Kinematics Robotics Arm Project

Kinematics and Control of Robotic Systems

Benjamin “Ben” Sedmont,

Alexander “Alex” Streck,

William “Billy” Roche,

11/19/2025

Table of Contents

1. Introduction

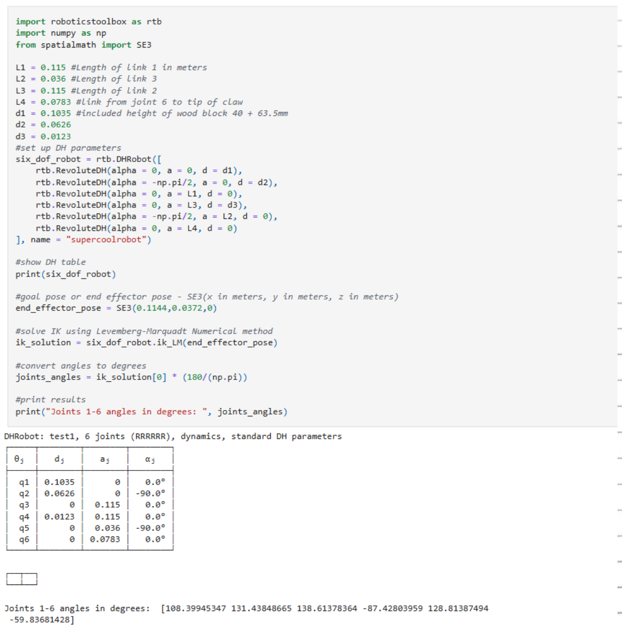
For this project, we were tasked with building and coding a robot arm with 6 degrees of freedom (DoF). The robot would utilize 6 servo motors powered through an external supply and controlled via an Arduino Uno. Robot links would be created with 3D printing and joined using various screws and servo kit horns. Assembly was performed following online guides. Once assembled, the robot would undergo motion testing with coding for Bluetooth connectivity through an android app. The app would allow for user defined motion in real time from array of control sliders. After motion testing was finished, the Bluetooth elements were removed from the circuitry and improvements were made to the structural stability of the robot. A new focus was given to predefining a path for the robot to follow and accomplish a desired action. The robot was analyzed using inverse kinematics, and a Denavit-Hartenberg (DH) table was constructed for the joints and gripper. Using Python libraries, the DH table helped to create T-matrixes for the various positions of the robot. These T-matrixes were applied to an Arduino IDE motor control code that would be uploaded to our physical system. Physical testing was performed using the given challenge layout and the goal object rubber duck. Gradually modifying the motor control program to reach all desired positions, we were able to successfully set a predefined path for the arm to move in.

2. Bill of Materials

- 3D-Printed Parts (including base, shoulder, arm, grippers, and wrist pinch and roll pieces) (Qt. 14)
- MG996R Servo Motors (Qt. 3)
- SG90 Micro Servo Motor (Qt. 3)
- Arduino Board (Qt. 1)
- 5V 2A DC Power Supply (Qt. 1)
- Block of Wood (for stability) (Qt. 1)
- Breadboard and Arduino wires (Qt. 20 minimum)
- Rubber Ducky (Qt. 1)
- Checkerboard Paper (Qt. 1)
- Screw's (Qt. 35, the more the merrier)

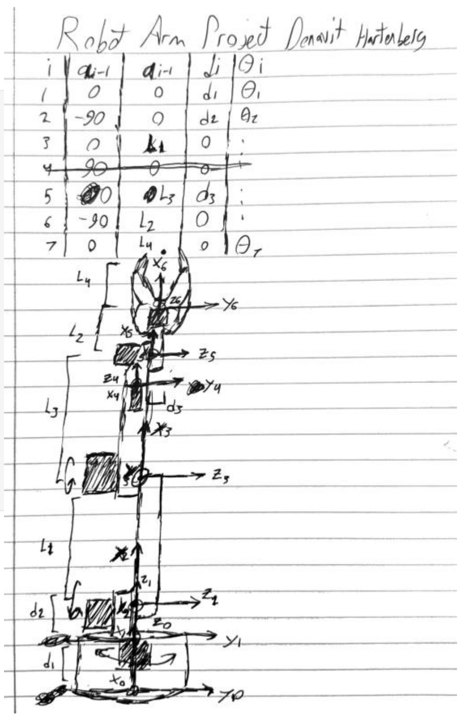
3. Methods and Procedures

To build this robot, you first must acquire every material before starting, as it will get hectic when you initially start physical construction. First, we removed the support filament off the 3D-printed parts. An extra step is needed for this since some support is printed inside the wire guides. The wire guide's length and tight space make removing support difficult, so to make it easier, the print is put in hot water until it softens. Before attaching the servos to their respective spots, make sure to test them all and see that they are all functioning. After testing the servos, we then proceeded to build the robot. For instance, place the base of the robot on a solid foundation (like a piece of wood) and screw it in. Next, attach the base servo inside of the base part and screw it in place. After that, place a servo horn on the "waist" piece and connect it to the base servo. Then place the waist servo on the waist piece, screwing it in. Attach a servo horn onto the link between the shoulder and elbow joints and connect it to the waist servo. Then attach a servo onto the other side of the elbow joint and place the next link with the wrist piece. Moving on to the smaller servos, you want to place one inside of the upper arm of the robot and screw it in. We then placed a servo horn onto the wrist joint between the upper arm and hand links and attached the servo to the horn, connecting the two links. After that we placed another servo onto the wrist link to connect it to the gripper link. We then placed the last servo horn on the gripper link to connect the two links together. Lastly, we screwed on the last servo to open and close the claws and screwed the actual claw pieces together. Once the robot was complete, we then attached the wires to the breadboard and Arduino, securing the connection between hardware and software. At this point we started the software portion of the project and began working on the code to move the robot around.



```python
import roboticstoolbox as rtb
import numpy as np
from spatialmath import SE3

L1 = 0.115 #Length of link 1 in meters
L2 = 0.036 #Length of link 3
L3 = 0.115 #Length of link 2
L4 = 0.0783 #link from joint 6 to tip of claw
d1 = 0.1035 #included height of wood block 40 + 63.5mm
d2 = 0.0626
d3 = 0.0123
#set up DH parameters
six_dof_robot = rtb.DHRobot([
    rtb.RevoluteDH(alpha = 0, a = 0, d = d1),
    rtb.RevoluteDH(alpha = -np.pi/2, a = 0, d = d2),
    rtb.RevoluteDH(alpha = 0, a = L1, d = 0),
    rtb.RevoluteDH(alpha = 0, a = L3, d = d3),
    rtb.RevoluteDH(alpha = -np.pi/2, a = L2, d = 0),
    rtb.RevoluteDH(alpha = 0, a = L4, d = 0)
], name = "supercoolrobot")

#show DH table
print(six_dof_robot)

#goal pose or end effector pose - SE3(x in meters, y in meters, z in meters)
end_effector_pose = SE3(0.1144,0.0372,0)

#solve IK using Levemberg-Marquadt Numerical method
ik_solution = six_dof_robot.ik_LM(end_effector_pose)

#convert angles to degrees
joints_angles = ik_solution[0] * (180/(np.pi))

#print results
print("Joints 1-6 angles in degrees: ", joints_angles)
```

```
DHRobot: test1, 6 joints (RRRRRR), dynamics, standard DH parameters
┌────┬────────┬────────┬────────┐
│θⱼ  │   dⱼ   │   aⱼ   │   ⍺ⱼ   │
├────┼────────┼────────┼────────┤
│ q1 │ 0.1035 │      0 │   0.0° │
│ q2 │ 0.0626 │      0 │ -90.0° │
│ q3 │      0 │  0.115 │   0.0° │
│ q4 │ 0.0123 │  0.115 │   0.0° │
│ q5 │      0 │  0.036 │ -90.0° │
│ q6 │      0 │ 0.0783 │   0.0° │
└────┴────────┴────────┴────────┘

┌─┬─┐
└─┴─┘

Joints 1-6 angles in degrees:  [108.39945347 131.43848665 138.61378364 -87.42803959 128.81387494
 -59.83681428]
```



Robot Arm Project Denavit Hartenberg

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	d_1	θ_1
2	-90	0	d_2	θ_2
3	0	L_1	0	:
~~4~~	~~90~~	~~0~~	~~0~~	
5	0	L_3	d_3	:
6	-90	L_2	0	:
7	0	L_4	0	θ_7

Figures 1 & 2: Inverse Kinematic python solver (left), DH table and diagram (right)

To calculate the inverse kinematics, we first calculated the forward kinematics. We accomplished this by finding the DH table parameters on paper. Following this, we wrote an inverse kinematics solving program in Python using the Robotics Toolbox by Peter Corke. This program uses the DH table to solve the inverse kinematics using a built-in algorithm if it is provided with a wanted position. After this, the program gives joint angles that we proceeded to plug into our Arduino program.

The Arduino program is relatively simple and uses a loop function, a setup function, and a home function. The program starts by homing the robot in a straight-up position. The home function uses for loops to slowly increment the joint angles from their current angle until they reach the desired angle. The loop function works pretty much the same, except there are two positions. Position 1 is the location of the rubber ducky, and position 2 is the location of the box. The loop function also uses for loops to move from the home position to position 1, then to position 2, and then finally it homes again. This action will be repeated until the robot is unplugged.

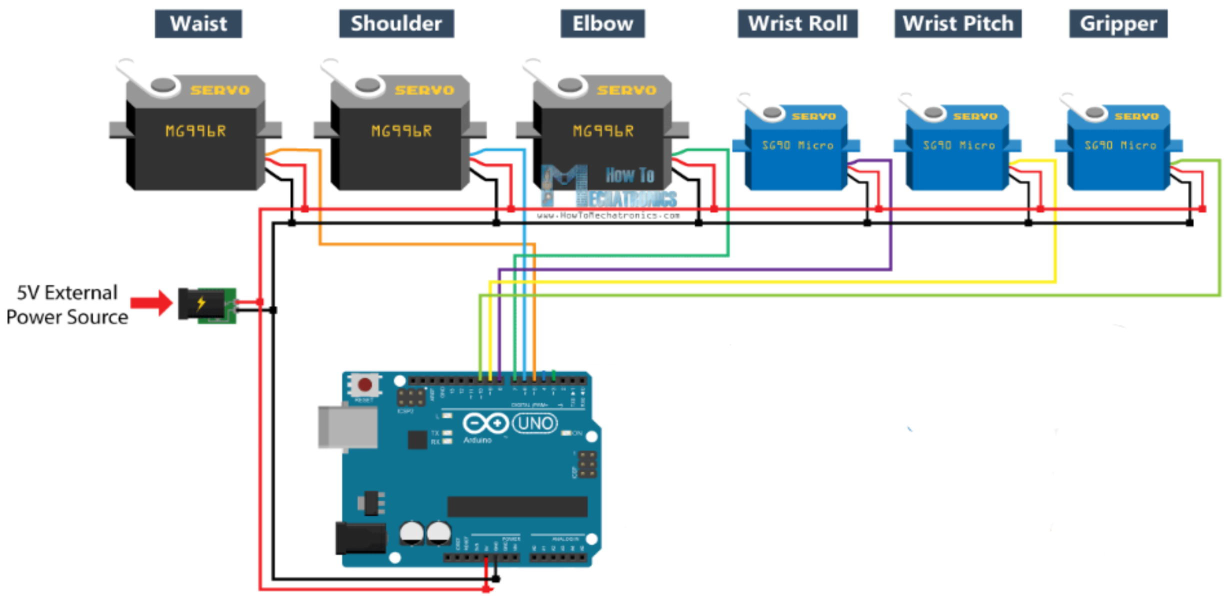


Figure 3: wiring diagram

4. Results and Discussions

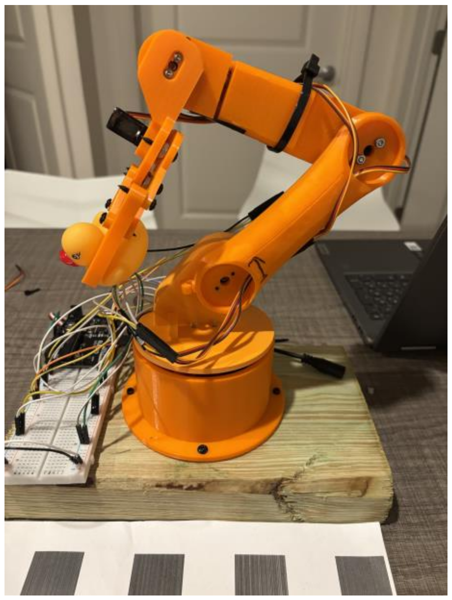


Figure 4: fully assembled robot with the duck in its claw

Primary assembly of the robot came rather quickly once all desired parts were present. A slight issue was found in the "waist" of the robot, where a gap causing imbalance would be created by the design of the servo placement and its interaction with the plastic servo horn. This issue had no simple fix and would instead be worked around in the motion control code.

Further physical issues appeared with the plastic servo horns where they would get stripped after some usage, and the motors could no longer grip them. The fix for this involved using more appropriate screws with the horns and ensuring their tightness before operation. With these issues resolved, physical testing would conclude with overall success.

During initial testing of the electronics, a mistake was made where power from our battery pack went into the Arduino we had connected and fried the board. This would be prevented in later testing by removing common power but keeping common ground for stability. Further issues were found with some of our available motors not responding properly or displaying no motion at all. These motors would be swapped out, and wiring would be simplified to allow for easier circuit access. Fixing the mentioned problems and refining our circuitry allowed for initial electronics testing to conclude successfully.

In testing with the Bluetooth module and control app, initial issues were found in achieving stable connection while running the app. Trying different app setups provided improved stability but physical motion would still involve some rather abrupt and unintended errors. Overall, Bluetooth testing would conclude with successful app control and minor faults in the fluidity of motion.

Creation of the DH table and resulting T-matrixes for automated motion using inverse kinematics appeared to be straightforward, but actual implementation of our work proved difficult. Initial testing with inverse kinematics produced results where the robot arm moved randomly upon code being uploaded, only being able to carry out parts of what was programmed. The cause of this behavior was uncertain as physical circuitry had not changed, and it seemed to happen regardless of code. It is possible that the DH table or resulting T-matrixes were flawed, our method of uploading code was improper, or our code itself had errors, among other things. It was ultimately decided that we would use the inverse kinematics solution for understanding and direct coding measures for the competition to accelerate our testing.

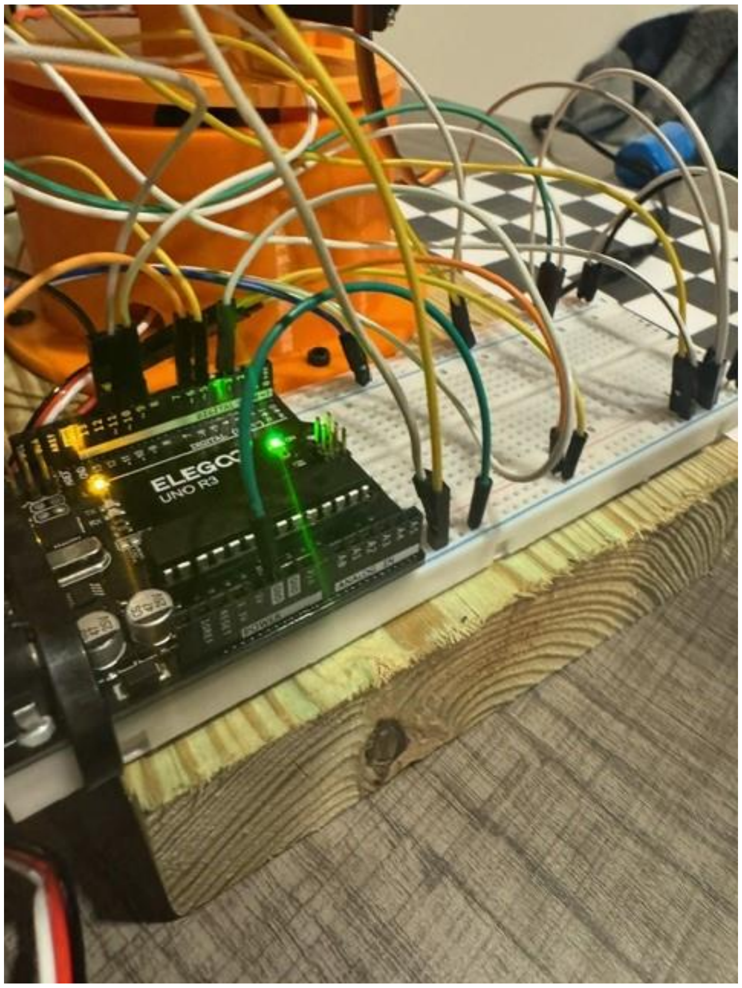


Figure 5: circuitry close-up

Final motion control for the competition would present some challenges involving stability, speed, and physical limitations of the arm. Initial testing showed that objects picked up by the smooth claw could easily fly out due to motion being too fast and inconsistent. This issue was largely resolved by ensuring proper battery power before testing, using ‘for’ loops to slow specific joint motions, and reducing unnecessary movements. To manage the odd shape of the toy duck with the claw we were using, the arm would be coded to approach from above and grab the duck by its head before moving it to the shoebox. The arm was set up to start by point straight upwards and using that as a reference for later motion. This testing would conclude successfully with the robot, in normal circumstances, being able to grab the duck from a set position and place it where we wanted for as long as we allowed.

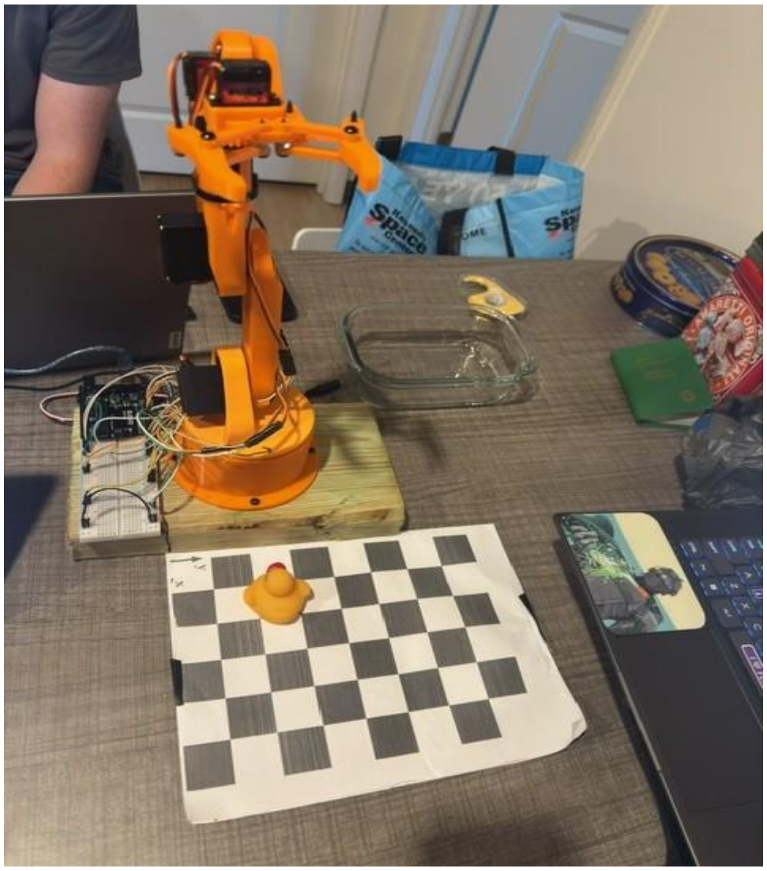


Figure 6: simplified competition layout

5. Conclusion

In essence, this project exercised a good understanding of designing and implementing kinematics on a robot, through both physical and digital representations. Each aspect of the project enhanced our understanding of inverse kinematics, physical assembly, and learning the coding software of an Arduino. The physical construction of the robot arm, breadboard wiring, and Arduino wiring improved our skills in wire assembly, optimization, and physical testing of the robot. The digital representation helped sharpen our coding in Arduino and Python for the inverse kinematics.

6. Github Repo + Demo Video

GitHub - williamjroche/6-DOF-Robotic-Arm: Kinematics and Control of Robotic Systems class project

Demo Video - https://youtube.com/shorts/FU3pKQ7Fjlg?feature=share

7. Acknowledgements

We would like to acknowledge Dr. Ngo and thank him for his help and support during the project duration. We would also like to thank the author of the articles for the robot arm project, Mr. Dejan, for providing the instructions and overall project itself.

8. References

Initial Robot Project: DIY Arduino Robot Arm with Smartphone Control - How To Mechatronics

Updated Robot Project: Arduino Robot Arm and Mecanum Wheels Platform Automatic Operation

Peter Corke Robotics Toolbox: GitHub - petercorke/robotics-toolbox-python: Robotics Toolbox for Python