# Fabrication of A 3D-Printed Robotic Arm With Kinematic Modeling For Pick-And-Place Operations

## 1. Problem Statement

Modern robotic arms are integral in tasks ranging from industrial assembly to medical procedures. This project aims to create a low-cost, programmable, and anthropomorphic robotic arm capable of performing basic pick-and-place operations. By integrating servo-based actuation and a Raspberry Pi controller, the goal is to implement fundamental kinematic principles—both forward and inverse kinematics—to enable the arm's end-effector to reach designated positions accurately within its workspace. Concepts learned during MAE 547—such as Denavit-Hartenberg (DH) parameterization, workspace analysis, forward and inverse Kinematics—are now integrated into a tangible system. By building this initial prototype, we not only reinforce our understanding of these core robotic principles but also create a platform from which we can evolve our work. Future improvements, such as introducing closed-loop feedback, advanced trajectory planning, and more complex inverse kinematic solutions, are natural next steps as we continue to grow our expertise beyond the scope of the current class.

**Objective**

- Design, fabricate, and program a simple 3-DOF robotic arm (plus a gripper) using a Raspberry Pi and servo motors to perform pick-and-place operations.
- Develop and utilize a defined configuration space to understand the arm's feasible workspace.
- Apply forward and inverse kinematics for precise positioning of the end-effector.
- Validate the system through simulations and real-world tests, and assess accuracy, repeatability, and reliability.

## 2. Approach

### 2.1. Initial Design Rationale and Assumptions

The foundational concept for this robotic arm design was to strike a balance between mechanical simplicity and meaningful complexity. By opting for a three-degree-of-freedom (3-DOF) configuration—consisting of a base rotation, a shoulder joint, and an elbow joint—plus an additional servo-driven gripper, the design remains manageable while still demonstrating fundamental robotic motion principles. The relatively low dimensionality of this system limits the complexity of the inverse kinematics calculations and controller logic, making it an ideal educational platform for exploring core concepts in robotics without the overhead of more intricate multi-joint systems.

Figure 1. Side view of arm

We also decided to simplify the inverse kinematics problem by orienting the base joint so that the target point lies in a plane shared by the shoulder and elbow joints. Reducing what could be a 3-DOF inverse kinematics challenge to a 2-DOF problem ensures that analytical solutions remain tractable. This assumption provides a clearer path to success for initial testing and demonstration, serving as a foundation that can be expanded as future improvements—such as adding more degrees of freedom or sophisticated sensors—are introduced.

### 2.2. Hardware Setup

The following components were utilized in manufacturing the robotic arm:

1. **Raspberry Pi:** Serves as the primary controller, running Python scripts to generate PWM signals and execute kinematic calculations.
2. **Servo Motors (3 x Hitec MG5465 + 1 x Micro Servo):** Provide actuation at the base, shoulder, elbow, and gripper joints. The MG5465 servos handle larger loads, while the micro servo suffices for the gripper's lighter task.
3. **Power Supply:** A stable DC power source capable of delivering 5 V at 5 A to handle peak currents drawn by servos under load.
4. **Robot Arm Structure:** Fabricated using PLA and PETG for a balance between rigidity and weight. Link lengths were chosen to ensure the applied torques would

be within the servos' acceptable limits. The gripper's jaws use parallel links attached to the micro servo to actuate.

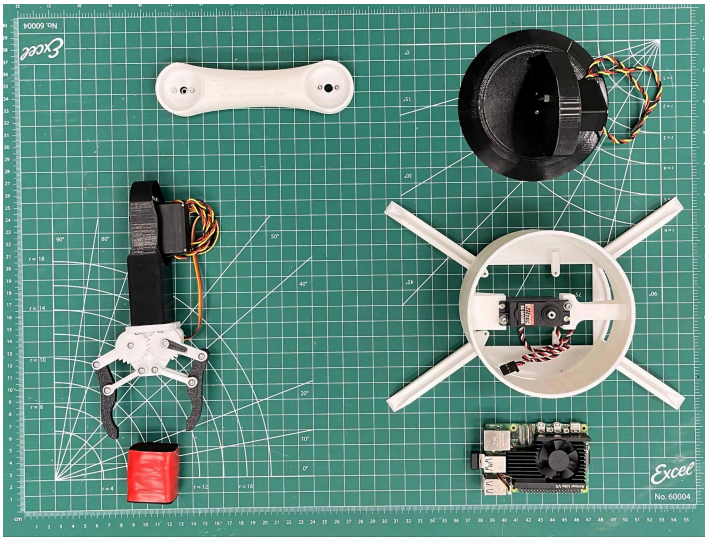


Figure 2. Hardware setup of-DOF Anthropomorphic Robotic Arm

All the links and gripper parts were designed in SolidWorks and a basic assembly and motion study was performed to demonstrate all the parts assembled and joint movements. The figure below shows the SolidWorks Assembly:

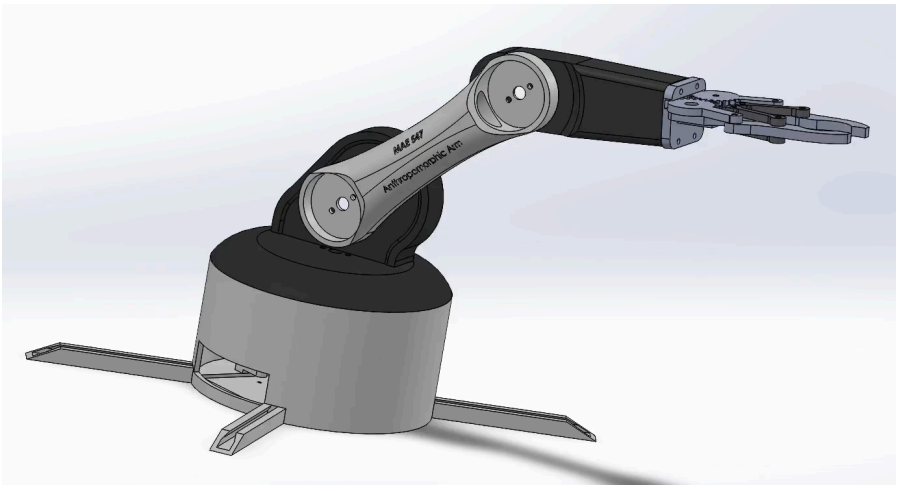


Figure 3. SolidWorks Assembly

The assembly and initial testing of the robotic arm follows this order:

1. **Mechanical Assembly:** Each servo is secured in place using bolts and heat set inserts. Links then attach to the servo's axle using bolts. The gripper is attached to the forearm using bolts and heat set inserts as well.
2. **Electrical Connections:** Servos are wired to the Raspberry Pi's GPIO pins, ensuring common ground and stable power. An external power supply provides the power to drive the servos.
3. **Initial Testing:** Each servo was tested to check that it responds to basic PWM signals. We then attached the links to the servo axle and verified that the arm indeed reaches the end points of the calculated workspace.

### 2.3. Configuration Space and Workspace Definition

#### 2.3.1. Joint Angle Limits and Servo Constraints

While the servos are rated for approximately 180° rotation, practical testing revealed an effective range of ~160° due to mechanical constraints and non-linearities. The Raspberry Pi's PWM resolution and servo response also affect achievable accuracy, potentially limiting the smallest repeatable angle increment to ~2°.

#### 2.3.2. Defining Joint Angles

Let $\theta_1$ be the base rotation angle (about z-axis), $\theta_2$ the shoulder angle (pitch), and $\theta_3$ the elbow angle (pitch). The gripper ($\theta_4$) is a simple open/close command rather than a continuous angle. Approximate ranges:

- $\theta_1$: 0° to 180°
- $\theta_2$: 0° to 180°
- $\theta_3$: -90° to 90°

#### 2.3.3. Workspace Boundaries

By considering link lengths and joint limits, the maximum horizontal reach is about 30cm, and vertical reach is about 35cm.

Simulation tools through matlab help visualize reachable points. Preliminary simulations confirm that the planned pick-and-place tasks (e.g., moving small items from a bin to a target area) are achievable within these bounds

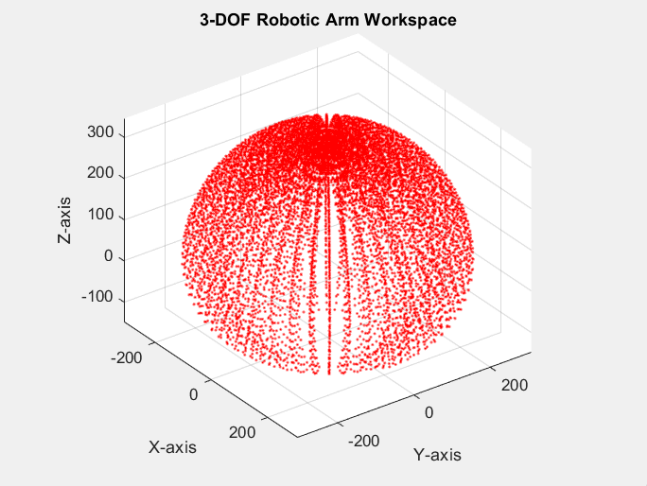


Figure 4. Workspace of the Robotic Arm

### 2.4. Kinematics Implementation

Using the Denavit-Hartenberg (DH) Convention with the shoulder joint as the base frame we get the following DH Table:

| Link No | a | α | d | θ |
|---|---|---|---|---|
| 1 | 0 | π/2 | $L_1$ = 38.55 mm | $\theta_1$ |
| 2 | $L_2$ = 120 mm | 0 | 0 | $\theta_2$ |
| 3 | $L_3$ = 187.75 mm | 0 | 0 | $\theta_3$ |

From these parameters, construct homogeneous transformation matrices and multiply sequentially to get the end-effector frame. In code, given ($\theta_1$, $\theta_2$, $\theta_3$), we can quickly compute x, y, z coordinates.

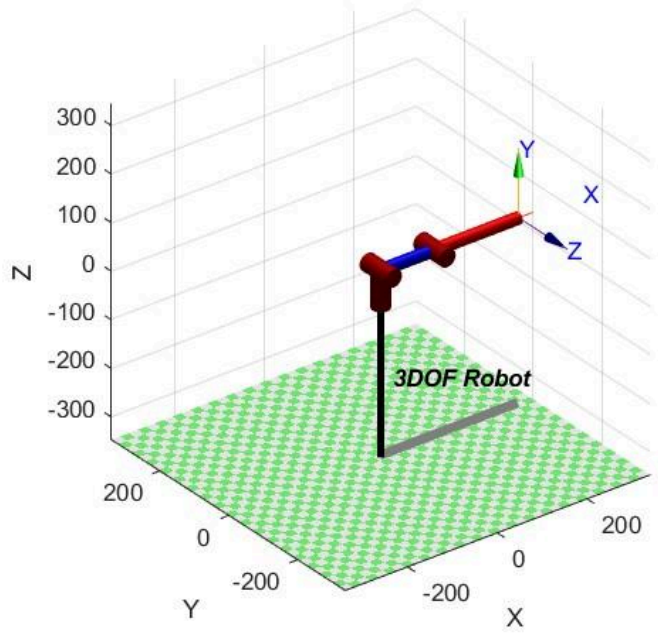


Figure 5. Initial position of Robotic arm created using DH table

### 2.4.1. Inverse Kinematics

Without simplification, a 3-DOF robotic arm's inverse kinematic equations can yield multiple solutions (up to four sets of angles), some of which may be physically unattainable or cause joint collisions. By pre-aligning the base joint so that the target lies within a vertical plane, the problem reduces to solving a planar 2D inverse kinematics problem for the shoulder and elbow. Standard trigonometric relations (law of cosines, sines) yield closed-form solutions for $\theta_2$ and $\theta_3$.

***Approach:***

- Compute projection of target point in the plane defined by $\theta_1$.
- Use geometry to find $\theta_2$ and $\theta_3$ angles that achieve the desired (x, y) position.
- If multiple solutions exist, select the one that is within permissible joint values.

With IK implemented on the Raspberry Pi, given a target (x, y, z), we solve for $\theta_1$ (base turn), then $\theta_2$ and $\theta_3$, and finally command the servos.

## Pseudocode:

```
IK(pWx, pWy, pWz, L1, L2, L3):

BEGIN

# Moving pWz from frame 1 to frame 2

pWz = pWz - L1; # This is to make calculations easier

# We find possible values for theta_1 and pick the one that is valid for our  #
joint-space

theta1_1 = atan2(pWy, pWx);    # First solution for theta_1

theta1_2 = atan2(-pWy, -pWx);  # Second solution for theta_1

# Since theta_2 and theta_3 are dependent on each other, we calculate them    #
together

c3 = (pWx^2 + pWy^2 + pWz^2 - L2^2 - L3^2) / (2 * L2 * L3);

# We have two possible values for sin(theta_3) so we check both

s3_pos = sqrt(1 - c3^2);

s3_neg = -s3_pos;

# Now we use the two sine values to find the two possible valid theta_3 values

theta3_1 = atan2(s3_pos, c3);  % First solution for theta_3

theta3_2 = atan2(s3_neg, c3);  % Second solution for theta_3

# Finally, we use the theta_3 values to find the theta_2 values

theta2_1 = atan2(pWz,sqrt(pWx^2 + pWy^2)) - atan2((L3*sin(theta3_1)),(L2 +
L3*cos(theta3_1)));

theta2_2 = atan2(pWz,sqrt(pWx^2 + pWy^2)) - atan2((L3*sin(theta3_2)),(L2 +
L3*cos(theta3_2)));

# We need a combination of theta_2 and theta_3 values where both are within   #
joint limits

IF theta2_1 >= 0 && theta2_1 <= pi && theta3_1 >= -pi/2 && theta3_1 <= pi/2
THEN

      theta2 = theta2_1;
```

```
      theta3 = theta3_1;

ELSEIF theta2_2 >= 0 && theta2_2 <= pi && theta3_2 >= -pi/2 && theta3_2 <= pi/2
THEN

      theta2 = theta2_2;

      theta3 = theta3_2;

ELSE # If this happens, both theta_2 and theta_3 are invalid

      Return

ENDIF

# Finally, we pick the theta_1 value. If the target is in the positive half of
# the robot-target plane, we can use the theta_2 and theta_3 values we have   #
already found. If the target is in the negative half, we need to change     #
theta_2 and theta_3 to accommodate for this.

IF theta1_1 >= 0 && theta1_1 <= pi THEN

      theta1 = theta1_1;

ELSEIF theta1_2 >= 0 && theta1_2 <= pi THEN

      theta1 = theta1_2;

      theta2 = pi-theta2; # New theta_2 becomes pi-theta_2

      theta3 = -theta3; # New theta_3 becomes -theta_3

ELSE

      return;

ENDIF

Return [theta1, theta2, theta3]

END
```

### 2.5 Raspberry Pi (Python) Code Implementation

#### 2.5.1. Libraries and Setup

To control the servos and perform kinematic calculations, several Python libraries and modules are used:

- **`helper.servo`**: A custom module containing functions to abstract servo angle-to-PWM conversions, initialization routines, and servo actuation commands.
- **`time.sleep`**: Pauses the code execution to allow for servo movement and timing-based operations.
- **`numpy`** (`np`): Provides mathematical functions, array operations, and trigonometric utilities required for inverse kinematics calculations.
- **`gpiozero.pins.lgpio` and `lgpio`**: Low-level libraries enabling precise and reliable GPIO pin control, facilitating stable PWM signals for the servos on the Raspberry Pi.

#### 2.5.2. Controls Implementation

- **Class Robot:** This class is defined in the file main.py. The purpose of this class is to hold information about the robot, i.e. the link lengths, that we need to perform the inverse kinematics operation, which also lies in this class.
- **Inverse Kinematics Function:** Input desired (x, y, z), compute joint angles. This computation happens in main.py, under the class Robot. It gives a valid combination of joint angles that can be attained by the robot within joint limits.
- **Command Servos:** Convert joint angles to corresponding PWM duty cycles. Account for servo offset calibration (e.g., $\theta_2 = 0$ might not correspond to a 1.5 ms pulse precisely). This code is defined in helper/servo.py.

#### 2.5.3. Testing and Calibration

- **Static Points:** Move the arm to a series of predefined points along known coordinates. Measure the end-effector position with a ruler or a camera-based system.
- **Iterative Adjustments:** Apply small angle offsets to correct for mechanical misalignments. Document final calibration values.
- **Repeatability Trials:** Run the arm to the same coordinate multiple times to assess positional repeatability and servo backlash effects.

## 3. Results: Simulation and Results Visualization

### 3.1. Simulation Results

Plots generated in MATLAB or Python show a dense cluster of achievable end-effector positions, forming a workspace region. Example target points (e.g., [placeholder: (10 cm, 5 cm, 0 cm)]) yield valid solutions.

FK validation: Input known joint angles into the FK model and compare the computed position against manually measured positions. Deviations of less than [placeholder: e.g., 5 mm] were observed in most tested cases.

### 3.2. Real-World Testing

The arm successfully placed small objects (e.g., 30 by 30mm plastic block) from a start position to a target area. Video evidence shows the arm rotating the base servo to face the target and adjusting the shoulder and elbow to reach it.

The gripper, though simple, reliably opened and closed around small, lightweight objects without significant slip.

### 3.3.Inverse Kinematics Testing

Several random target points were chosen, and their corresponding joint angles were computed via IK. Running FK on these angles returned target coordinates. This closed-loop verification in simulation and in practice confirmed that our kinematic models were sound.

## 4. Discussion

### 4.1. Analysis of Discrepancies

While simulation results indicated near-perfect motion, real-world tests introduced non-idealities:

- Servo jitter and limited PWM resolution led to minor, unpredictable position errors.
- Mechanical tolerances and slight misalignments between joints caused the end-effector position to deviate from ideal values.

### 4.2. Challenges and Limitations

- Limited servo resolution: Although commanded for full rotation and precise increments, actual movement fell short of the theoretical range.
- Backlash: Gearing within the servos introduced small position errors that weren't fully modeled.
- Static open-loop control: Without feedback sensors, the system cannot correct positional errors that accumulate over time.

### 4.3. Future Improvements

- **Closed-Loop Feedback:** Adding encoders or Hall-effect sensors would allow the system to measure actual joint angles and correct errors in real-time.
- **Enhanced Trajectory Planning:** Instead of simple point-to-point moves, implement trajectory planning (e.g., cubic spline interpolation) to achieve smoother and more predictable motions.
- **Increased DOFs:** Adding a wrist joint or more complex gripper could expand the range of achievable tasks and test more advanced IK algorithms.
- **Material and Mechanical Refinement:** More rigid materials and precision machining could reduce tolerances and enhance positional accuracy.

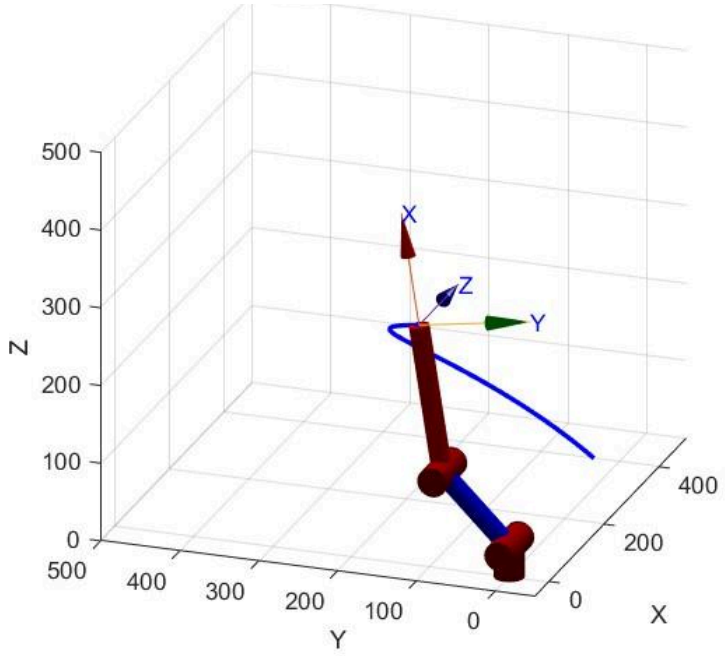


Figure 6. 3-DOF Robot Trajectory Simulation With Dynamic Path

## 5. References

Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2010). *Robotics: Modelling, Planning and Control*. Springer Science & Business Media.

Petercorke. (n.d.). *GitHub - petercorke/robotics-toolbox-matlab: Robotics Toolbox for MATLAB*. GitHub. https://github.com/petercorke/robotics-toolbox-matlab

# Contribution Claim in Group No. 14

| **Member Name** | **Role in the project**<br>**(Clearly describe each team member's role for the project)** | **Contribution percentage**<br>**(The sum of all members' contributions should be 100%)** |
|---|---|---|
| Sukhpreet Singh Nolastname | Designing and Printing 3D model of a Robotic Arm | 20 |
| Vishavjit Singh Khinda | Formulating Analytical Inverse Kinematic solution | 20 |
| Vinamr Arya | Designing and Printing 3D model of a Robotic Arm | 20 |
| Vatsin Ninad Shah | Using Python to implement Algorithms on the Raspberry pi | 20 |
| Dilli Babu Kalluru | Calculating Workspace of the robotic arm | 20 |

Signature by all group members:

Sukhpreet, Vishavjit, Vinamr, Vatsin, Dilli Babu

(By signing, it means you have made the consensus about everyone's contribution percentage)