \end{bmatrix} \right)^{-1} \begin{bmatrix} q_0 \\ q_f \\ \dot{q}_0 \\ \dot{q}_f \end{bmatrix} \quad \dot{q}_0 = \dot{q}_f = 0 \tag{6}$$

We input the robot's initial and final joint positions q_0 and q_f respectively and initial guesses for the joint angles, along with a sampling time to ensure smoother motion. The code then produced an array of joint angles, which we sent to the robot using Arduino to execute the trajectory.

To validate this, we fed the trajectory angles into our Simscape model and plotted the robot's motion on each axis. This plot was compared to the code-generated output, and we verified that the robot's movements in the simulation matched our expectations. We made sure the plots matched those produced by the code and that the robot's movements in the simulation were as expected.

We also implemented the trajectory planning code in Python and integrated it with CoppeliaSim to double-check that the robot's behavior matched our expectations before we tested it on the real servos.

IV. Simulation Results

In this section we will show the simulation results which are the 4 trajectories applied on both Matlab and CoppeliaSim.

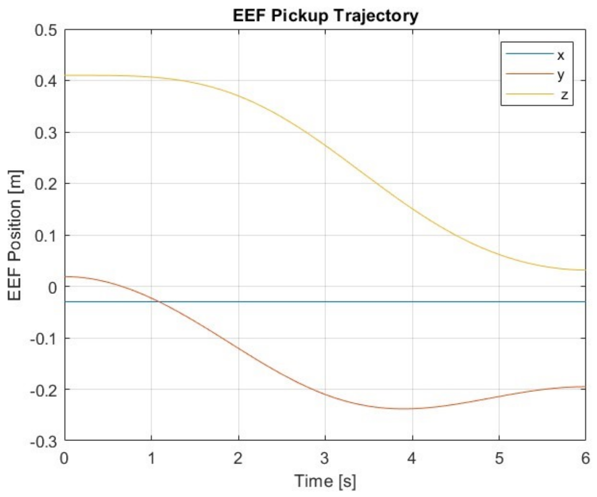

Figure 3: Trajectory 1

The first Trajectory which is the pick. the robot is at Zero Position (at rest) and it moves towards the object it will pick as shown in figure 3.

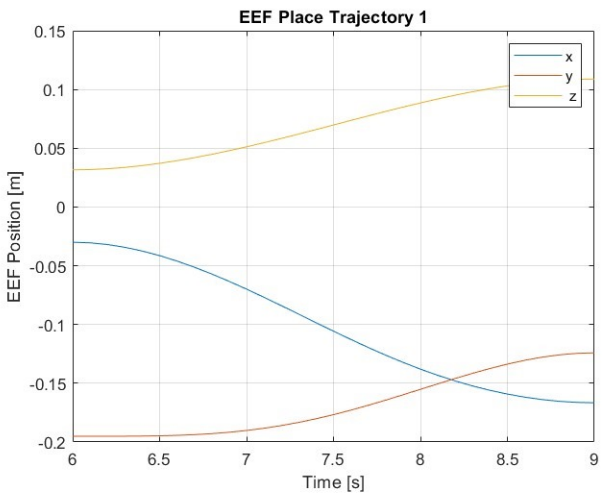

Figure 4: Trajectory 2

The Second Trajectory which is the first half of the place. the robot is at the object's Position holding it and it moves upwards so it doesn't drag the object on the surface as shown in figure 4.

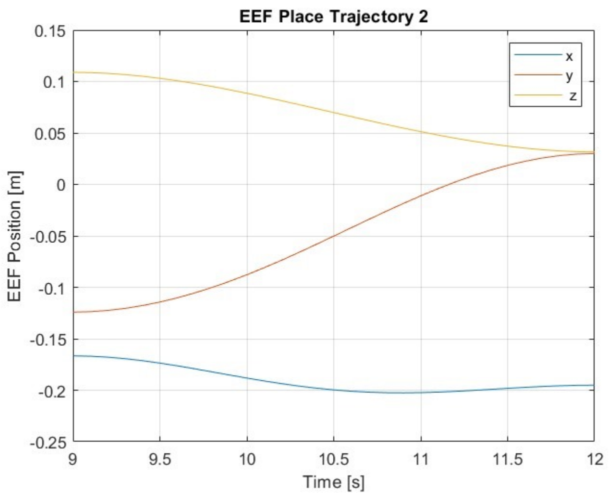

Figure 5: Trajectory 3

The Third Trajectory which is the second half of the place. the robot is holding the object in the air and it moves towards the place position as shown in figure 5.

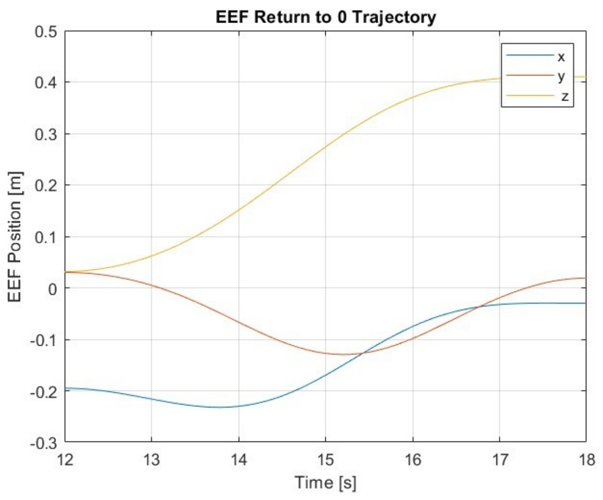

Figure 6: Trajectory 4

Finally, The Fourth Trajectory which is back to rest. the robot has placed the object and it moves towards its zero position as shown in figure 5.

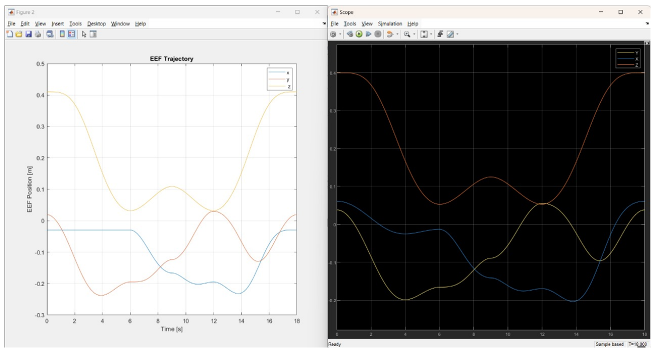

Figure 7: Full Trajectory Matlab and SimScape

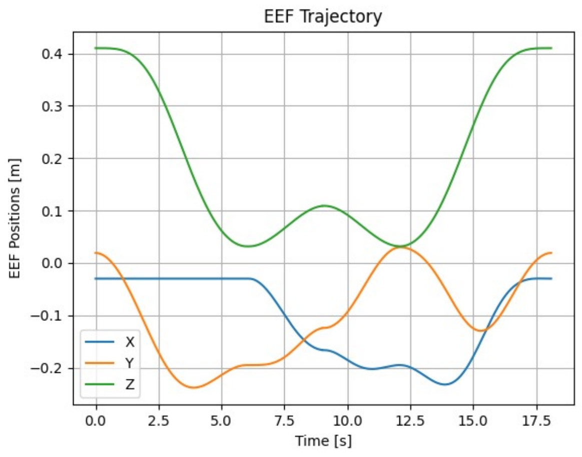

Figure 8: Full Trajectory CoppeliaSim

Figures 7 and 8 shows all the 4 trajectories combined in 1 trajectory which shows the full movement of the robotic arm in both CoppelliaSim, Simscape and Matlab. For a video visualization of the results scan QRcode in Fig 12 to be able to see the video which will be named Video1.

now to show the real life testing of our robot.

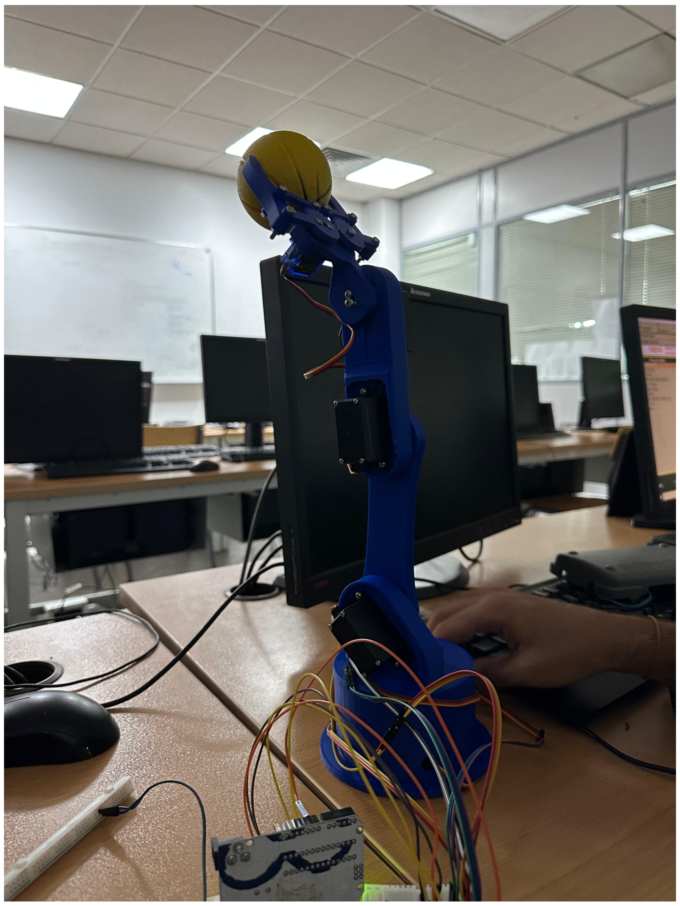

Figure 9: Robot holding object mid air

Fig 9 shows that the robot holding the object mid air. For a video visualization of the Both Robotic arms operating scan QRcode in Fig 12 to be able to see the video which will be named Video2. now to show our robot performing a collaborative task. Serial connection between the two robots is established using Arduino and MATLAB, with dedicated pins for the receiver (RX) and transmitter (TX). The process begins with the pick-and-place robot waiting for a high signal from the piercing robot to start the picking and putting phase. When the pick-and-place robot receives the high signal, it responds with a high signal, followed by a low signal to indicate the start of the procedure. After finishing the picking and putting phase, the pick-and-place robot transmits a high signal to the piercing robot, which acknowledges by giving back a high signal before the signal is pulled low, triggering the start of the piercing trajectory. Once the piercing task is finished, the piercing robot retracts to its home position and transmits a high signal to the pick-and-place robot, signaling it to proceed to its final destination. After successfully placing the object, the pick-and-place robot returns to its home position and sends a high signal to the piercing robot, acknowledging it and restarting the cycle.

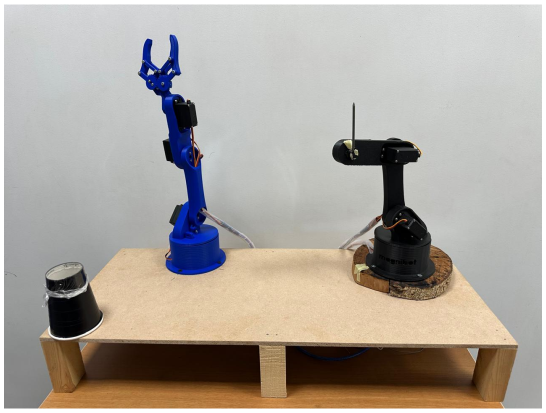

Figure 10: Zero position of both robots

Fig 10 show both robots at their zero positions before starting their tasks the task is pick and place and keep holding then pierce then pick and place and leave.

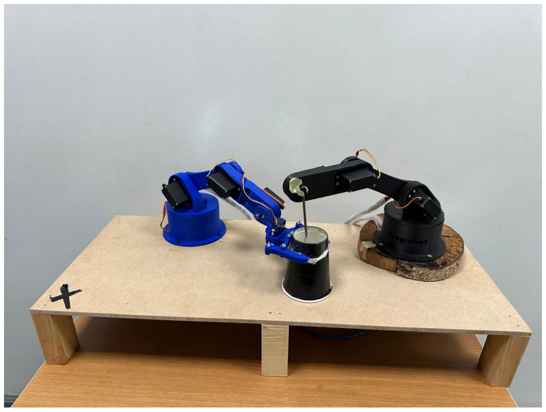

Figure 11: Zero position of both robots

Fig 11 shows the robots mid operation after the robot placed the object and is holding it while the other robot is piercing it.

Figure 12: QR for Videos

For a video visualization of the Both Robotic arms operating scan QRcode in Fig 12 to be able to see the video which will be named Video3.

V. Conclusions and Future Recommendations

In conclusion, this work demonstrated how to efficiently use MATLAB and Python to plan and control joint-space trajectories for a robotic arm. In order to move the arm's end-effector between these spots, we had to define the robot's starting, middle, and final positions, determine the joint angles required for each position, and design smooth routes. By using MATLAB for trajectory planning and visualization, and Python for data handling and analysis, we created a strong, integrated approach for robotic motion control.

We also faced and solved challenges, like troubleshooting unexpected data dimensions and making sure the two programming environments could work together smoothly. These solutions are crucial for making sure the system operates as expected.

Overall, this work is a solid starting point for more complex robotics projects. It lays the groundwork for future work in real-time control, adaptive strategies, and more advanced motion planning. The methods we've developed can be used to create more sophisticated and precise robotic systems, whether in research or real-world applications.

References

[1] M. Farman, M. Al-Shaibah, Z. Aoraiath, and F. Jarrar, "Design of a three degrees of freedom robotic arm," *International Journal of Computer Applications*, vol. 179, pp. 12–17, 04 2018.

[2] T. Younas, M. Khan, S. Urooj, N. Bano, and R. Younas, "Four degree of freedom robotic arm," 12 2019, pp. 1–4.

[3] Z. Dongxu, R. Jia, and M. Xie, *Mirobot: A Low-Cost 6-DOF Educational Desktop Robot*, 01 2022, pp. 189–200.