Research on Prototype and Simulation of 5-DOF Robot Arm based on MATLAB and ADAMS

Cong Fu, *congfu@umich.edu* Boyuan Li, *liboy@umich.edu*
Ting Wang, *tingwa@umich.edu* Shiyi Qian, *qsy@umich.edu*

***Abstract*—This project is concerned with prototype, control and simulation of 5-DOF robot based on MATLAB and ADAMS. First setting Denavit-Hartenberg table and using Geometric Approach to calculate forward and inverse kinematics, respectively. The physical model of robot arm is built by Solidworks and ADAMS and then prototyped by 3D printing. Furthermore, a control model is created in MATLAB/Simulink and the co-simulation model was established based on ADAM/Control and MATLAB/Simulink. The simulation results indicate that robot arm system has relatively preferable response characteristics and nice trajectory-tracking ability.**

***Index Terms*—Forward kinematic, inverse kinematics, trajectory planning, MATLAB, ADAMS.**

I. Introduction

Robot arm is a reprogrammable, multifunctional manipulator design to move material, parts, tools, or specialized devices through various programmed motions for the performance of variety of task. Nowadays, robots in industries have a wide spectrum of applications. The most popular applications are painting,welding and assembling manipulators in vehicle manufacturer. Moreover, many academic and research laboratories have been trying and developing new motion plan and position and force control algorithm in order to extend its usage to diverse applications.

In order to get a deep understanding of what we have learned in the class, a three-link robot manipulator with five degree of freedom has been designed. The objective of our designed robot is to be implemented with hardware (Arduino) and simulated with MATLAB/SIMULINK and ADAMS. A interface built in MATLAB would allow users to achieve accurate control of robot control by specifying trajectory planning. Unfortunately, feedback control can not be achieved due to the lack of sensor in motor.

To accomplish desired tasks above, our project is divided into four subsections. Section 1 focus on the geometry analysis of robot arm. We first use Danevit-Hartenberg table to calculate the forward kinematics and determine the end effector's position and orientation in terms of the joint variables. Then we use the inverse kinematics to find the joint variables in terms of the end effector's position and orientation. Section 2 highlight the trajectory planning algorithm. In this section we set the desired trajectory as a circle in XY plane with the center at (0,150,200) and radius 30mm. In section 3, a MATLAB model is built to connect our math model and hardware through Arduino. A MATLAB GUI is also built in this section to allow users to control the robotic motion. The final section, which is also the highlighted section in this project, concentrates on the simulation of our designed robot manipulator. A three-dimensional model of our robot arm is first constructed in Solidwork and printed out through 3D printing process, the model is then exported to ADAMS software for simulating kinematic behavior of robot arm with an acceptable error.

II. Kinematics Design

A. Forward Kinematics

Forward kinematics refers to the use of the kinematic equations of a robot to determine the position and orientation of the end effector given the values for the joint angle variables of the arm robot. We establish the relationship between the base fixed frame and the end effector fixed frame. A commonly used convention for selecting frames of reference for robots is the Denavit-Hartenberg (DH) convention, a convention for the definition of the joint matrices and link matrices to standardize the coordinate frame for spatial linkages[1]. When applying this convention to the arm robot, we can easily get the homogeneous transformation matrix connecting two adjacent links, which consist of four basic transformation matrices.

DH convention defines four parameters associated with link i and joint i, which are θ_i(joint angle), d_i(link offset), a_i(link length), and α_i(link twist). The schematic of armbot(initial configuration) and DH table is showed in Fig. 1 and Table I, respectively.

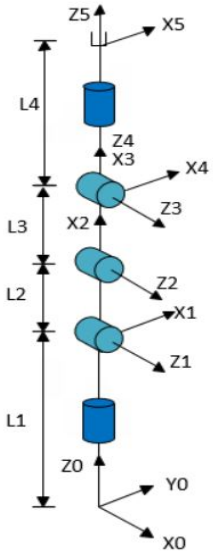

Fig. 1: Schematic for forward kinematics

TABLE I: DH Table

Link	θ_i	d_i	a_i	α_i
1	$\theta_1 + \pi/2$	95	0	$\pi/2$
2	$\theta_2 + \pi/2$	0	115	0
3	θ_3	0	74	0
4	$\theta_4 - \pi/2$	0	0	$-\pi/2$
5	θ_5	60	0	0

Then we can express the origin of the end effector fixed frame in the base frame by multiplying $A1A2A3A4A5$ and homogeneous coordinate of the origin of end effector fixed frame. In addition, the arm robot we use is a planar robot, so the end effector position can be defined by x, y, z and ϕ, where ϕ is the angle between end effector and horizontal plane and defined as $(\theta_2 + \theta_3 + \theta_4 - 90°)$.

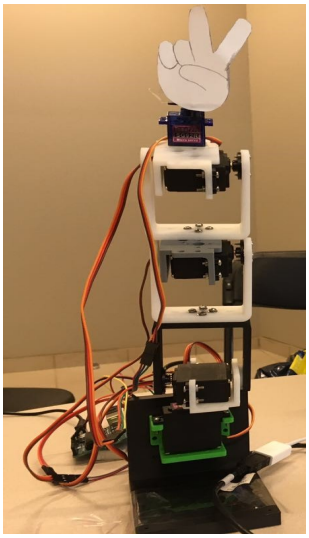

Fig. 2: 5-DOF Robot Arm

B. Inverse Kinematics

We tried to implement Jacobian and cyclic coordinate descent algorithm to do the inverse kinematics, but the results are not as good as we desired. The robot arm always moves to some crazy waypoints before reaching the command position./And another simple and common way to calculate the inverse kinematics is to use geometry method. For this 5-DOF Robot arm, we can simplify its configuration to three joints RRR arm to do the geometry inverse kinematics. We assume each joint angle is $\theta 1$, $\theta 2$, $\theta 3$, $\theta 4$,respectively. ϕ is the angle between the last rigid link and horizontal axis. We define ϕ is negative above the horizontal axis and positive below.

There are two different configurations for Robot arm, which are elbow up and elbow down. The only difference of these two configurations is the second and third links. So we should should calculate joint angle $\theta 2$, $\theta 3$ separately according different cases.

We assume (x,y,z,ϕ) is the destination position. We can calculate the base joint angle $\theta 1$ by using x and y coordinates. And this base joint angle is the same for both configuration.

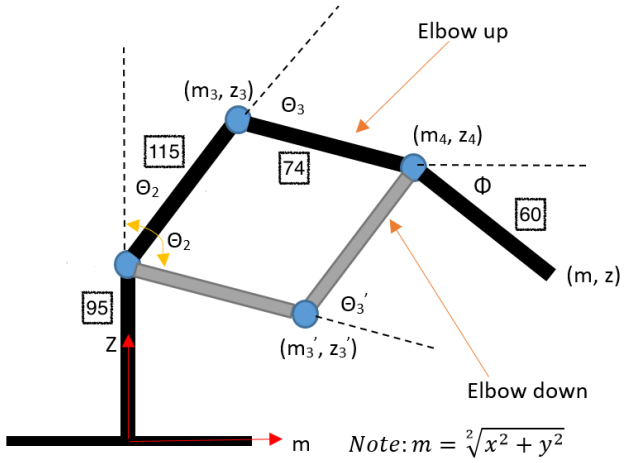

Fig. 3: Definitions for inverse kinematics calculations(side view)

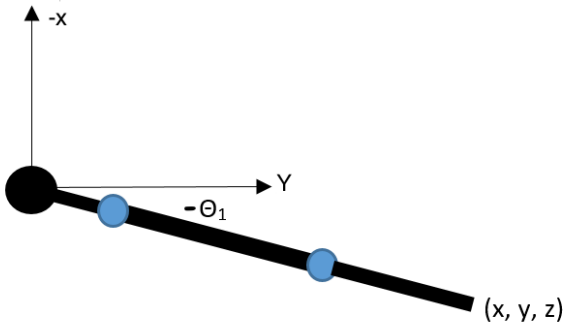

Fig. 4: Definitions for inverse kinematics calculations(top view)

$$\theta_1 = \arctan(\frac{x}{y}) \tag{1}$$

$$m = \sqrt{x^2 + y^2} \tag{2}$$

$$m_4 = m - d_5 \times \cos\varphi \tag{3}$$

$$z_4 = z - d_5 \times \sin\varphi \tag{4}$$

Here d_5 is 60 mm. And we define a intermediate variable λ,which is :

$$\lambda = \arctan(\frac{z_4 - z_2}{m_4}) \tag{5}$$

where z_2 is 95 mm in the picture. And here we introduce another three intermediate variables, a, which is the distance between joint2 and joint4, $\beta 1$ and $\beta 2$, which are the internal angle of the triangle composed by link2, link3 and distance a. In the following calculation, l2 is 115mm and l3 is 74 mm.

$$a = \sqrt{m_4^2 + (z_4 - z_2)^2} \tag{6}$$

According to the Law of sines, $\beta 1$ and $\beta 2$ can be calculated below:

$$\beta_1 = \arcsin(\frac{l_3 \times \sin\theta_3}{a}) \tag{7}$$

$$\beta_2 = \arcsin(\frac{l_2 \times \sin\theta_3}{a}) \tag{8}$$

For the joint angle $\theta 3$, we can use Law of Cosines to calculate it. For elbow down configuration:

$$\theta_3' = -\arccos(\frac{a^2 - {l_2}^2 - {l_3}^2}{2 \times l_2 \times l_3}) \tag{9}$$

$$\theta_2' = \frac{\pi}{2} - \lambda + \beta_1 \tag{10}$$

$$\theta_4' = \phi + \lambda + \beta_2 \tag{11}$$

For elbow up configuration:

$$\theta_3 = -\theta_3' = \arccos(\frac{a^2 - {l_2}^2 - {l_3}^2}{2 \times l_2 \times l_3}) \tag{12}$$

$$\theta_2 = \frac{\pi}{2} - \lambda - \beta_1 \tag{13}$$

$$\theta_4 = \phi + \lambda - \beta_2 \tag{14}$$

For geometric inverse kinematics,it is very important to determinate a proper ϕ for Robot Arm. With a constant ϕ input,the robot might not reach some points that should be reached with a different ϕ. We implemented two method to fix this problem. One method is to set a proper ϕ for each waypoints in the later path planner. Another method is to create a function which is used to calculate the ϕ automatically, which means if the geometry inverse kinematics can not calculate a solution for input ϕ, it will automatically decrease the ϕ by 2 degree and do the calculation again until it finds the solution and outputs the joint angle.

III. Trajectory Planning

If we give a command to make the arm move to a specific point, joints will move arbitrarily to the desired angle. But sometimes we need to find a trajectory connect initial and final configuration to satisfy the velocity and acceleration constraint, and make the move smooth. In the we use cubic spline method. There are basically two method to generate cubic spline, joint angle and end point of arm. We adopt method of generating cubic spline of joint angles, and use forward kinematics to get the end point coordinates. This could sample enough waypoints to make the motor move in a smooth trajectory and avoid angle spike.

If we want to generate a polynomial joint trajectory between two configuration and specify the start and end velocity, we require a polynomial with four independent coefficients. Plus, we also need to define the initial and final time t_0 and t_f, and the time step t_{delta}.

The cubic spline inputs are the initial time t_0, joint angle q_0, joint angle velocity v_0, and the final timet_f, joint angle q_f, joint angle velocity v_f. Then the cubic spline function will compute the cubic spline coefficients for each joint. For the cubic spline coefficients computation, we get four equations for each motor (the i-the step matrix equation is shown as below).

$$\begin{bmatrix} 1 & t_{i0} & t_{i0}^2 & t_{i0}^3 \\ 0 & 1 & 2t_{i0} & 3t_{i0}^2 \\ 1 & t_{if} & t_{if}^2 & t_{if}^3 \\ 0 & 1 & 2t_{if} & 3t_{if}^2 \end{bmatrix} \begin{bmatrix} a_{i0} \\ a_{i1} \\ a_{i2} \\ a_{i3} \end{bmatrix} = \begin{bmatrix} q_{i0} \\ v_{i0} \\ q_{if} \\ v_{if} \end{bmatrix}$$

And then we generate a series of waypoints between q_0 and q_f at a defined time interval t_{delta} (functions are shown as below).

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$
$$v(t) = \dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2$$
$$t = t_0, t_0 + t_{delta}, t_0 + 2t_{delta}, ..., t_f$$

Algorithm 1 Cubic spline

function $cubic\_spline(P_0, P_f, V_0, V_f, t_0, t_f)$
  $t \leftarrow linespace(t_0, t_f, 100 * (t_f - t_0))$
  $M \leftarrow CoefficientMatrix$
  $b \leftarrow [q_0; v_0; q_1; v_1]$
  $a \leftarrow inv(M) * b$
  $q_traj \leftarrow polynomialwithCoefficientMatrix$
  $v_traj \leftarrow diff(q_traj)$
  $a_traj \leftarrow diff(v_traj)$

IV. Software Design

A. Arduino

With MATLAB Support Package for Arduino Hardware, we use MATLAB to interactively communicate with an Arduino board. Five servo motors are controlled by the Arduino board and working as five joints. The servo motors can only rotate 180 degree, so θ_1 to θ_5 are limited to -90 to 90 degree. Figure 5 shows the circuit of the 5-DOF robot arm. The Arduino UNO is the microprocessor of the robot, which is driven by 5 servos with 9V external DC power. The servo signal wires are connected to the digital/PWM ports 5,6,7,8,9 of the Arduino UNO.

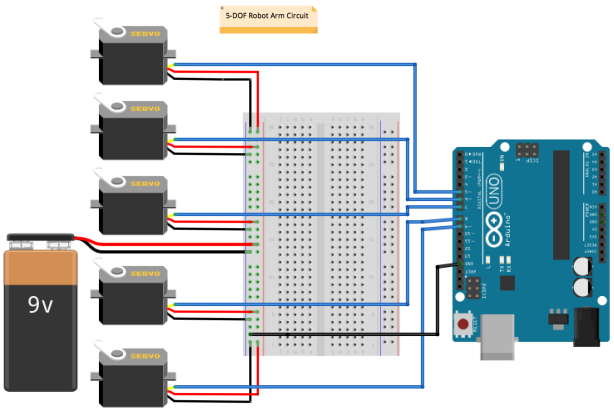

Fig. 5: 5-DOF Arm Robot Circuit

B. Graphical user interface

To build a friendly user interface, a MATLAB GUI with Support Package for Arduino Hardware is used in this project. The interface is shown in Figure 6 and users can interact with the robot arm easily in several ways.

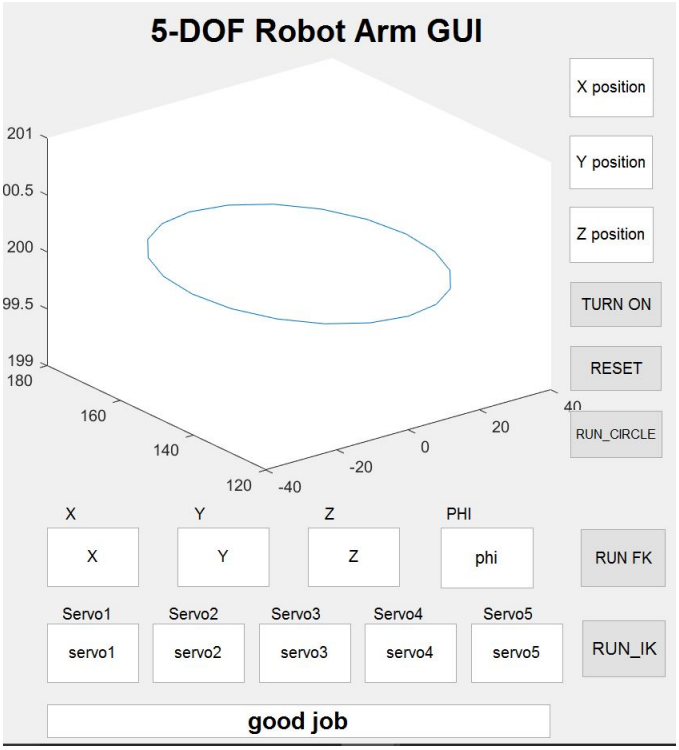

Fig. 6: Graphical user interface

First of all, users can enter five angles to the five servo motors respectively and press the 'Run' button, then the robot arm will rotate following the command. Besides, the coordinate position of the end-effector calculated by forward kinematics method will be shown in the three boxes right to the graph.

Secondly, users can set an end position for the robot arm to reach. Enter the x,y,z coordinate and angle of the end-effector in the four boxes and program will calculate the five rotation angles for each servo through the inverse kinematics method. If robot arm can not reach the position set by users, a warning of 'No Solution' will be shown in the bottom of interface. Otherwise, 'good job' message will tell users that the robot has reached the desired position successfully.

Moreover, the 'Reset' button will force the robot arm to get to the initial position. And the 'Run Circle' button will force the end-effector to move in a circular trajectory. And the 3D circular trajectory will be plotted on the GUI 3D graph, just like the Figure 6 shows.

V. Simulation

A. Modeling

We modeling the robot in Solidworks and export the connection part into `STL` file for 3D printing. And then we export the whole model into `x_t` file in order to import that into ADAMS for future kinematic simulation. We make full use of the virtual simulation technology by building the virtual prototype in ADAMS and improving the design efficiency of the robot arm.

B. ADAMS and MATLAB Co-simulation

Through the interface modular part `ADMAS/CONTROL`, we can establish the robot arm system in `MATLAB/simulink`. The coordinated simulation was successfully implemented. And the result of simulation(8) shows that the robot arm can track the specified trajectory well.

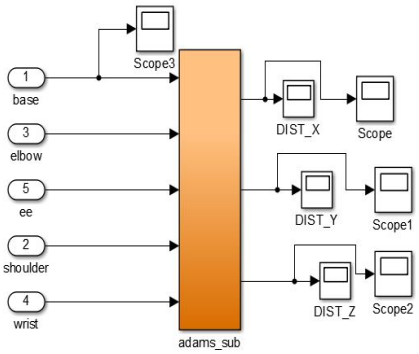

Fig. 7: Simulink block diagram

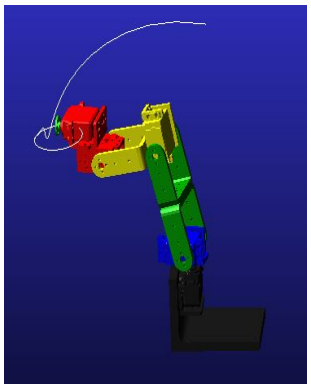

Fig. 8: Circle trajectory in ADMAS

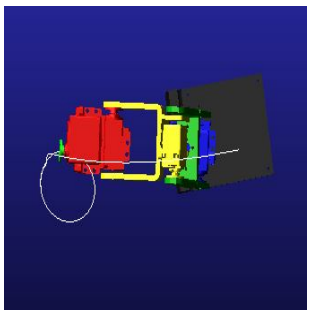

Fig. 9: Circle trajectory in ADMAS

VI. Results and Lessons Learned

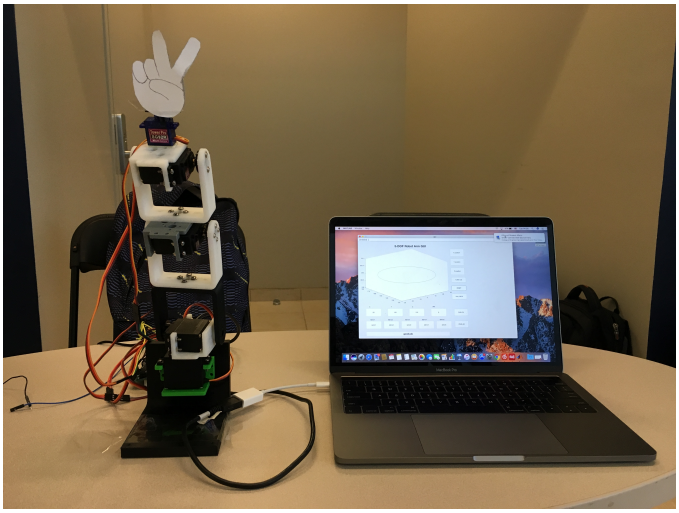

Fig. 10: Robot arm prototype

Figure 10 shows the prototype of 5-DOF robot arm. Because the servos we used in this project has no feedback, we can not draw a correct actual trajectory that the robot follows. But we can roughly compare the actual trajectory with the simulation result. Compared the actual trajectory with the ADAMS/MATLAB co-simulation results, we find that the robot can basically follow a command trajectory and finish drawing a circle. The error between the simulation and actual result comes from the servo accuracy and the stiffness of the robot structure. Because the robot is made by ABS material, and the first servo shaft is supposed to bear the weight of all the last four links and servos, so this first joint connection is not much stiff. That is also the main reason that the robot can not absolutely straight up when we set the robot move to the rest configuration, where five servos should all be set to zero. The lessons we learn from this project are as followings.

- For `Kinematics`, we learned to describe the position and orientation of bodies attached to the manipulator, and the rotation matrices and homogeneous transformations. Next, we knew the difference between direct kinematics and workspace and joint space and learned how to draw DH table. Besides, we learned to perform the inverse kinematics method to find joint variables in terms of end effector's position and orientation.
- For `Differential Kinematics and Statics`, we learned to describe the velocity of bodies attached to the manipulator and obtained manipulator Jacobians. Besides, we learned how to generate trajectory as planned.
- For `Control`, we learned to describe the relationship between generalized positions, velocities, accelerations, and forces. Meanwhile, we learned the techniques to specify the generalized forces developed by the actuators so as to effect a desired behavior of the robot
-
- For `Simulation`, we learned how to use ADAMS to build virtual prototype. In this way, we can implement any robotics related knowledge to the robot prototype and visualize the result. Also, we learned how to simulate the robot based on MATLAB and ADAMS. Co-simulation allows us to make the most of the numerical calculation and Simulink in MATLAB and dynamic simulation in ADAMS. Using co-simulation, we can implement complex control strategy related to both kinematics and dynamics.

VII. Conclusion

We have built a prototype of a 5-DOF non-parallel robot arm by 3D printing, which is controlled by Arduino UNO. We also implemented a friendly user interface in MATLAB with connection to Arduino. This interface allow users to control robot by specifying end-effector coordinates or each joint angle. Finally, we implemented MATLAB and ADAMS co-simulation of the robot. In this way, we can pass trajectory planning from MATLAB into the robot model and visualize the robot end-effector trajectory.

Acknowledgment

We would like to thank Prof. Brent Gillespie, Mr. Daniel Bruder and our fellow students in ME 567 course for the helping and inspiring us through out the duration of the project. We are grateful to the Prof. Brent Gillespie, University of Michigan for providing us with all the resources needed for the project.

References

[1] Spong, Mark W., Seth Hutchinson, and Mathukumalli Vidyasagar. Robot modeling and control. Vol. 3. New York: wiley, 2006.