Designing a Robot Arm for Fused Deposition Modelling with Stepper Motors
Challenges and Solutions

Sushant Singh
Department of Mechanical Engineering
SRM University
Chennai, Tamil Nadu
sushant_singh23@yahoo.com

Vasanth Kumar C.H
Department of Mechanical Engineering
SRM University
Chennai, Tamil Nadu
vasanthkumar.s@ktr.srmuniv.ac.in

Abstract—This journal outlines the approach we took for the design of the Robot Arm and the challenges that one might experience while building a robot arm (SCARA based) for fused deposition modeling using stepper motors. The robot has two degrees of freedom in the X-Y plane and another in Z axis to move the arm vertically. This design brings the advantages of quick positioning as only two stepper motors are required for the positioning in the X-Y plane as compared to an articulated robot. We experienced some challenges with respect to positioning the arm. A deeper dive into this revealed that the problem was a result of the limitations of the inverse kinematics algorithm we were using which resulted in points of no solution (invalid solution). Since we were using stepper motors in an open loop configuration the system was unable to detect these points of no solution. This paper delineates the approach taken by us and the possible solutions to these problems.

Keywords—SCARA, 3D printing, Fused Deposition Modelling (FDM), Micro-Stepping, Robotic Arm.

I. INTRODUCTION

It is said that Additive Manufacturing is the third most disruptive technology from a chronological perspective and one that is likely to revolutionize the manufacturing industry as we know it today. Multiple techniques are available to implement Additive Manufacturing; examples are CNC 3D printing [8], Delta Printers, and Robotic Arms. One area where Robotic Arms have an edge over the other techniques is their versatility, ability to be mounted on a linear guide and the possibility of using them in arrays. Collectively these advantages are driving the interest and adoption of the Robotic Arm in the manufacturing industry; especially for rapid prototyping.

II. OBJECTIVE

The objective of our project and this document that describes it is to construct a Robotic Arm using Stepper Motors and document the learning arising from it. The Robotic Arm we constructed has three Degrees of Freedom (DoF), the first in the X-plane, the second in Y-plane and the third in Z-plane. While most of the work is performed in the XY plane, the Z-plane comes into play only after the successful deposition / addition of a layer. This is done by raising arm in the vertical or Z-plane by a defined linear distance.

III. DESIGN REQUIREMENTS

The design requirements for construction are explained as per different sections of the robotic arm:

A. Fore-Arm (link 2)

The forearm has to have the ability to sustain the bending moment in the plane perpendicular to that of the forearm which is the vertical plane. The bending moment is generated by the combined weight of the links that make up the arm and the load of the end effector. Using the Bending Equation, we have calculated the minimum Moment of Inertia (MI) required about the bending axis as well as the deflection induced.

\[
\frac{M}{I} = \frac{\sigma}{y}
\]  

(1)

Where: \( M \) is the bending moment generated due to the culmination of all loads on the forearm, \( I \) cross section beam is used for the best weight to bending ratio, \( \sigma \) is Bending Stress (N/m²) and \( y \) is distance from neutral axis.

To calculate deflection we use the standard formula:
Where: \( \delta \) is deflection induced at the end of the forearm, \( w \) is self-weight of forearm to length of the forearm (N/m) and \( W \) is load at acting at the end of the forearm. The deflection calculated by the above formula is less than 0.01 mm; which is less than the Control Resolution of our stepper motors. This can be further arrested by attaching a second link connected to the primary via threaded shafts pivoted along the same axis of rotation of the stepper motor spindle.

B. Elbow-Arm (link 1)

The design of the Elbow is similar to that of the Forearm. The only variation is the change in self-weight (usually higher) of the Elbow and the change in length (usually longer). Therefore, equation is commensurate to the Forearm and is modified as:

\[
\delta = \frac{w_1 L_1^2}{6EI} + \frac{w_2 L_2^2}{6EI} + \frac{w_2 L_2 \left(L_1 + \frac{L_2}{2}\right)}{3EI}
\]

Where: \( w_1 \) is the ratio of the self-weight of the Forearm to length of the Elbow (N/m), \( w_2 \) is self-weight of the Forearm to length of the Forearm (N/m), \( L_1 \) is Length of the Elbow and \( L_2 \) is Length of the Forearm. Similarly, the deflection calculated by the above formula is less than 0.01 mm; which is less than the Control Resolution of our stepper motors. We used two parallel placed aluminum sheets attached to each other with stainless steel shafts to arrest their relative movement. This helps in making the links more rigid and less prone to vibration. The frame on which the Arm is mounted is made out of mild steel to make the frame resilient to vibration.

IV. CONTROL SYSTEM

In the final setup we used electronic stepper motor driver controllers driven by the Arduino Galileo Gen 2 microcontroller board.

*Initial considerations for actuation:*

- Open Loop vs Closed Loop: Stepper vs Servo respectively.
- The driving method: Direct Pulse Width Modulation (PWM) trough Arduino or through Drivers
- The mode of operation for Stepper Motors: Full Step or Micro-Stepping
- Torque: Calculation is done for a constant load.

A. Motors

We chose to go with the stepper motor for actuation. They are very accurate and are able to maintain their position by generating a holding torque, once powered. Selection of motors was done on the basis of torque requirements. The selection of motors was done based on the torque required. The formula used to calculate necessary torque \( T_1 \) and \( T_2 \) is as follow.

\[
T_1 = \left( W_1 L_1^2 / 2 \right) + (M_2 L_1) + \left( W_2 L_2^2 / 2 \right) + (M_2 L_2)
\]

\[
T_2 = \left( W_2 L_2^2 / 2 \right) + (M_2 L_2)
\]

Where: \( W_1 \) is Weight of link 1, \( W_2 \) is Weight of link 2, \( M_2 \) is weight of Motor 2, \( M_3 \) is Weight of Motor 3, \( L_1 \) is Link 1 or Elbow’s length and \( L_2 \) is Link 2 or Forearm’s length. Fig.1 is provided as a reference and illustration.

B. Motor Drivers

There are several electronic stepper motor drivers commercially available for driving stepper motors. We initially used TB6550 four-axis CNC Stepper Driver based on its technical specification. Half way through the project the TB6550 electronics failed; possibly because it was a little underrated for our application. We then looked for appropriate driver boards by raising the headroom of specifications (safety factor to two). For example for a motor that consumes 1 ampere consider drive electronics that can source at least a 2 amperes to protect the driver from being over-driven and getting damaged. We upgraded our motor drivers from TB6550 to Bhola Bath Micro Stepping (MS) drivers. The drivers normally have two connectors named as P1 & P2 (It may differ slightly in nomenclature from driver to driver). The
function of the P₁ connector is control the signal connection, whereas P₂ connector is for providing power and connecting motor wires. The tables mentioned below further delineate the different pins under both connectors P₁ and P₂.

### TABLE I. PIN CONFIGURATION OF CONNECTOR P₁

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Details of Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUL</td>
<td>PULSE NEGATIVE</td>
<td></td>
</tr>
<tr>
<td>DIR</td>
<td>DIRECTION NEGATIVE</td>
<td></td>
</tr>
<tr>
<td>EN</td>
<td>ENABLE NEGATIVE</td>
<td></td>
</tr>
<tr>
<td>VCC</td>
<td>SIGNALING VOLTAGE</td>
<td></td>
</tr>
</tbody>
</table>

### TABLE II. PIN CONFIGURATION OF CONNECTOR P₂

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Details of Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDC+</td>
<td>A DC POWER SUPPLY</td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>GROUND</td>
<td></td>
</tr>
<tr>
<td>A+ and A-</td>
<td>MOTOR PHASE OF A</td>
<td></td>
</tr>
<tr>
<td>B+ and B-</td>
<td>MOTOR PHASE OF B</td>
<td></td>
</tr>
</tbody>
</table>

Where: PULSE is to set motor frequency (Hz), ENABLE is ON/OFF, VCC is 5V, Direction is Clockwise and Counter-Clockwise and VDC is 24V.

### C. Microcontroller

Controlling stepper motors requires providing precisely timed electronic pulses spaced at microseconds and following a temporal order relative to other control pulses. Initially we used Raspberry Pi 3 because it runs a light version of Linux operating system that supports Python IDE that can be used for writing applications in Python. But we noticed that due to processing overheads of the Linux OS, performing I/O (input / output) through its GPIO interface did not give us deterministic timing pulses we needed, this resulted in slipping and missing of steps at times by the motors. To address this we used the Intel Galileo Gen 2 microcontroller board because it is one of the most powerful Arduino boards. Galileo can send PWM waves within a tolerance of plus minus five microseconds and we found it adequate for achieving the timing of the control pulses to the stepper motor controller and no steps were missed.

### D. Control Schematic

M represents Stepper Motor (1, 2, 3 and 4)

### V. CALCULATIONS

#### A. Inverse Kinematics

We used Inverse Kinematics to figure out the distance to be moved by the stepper motor to achieve desired target X and Y position; point P in the figure below. This is in turn used to calculate the rotation of the motors. That point \( P(p_x, p_y) \) are attained as follows. The equations are an exception to the prescribed specifications of this template. You will need to determine whether or not your equation should be typed using either the Times New Roman or the Symbol font (please no other font). To create multileveled equations, it may be necessary to treat the equation as a graphic and insert it into the text after your paper is styled.

\[
\begin{align*}
    p_x &= l_1c_1 + l_2c_{12} \\
    p_y &= l_1s_1 + l_2s_{12} \\
    q_2 &= \text{ATAN2} (s_2, c_2) \\
    q_4 &= \text{ATAN2} [p_y, p_x] - \text{ATAN2} [l_4 s_2, l_4 + l_2 c_2]
\end{align*}
\]
Where: $s_1$ is sine of $q_1$, $s_2$ is sine of $q_2$, $c_1$ is cosine of $q_1$, $c_2$ is cosine of $q_2$, $s_{12}$ is sine of $q_1 + q_2$ and $c_{12}$ is cosine of $q_1 + q_2$.

**Fig. 2. Inverse Kinematics**

**B. Motor Speed Control**

Using the combination of Galileo Gen 2 feeding the PULSE pin on the stepper motor controllers we can control the frequency of the pulses used to rotate the motors which translates to motor speed in R.P.M. The PULSE signal is generated by Arduino Galileo Gen 2 under program control. To calculate the required frequency for given R.P.M we use the following formula.

$$\text{Frequency (Hz)} = \frac{200 \times n \times \text{R.P.M}}{60} \quad (10)$$

Where $n$ is the micro stepping factor.

**C. Jacobian Matrix**

The angular velocity of the motors varies based on the position of the Arm or the angular rotation of the motors. We used the Jacobian matrix to calculate the velocity required for traversing each point. Achieving the said angular velocity is critical to achieving an accurate Feed Rate of the end effector. The g-codes provided the coordinates of the points to be visited and the required feed rate. The Jacobian matrix for two arm manipulator is $J(q)$ and is in the form of the matrix below.

$$J(q) = \begin{bmatrix}
-L_1 \sin(q_1) + L_2 \sin(q_1 + q_2) & -L_2 \sin(q_1 + q_2) \\
L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) & L_2 \cos(q_1 + q_2)
\end{bmatrix}$$

The velocity for the end effector can be found by taking the differential of the matrix given above i.e. denoted by $\dot{x}$ which is the velocity matrix for the end effector, then

$$\dot{x} = J(q)\dot{q} \quad (11)$$

The most important factor for maintaining an accurate trajectory is for the motors to complete their respective angles in equal time. Using the angles calculated from inverse kinematics we can get the ratio of angular velocities of the Elbow and the Shoulder motor. When substituted into the Jacobian matrix it yields the angular velocity achieved for each point. This means the motor velocity will change from point to point.

$$\begin{bmatrix}
\frac{\theta_S}{\theta_E}
\frac{\omega_S}{\omega_E}
\end{bmatrix} = t \quad (12)$$

Where: $\theta_S$ is Angle moved by the shoulder motor, $\theta_E$ is Angle moved by elbow motor, $\omega_S$ is Angular velocity of shoulder motor, $\omega_E$ is Angular velocity of elbow motor, $t$ is Time taken for translation from one point to next. This relation can be used for setting the extrudate feed rate. Cura software can also be used to determine the amount of filament to be extruded.

**D. Control Resolution**

Control resolution can be understood as the circumference of the maximum attainable or subtended circle divided by the required number of steps to complete one revolution. Since we are driving our motors at 1/16 micro-stepping the required number of steps to complete one revolution would be equal the 3200 steps; 1/16 of step angle of 1.8°.

$$\text{Control Resolution} = \frac{2\pi}{3200} \quad (13)$$

**VI. G-CODES FOR 3D PRINTER**

G-Codes provide the control information for 3D printing. We used the RepRap G-Code system and not the Linux-based CNC G-Codes. The part once designed on SolidWorks or Catia has to be transferred into a STL format. The file
containing STL format is then processed by a program's such as Cura or Slic3r. Cura divides the designed part into individual layers of a certain thickness also known as slicing. Then it produces a single G-Code that contains the data corresponding to each layer. The software also provides us the amount of material consumed along with other configuration data such as when to extrude, how much to extrude and the overall feed rates. The algorithm developed by us processes the G-Codes and does the following:-

- Positioning of the four stepper motors; three for X, Y and Z plane and the forth the extruder motor.
- Maintaining a constant feed rate.
- The mode of operation for Stepper Motors: Full Step or Micro-Stepping
- Setting nozzle temperature and feed of filament.

VII. PROCESS

The program developed on Arduino Galileo Gen 2 platform follows the following steps to operate the machine:
1. Cura is used to create a g-code file, this step is performed on a PC.
2. The g-code is copied to an SD card which is then inserted into the Galileo Gen 2 board.
3. The program parses the g-code file for coordinate values and federates for each layer.
4. Using Inverse Kinematics the program calculates the rotation of the motors in degrees to be moved for every point to be traversed.
5. Jacobian gives us the velocity vector for each point.
6. The number of steps are calculated and then executed.

VIII. RESULTS

The images below illustrate the results of software program and the corresponding movement of the Robot Arm. The Arm drew these figures by running g-code files from Cura. The lighter strokes are retraction strokes whereas the darker markings depict the printing trajectory. These images show that the algorithm developed works fine excluding the points where the algorithm does not produce any solution. At these points there is no motor movement and the entire trajectory shifts or slips.

The images were drawn by disabling the Z-axis so that the retraction strokes can be seen. Note that for images with sharp corners the trajectory undergoes a shift, see Figure 4. However when there are no sharp corners the trajectory is maintained, see Figure 5.

IX. CHALLENGES

There were a few challenges which we noticed as we started testing the machine; although the machine was accurate and precise in positioning to home coordinates it seemed to drift from its main trajectory from time to time. After careful analysis we noticed that all the points in the g-codes are not generating a valid solution using inverse kinematics. Also the R.P.M of the motor calculated using the Jacobian matrix could not be achieved by the motor at higher feed rates. This is especially during micro-stepping as overall speed characteristics of the motor falls i.e. there is a trade-off between speed and accuracy although that can mitigated by running the machines at lower feed rates. Also there are only two pins on Intel Galileo Gen 2 that support hardware interrupt, this limits the number of positional sensors; like end-stops sensors we can use.
X. SOLUTIONS

A. Closed Loop System

Instead of an open loop we can use closed loop control by using stepper motors in conjunction with rotary encoders. This will allow the processor to sense the lack of any movement when one is expected. Whenever the system executes the point of no solution there will not be any motor movement this can be used as a trigger using the encoder to identify such points. When these no solutions points are identified the system can extrapolate; i.e. increment the points by a fraction of distance in both X and Y coordinate until a solution is achieved. This will prevent the arm from losing its trajectory as it prints.

B. Pre-Processing of Work Space

Alternately, once the work space is defined and minimum control resolution of the machine is calculated the system can identify each unique point in the work space. In pre-processing all the points of no solution have to be excluded and replaced with near accurate points for which solution is possible. This data can be fed into the system, whenever parsing through a g-code file the system could identify the points for which it won’t get a solution and use a suitable algorithm to replace the coordinates of those points with the newly calculated solution points. A combination of these techniques can be used to create a more stable operation.

Acknowledgment

We would like to thank Dr. S. Prabhu, Head of Department, Mechanical Engineering, SRM University, Chennai for constantly encouraging and guiding us and helping us get the funding for the project from the University. We would also like to thank Pragalbha Sudame for the assistance in designing the frame of the robot arm and Anupam Jha for developing the software application.

References

[1] The Research and Application of Descriptive Programming for Robot Control System Jinhua Song*1, 2, a, Zhengdong Wang1, 3, b, Haitao Fang1, 3, c, Kai He1, 3, d, Ruxu Du4, e.


[3] Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods Samuel R. Buss
Department of Mathematics University of California, San Diego La Jolla, CA 92093-0112 sbuss@math.ucsd.edu October 7, 2009.


[6] Innovation futures:). Wesam Mohammed Jasim(Solution of Inverse Kinematics for SCARA Manipulator Using Adaptive Neuro-Fuzzy Network)