Crazy Shot

Team 4 Final Project Report – Fall 2024

Michael Darren Doyle
Biomedical Engineering Graduate Program
MichaelDoyle@gatech.edu

Sofia Fraija Vaca
Biomedical Engineering Undergraduate Program
svaca6@gatech.edu

Hendrik Maximilian Carius
Civil and Environmental Engineering Graduate Program
hcarius3@gatech.edu

Paloma Santiago Walker
Biomedical Engineering Undergraduate Program
pwalker41@gatech.edu

Georgia Institute of Technology
Atlanta, USA

***Abstract* — Robotics plays an increasingly vital role in advancing healthcare technologies, enabling precise surgical tools, assistive devices, and automation systems that improve patient outcomes. In this introductory course, this project aims to provide a foundation in robotics by integrating mechanical design, electrical systems, and programming principles to develop a 4-degree-of-freedom robotic manipulator capable of following pre-determined geometric trajectories. A resolved rates algorithm was implemented to compute joint velocities to enable smooth and precise trajectory execution. The robotic manipulator is designed to operate on arbitrarily angled planes with a complementing computational simulation. To ensure repeatable accuracy, the robotic manipulator includes a calibration system with limit switches at each joint. These switches allow the robot to determine its starting position reliably by moving each joint to its limits before executing any trajectory. Additionally, the robot's base is designed to house and protect its electronic components, featuring integrated functionalities such as a stop button for safety, LED indicators for communication, and accessible ports for power supply and Arduino connection. This hands-on project bridges theoretical robotics concepts with practical skills, highlighting how foundational engineering principles can be applied to impactful domains like healthcare. The outcome not only demonstrates the capability to control robotic motion in real time but also inspires future exploration of robotics in medical applications.**

I. Introduction

"Crazy Shot," our 4-degree-of-freedom robot (DoF), was developed in response to the growing demand for robotic systems in the medical field, particularly in general surgery. Robots offer numerous benefits, including enhanced precision, stability, and the ability to perform repetitive tasks with high accuracy.

To reflect the intricate motions required during surgical procedures, we designed our robot to replicate similar movements through a variety of test trajectories, such as circles, triangles, and squares. The primary goal of this project was to develop a 4-DoF robotic manipulator and control its end-effector to follow pre-defined paths, demonstrating its capability for tasks demanding precision and adaptability.

II. Materials

A. Hardware

The Hardware components of the *"Crazy Shot"* a 4-DoF robot include a combination of off-the-shelf materials and tools that complement our custom 3D-printed parts. The primary electronic components include an Arduino Mega microcontroller, a CNC shield, four stepper drivers, and four stepper motors, all powered by a 12V 10A power supply connected through a power jack and XT60 connectors. For safety and precision, we incorporated limit switches, an emergency stop button, and ball bearings. The structural framework utilizes rectangular PVC tubes for rigidity. A longer Arduino cable connects the robot to a laptop running MATLAB and Arduino software for programming and control.

To enable testing and visualization, a whiteboard and mini expo markers were used for trajectory plotting, while a camera was employed to record and analyze the robot's movements. All components were housed in a 10.4 x 7.2 x 3.7 box to ensure an organized setup.

B. Custom Parts

Our custom components were designed using SolidWorks software and fabricated with a 3D printer to form the four joints of the "Crazy Shot" robot. These parts were specifically tailored to ensure precise movement and seamless integration with the electronic and structural elements of the system. In order to save time, we decided to go with a modular system which connects using the PVC tubes. This allowed us to adapt the arm lengths without having to reprint our custom joints.

- **Base Joint:** The base joint consists of a hollow cylindrical structure that houses the shaft of the primary motor. This design enables the motor's shaft to pass through and connect to the first joint, supported by a ball bearing. This configuration facilitates smooth and stable rotation of the robot arm.

- **Second Joint:** The second joint, distinguished by its dark blue color, securely holds the second motor. The motor shaft connects to the main arm structure, enabling controlled rotational movements.
- **Elbow Joint:** The third custom component forms the elbow joint, integrated with the third motor. It connects the arm to the second joint (dark blue) and allows precise up-and-down movements.
- **End-Effector Joint:** The final custom part supports the smallest motor. A custom half-hollow cylindrical component is used to hold and manipulate the marker. This joint ensures accurate positioning of the marker tip for drawing pre-defined trajectories.

III. Method

We implemented two distinct approaches for calculating the forward kinematics and planning the trajectory of our robot: Task Space Interpolation with Inverse Kinematics (IK) and the Resolved Rate Algorithm (RRA). Both methods relied on mathematical formulations to compute joint angles and ensure smooth end-effector motion along predefined trajectories.

A. Forward Kinematics

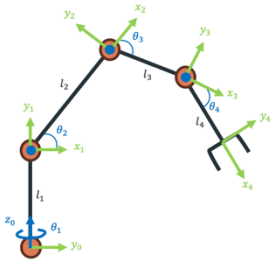

Figure 1. 4-DoF robot diagram with frames based on DH convention

The 4 DoF robot manipulator is composed of all revolute joints with the base joint having the only rotation axis that is not parallel to the other joint rotation angles, as shown in *Figure 1*. Its corresponding DH table is displayed in *Table 1*. It's the basis for computing the robot's forward kinematics, which were also used in the simulation done in MATLAB. It can be used to verify the generating of the path planning trajectories.

Table 1. DH Table based on Figure 1 configuration

Link	a_i [mm]	α_i [rad]	d_i [mm]	θ_i [rad]
1	0	$\frac{\pi}{4}$	193.3	θ_1
2	193.2	0	0	θ_2
3	150.5	0	0	θ_3
4	59.7	0	0	θ_4

To describe the position of the end-effector, we use the forward kinematics equation:

$$p = f(q)$$

Where:

- $p \in R^3$ is the position of the end-effector in Cartesian coordinates.
- $q = [q_1, q_2, q_3, q_4]^T$ are the joint angles of the robotic arm.
- $f(q)$ is a nonlinear function describing the kinematics of the robotic arm, derived using Denavit-Hartenberg parameters.

B. Task Space Interpolation with Inverse Kinematics

To generate smooth motion, we interpolated between points on the trajectory using task space interpolation. For example, to generate a square trajectory, the vertices of the square are defined, and linear task space interpolation is used to compute intermediate points. In addition to the position of the points we also set the velocity $\dot{p}(t)$ as well as the acceleration $\ddot{p}(t)$ to zero to ensure a smooth stopping motion. The resulting trajectory of the task space interpolation specifies the end-effector position at discretized timesteps. To compute the joint angles for a desired end-effector position p_d, we solve the inverse of the forward kinematics equation:

$$q = f^{-1}(p_d)$$

Our solution of the forward kinematics considers constraints like joint limits and ensures the desired orientation of the end-effector. Finally, the computed joint angles $q(t)$ are then sent to the robotic arm for execution or simulated using forward kinematics.

C. Resolved Rate Algorithm

The Resolved Rate Algorithm has an iterative approach to calculate joint velocities. It focused on dynamically driving the robot toward its goal by continuously reducing the position error.

Our algorithm is based on the Jacobian matrix $J(q)$, calculated using this toolbox [1], which relates to the end-effector velocity $\dot{p}$ to the joint velocities $\dot{q}$:

$$\dot{p} = J(q)\dot{q}$$

To find the joint velocities needed to reduce the error between the current and desired positions, we used the pseudoinverse of the Jacobian matrix:

$$\dot{q} = \mathrm{J}^{\dagger}(\mathrm{p_d} - \mathrm{p})$$

Where:

- $(p_d - p)$ is the position error vector.

This equation allows the robot to iteratively adjust its joint angles and move toward the target.

At each timestep, the joint angles were updated using:

$$q_{k+1} = q_k + \dot{q}\Delta t$$

Where:

- q_{k+1} is the updated joint angle vector.
- Δt is the timestep.

The process continues until the error $|p_d - p|$ fell below a specified tolerance, ensuring the desired accuracy.

This method was particularly effective for paths with complex shapes or when dynamic adjustments were needed. It eliminated the need for a closed-form solution to inverse kinematics, making it especially useful for redundant systems.

D. Mechatronics

Figure 2 provides a visual representation of the mechatronics system of Crazy Shot. The mechanical components, such as the stepper motors, linkages, and limit switches work seamlessly with the electronic systems, including motor drivers, LEDs, and emergency stop button. The computational subsystem, controlled by the Arduino and MATLAB interface, manages transitions between states.

In the Idle state, the robot remains stationary while awaiting commands. Once data is received, the Parsing and Processing states interpret and validate instructions. The Movement state ensures coordinated motor operation to execute planned trajectories.

Finally, Safety mechanisms such as Abort and Error states allow the system to stop immediately in response to faults or user interventions.

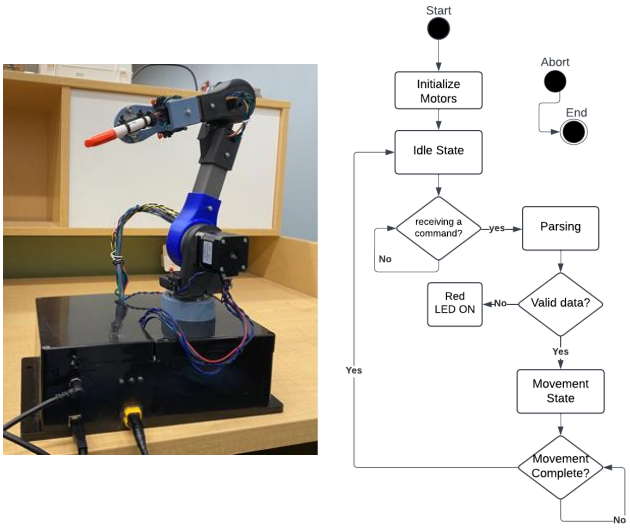

Figure 2. Crazy Shot State Machine Diagram

E. Some Common Mistakes

- Overstrain on 3D-Printed Components: The plastic 3D-printed parts broke due to excessive strain and overloading during testing, emphasizing the need to account for material limitations and stress distribution during the design process.
- Coding Errors in Path Flow: Misdirection in the robotics' movement path was caused by incorrect control logic. Using negative values (e.g., -300) instead of positive values, resulted in significant deviations in the robot's motion.
- Simulation vs. Real-World Mismatch: Assumptions that code working in MATLAB simulations would function identically on the physical robot led to delays. Real-world dynamics, hardware constraints, and system integration required additional adjustments and testing to align with the simulation results
- Motor Overuse Leading to Ghost Commands: Running motors for extended periods caused unintended actions or "ghost commands." These issues arose from timing mismatches in motor control and required refined algorithms to prevent errors.

IV. Testing & Results

After the robot assembly was complete including electrical connections and ensuring that the code translated to independent motor movement, the next phase was to test different trajectories. Before each test of the different geometric paths, the robot arm underwent a calibration process to ensure that the starting position remained consistent in each trial.

A. Independent Motor movement

To ensure that each motor in *Crazy Shot* could move independently to a discrete angle, we first established proper connections between the stepper motors (Nema 17), the Arduino Mega, and the A4988 motor drivers. This setup followed a minimal wiring diagram, like the one used in Lab 1-Motor Control. This testing only used Arduino, not relying on MATLAB input. By adhering to these procedures and thoroughly testing discrete angular movements for each motor, we verified the system's capability to achieve accurate and reliable motor control. It was also important to establish that the motor could move to the angled position in both directions. This first came with some issues as the direction pin in the stepper driver didn't have a direct connection due to a faulty connection in the CNC shield. After bridging the direction pin of the stepper drivers directly to an Arduino pin, the motors were able to move properly.

B. Simultaneous Motor Movement / Calibration

After the motors were able to move independently, they were tested to move simultaneously to confirm that the stepper drivers could handle the strain and were set at the corresponding current. In the simultaneous movement, a calibration code was made. It is designed to calibrate each motor by moving it to the first limit switch, then reversing the motion and counting the number of steps it takes to reach the other switch. The process is simultaneously performed for the motors 2 to 4. Since motor 1 only had one limit switch, we had to manually acquire the number of steps from that limit switch to its zero position during testing. Upon completion of the calibration for all motors, they moved to their zero position simultaneously in which the robot arm is aligned vertically.

C. Testing Figure Drawing

For all trajectories and both control algorithms, the end effector was moved to a point close to the trajectory before initializing the path movement.

a) Circle Trajectory: Tested on an inclined plane of 20°, the expected trajectory is a circle of 10cm in diameter. As seen in the computational model in *Figure 3*, this testing was done using the Resolved Rates Algorithm, which performed not as smoothly as a motion using Task Space Interpolation which also considers acceleration and deacceleration.

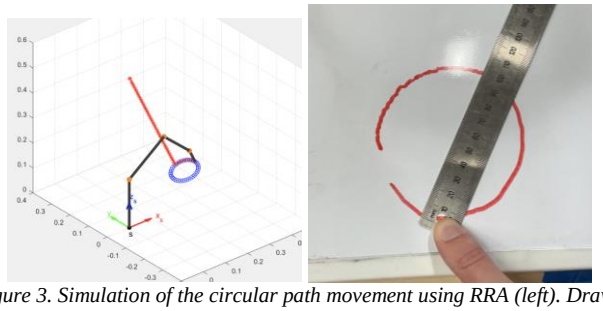

Figure 3. Simulation of the circular path movement using RRA (left). Drawn path with evidence 10cm measurement (right)

b) Triangle Trajectory: Tested on the XY Plane (0°), the expected trajectory was a triangle with a 10cm width base and 10cm height. For this trajectory, Task Space Interpolation was used showing a smooth motion along the plane.

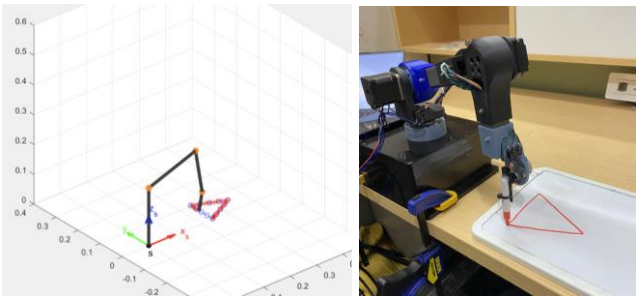

Figure 4. Simulation of the triangle geometric path movement using Task Space Interpolation (left). Crazy Shot following command and drawing along the 0° plane.

V. Conclusion

The team was able to construct a 4 DoF robotic manipulator with successful communication between MATLAB and Arduino to follow varying geometric paths along angled planes. The arm's movement was precise using both the Resolved Rates algorithm and Task Space Interpolation methods. We were able to have a computational model providing a reliable way to calculate the arm's end effector position. While both RRA and Task Space Interpolation with Inverse Kinematics were tested, the latter proved to be more effective in achieving a fluid motion compared to RRA. Further improvements would be necessary to refine the resolved rate algorithm to enhance its motion transitions.

Looking ahead, there are several potential areas for improvement and expansion. First, the robotic arm's functionality could include supporting a wider range of shapes and surfaces, allowing for more versatile applications. Additionally, developing a user-friendly interface for customizing shapes and movements would significantly enhance the usability and flexibility of the system, making it more accessible to users with varying technical backgrounds and increasing its applicability in the medical field. These improvements would enable the robotic arm to become more adaptive, efficient, and user-centric, opening doors for a variety of real-world applications.

Acknowledgment

We extend our gratitude to all team members for their unique contributions to this project. Special thanks go to Professor Chen for providing the funding through the Georgia Institute of Technology and granting us access to his robotics research lab, where we finalized our robot.

We would also like to thank GaTech's HIVE for providing the foundation of our robot by offering access to their soldering equipment, tools, and materials, which were instrumental in the early stages of development.

Lastly, we wish to acknowledge Jia Shen, a Ph.D. student and teaching assistant in Professor Chen's lab, for his unwavering support, always being available—even late at night—to answer questions and offer invaluable advice.

References

[1] Corke, P. (n.d.). Robotics Toolbox for MATLAB. Peter Corke. Retrieved December 4, 2024, from https://petercorke.com/toolboxes/robotics-toolbox/