Team Members: Ethan Chandler, Hushmand Esmaeili, Chaitanya Sriram, Girivaasan Chandrasekharan, Krishna Sathwik Durgaraju

RBE 501 Final Project Report: AR4 Robot Manipulator

Introduction

For our Robot Dynamics project, we began work for a long-term project involving a serial arm robot. Serial arm robots, or robot manipulators, are used in many different applications, including pick and place, grasping, soldering, and many other. They are used in different industries, from assembly in the car industry to chips industry to warehousing. They are "serial" robots because the arms have joints connected in series through links, and each joint has one or more actuators. There are many areas of research for robotic arms, including trajectory planning, vision-aided controls, dynamics, etc.

For this project, we decided to assemble a physical robotic arm, derive the kinematics and dynamics of the robot, and begin some work on trajectory planning. This project allowed us to apply the concepts and theory learned in class and work on the platform for future research. For this purpose, we acquired an Annin Robotics' AR4 6-DOF robotic arm. These robots are widely used in small automation processes and for research purposes.

In the next few sections, we will show the progress we did in assembling the robot and the kinematic and dynamic analyses work we did for it.

Hardware and Assembly Process

Putting together the manipulator took quite an effort. We started by assembling the joints and their bearings. Some integral components had to be 3D printed. Those parts went in between the joints along the motor. Each joint was integrated with stepper motors, and t. All the soldering were either covered with heat shrinks or liquid electric tapes to avoid short circuits.

All the wiring was covered with nylon-based material braids. Some of the machining works that we did while assembling were threading screw holes in the 3D printed parts, grinding to reduce material thickness, and pressing to insert the bearings into their slots.

Assembling the robot took more than 100+ hrs. of work. We could not complete the electronics integration on time for the final demonstration. However, we made considerable progress in setting up the platform for future work and research. Fig. 1 shows all the hardware components of the AR4 robot and our assembled robot.

Figure 1: Hardware parts of the AR4 robot and our assembly of the robot.

Kinematics

To derive the forward kinematics of the AR4, we used screw theory and product of exponentials (PoE) to apply the theory we had learned in lectures. The first step we took was to draw a kinematic diagram of the robot. Fig. 2 Shows the AR4 robot with a kinematic diagram of the same robot next to it. We first drew the base and end-effector frame, the joints and links and labeled the links dimensions. Then, we drew and labeled the screw axes $\{z_1,\ldots, z_6\}$. At this point we found the spatial twists, using the equation:

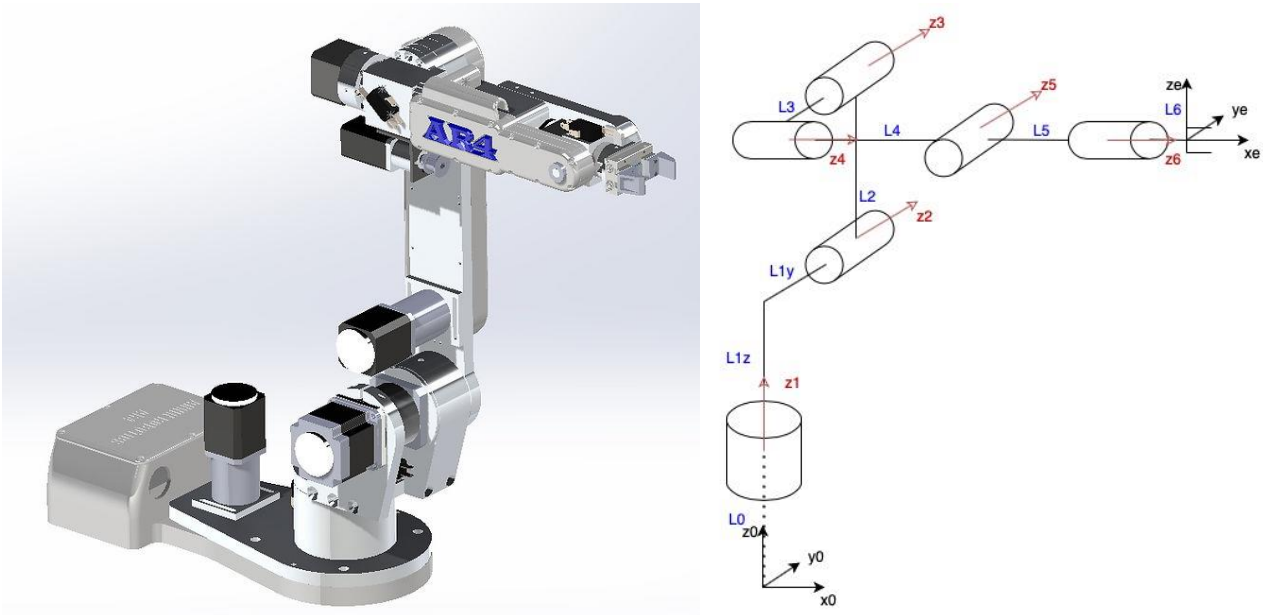


Figure 2: A render of the AR4 robot (left) and the kinematic diagram of the same robot (right)

$$\mathcal{V}_s = \begin{bmatrix} \omega_s \\ v_s \end{bmatrix} = \begin{bmatrix} \omega_s \\ \dot{p} - \omega_s \times (-p) \end{bmatrix}$$

The points *p* are symbolically given by

$$p_1 = [0, 0, L_0]^T$$

$$p_2 = [0, L_{1y}, L_0 + L_{1z}]^T$$

$$p_3 = [0, L_{1y}, L_0 + L_{1z} + L_2]^T$$

$$p_4 = [0, L_{1y} - L_3, L_0 + L_{1z} + L_2]^T$$

$$p_5 = [L_4, L_{1y} - L_3, L_0 + L_{1z} + L_2]^T$$

$$p_6 = [L_4 + L_5, L_{1y} - L_3, L_0 + L_{1z} + L_2]^T$$

Finally, we computed the spatial twists on MATLAB and the results are shown below:

```
Slist =

[0,          0,               0,             1,               0,             1]
[0,          1,               1,             0,               1,             0]
[1,          0,               0,             0,               0,             0]
[0, - l0 - l1z, - l0 - l2 - l1z,             0, - l0 - l2 - l1z,             0]
[0,          0,               0, l0 + l2 + l1z,               0, l0 + l2 + l1z]
[0,          0,               0,            l3,              l4,            l3]
```

These twists were verified using the intuitive, visual approach — “ninja” way.

After finding the special twists, we found the home position matrix to be used in the forward kinematics (FK) computation. Finally, we found the forward kinematics of the 6-DOF robot using PoE. The screenshot below shows the MATLAB script we used to find the symbolic FK using PoE. We do not include the symbolic solution in this report because it is too long. The kinematic solution was further verified by our DH solver.

```
function P = AR4_fk(q)

    syms L0 L1z L1y L2 L3 L4 L5 L6 'real'

    % Joint limits

    S1 = [0, 0, 1, 0, 0, 0]';
    S2 = [0, 1, 0, -(L0 + L1z), 0, 0]';
    S3 = [0, 1, 0, -(L0 + L1z + L2), 0, 0]';
    S4 = [1, 0, 0, 0, (L0 + L1z + L2), (L1y - L3)]';
    S5 = [0, 1, 0, -(L0 + L1z + L2), 0, L4]';
    S6 = [1, 0, 0, 0, (L0 + L1z + L2), (L1y - L3)]';

    S = [S1 S2 S3 S4 S5 S6];

    M = [1 0 0 (L4+L5+L6);
         0 1 0 (L1y-L3);
         0 0 1 (L1z+L2);
         0 0 0 1];

    P = fkine(S,M,q);
end
```

Dynamics

This serial arm was a test subject for a much larger framework developed by members of the team. This framework aims to take in a set of generic robot parameters, such as link lengths, parallelism, robot compliance, and actuator models, to generate a set of kinematic and dynamic equations. Furthermore, we are working on a generic optimal controller to simulate and control this general robot model. A critical aspect of this framework was devised in this course: a Recursive Newton-Euler solver. The implementation closely follows the work described in [Wisama Khalil. DYNAMIC MODELING OF ROBOTS USING RECURSIVE NEWTON-EULER TECHNIQUES. ICINCO2010, Jun 2010, Portugal. ffhal-00488037f] in Matlab.

Strangely, while we were researching this algorithm and similar ones (such as the Floating Base approach introduced by Featherstone and the recursive Lagrangian), we noticed that many implementations of these algorithms have a lot of unnecessary computations; namely, they go through the recursive procedure every single time they want to compute the inverse dynamics of the system. One can observe significant speedups by simply computing the recursive procedure once at the start of the program using symbolic variables and generating a hardcoded function from those final symbolic inverse dynamic functions. This is extremely simple to do in Matlab using the matlabFunction() function, which takes in a symbolic equation and generates a function from it. In fact, this particular feature in Matlab is a large inspiration behind this general framework because it allows for us to write everything symbolically and then automatically turns those symbolic equations into optimized, robot-specific functions which the user can plug robot-specific parameters into. Here is a snippet of our RNE implementation (partially adopted from Professor Fichera's example code):

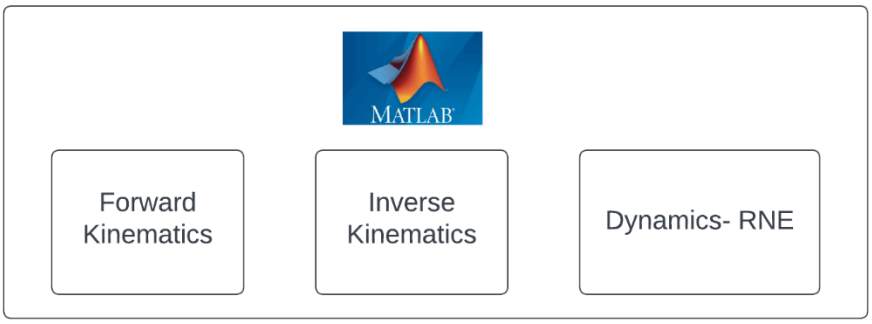
```
function tauSym = getInvDyn(self, qSym, dqSym, ddqSym, g, FtipSym, Mlist, Glist, Slist)
    n = self.dof;
    Mi = sym(eye(4));
    Ai = sym(zeros(self.space, n));
    AdTi = sym(zeros(self.space, self.space, n + 1));
    Vi = sym(zeros(self.space, n + 1));
    Vdi = sym(zeros(self.space, n + 1));
    Vdi(self.space/2 + 1: end, 1) = sym(-g);
    AdTi(:, :, n + 1) = vpa(utils.Adjoint(utils.TransInv(Mlist(:, :, n + 1))));
    Fi = FtipSym;
    tauSym = sym(zeros(n, 1));
    for i=1 : n
        Mi = Mi * Mlist(:, :, i);
        Ai(:, i) = vpa(utils.Adjoint(utils.TransInv(Mi))) * vpa(Slist(:, i));
        AdTi(:, :, i) = vpa(utils.Adjoint(utils.MatrixExp6(utils.VecTose3(Ai(:, i) ...
                        * -qSym(i))) * utils.TransInv(Mlist(:, :, i))));
        Vi(:, i + 1) = AdTi(:, :, i) * Vi(:, i) + Ai(:, i) * dqSym(i);
        Vdi(:, i + 1) = AdTi(:, :, i) * Vdi(:, i) ...
                        + Ai(:, i) * ddqSym(i) ...
                        + utils.ad(Vi(:, i + 1)) * Ai(:, i) * dqSym(i);
    end
    Ai = vpa(simplify(expand(Ai),100));
    AdTi = vpa(simplify(expand(AdTi),100));
    Vi = vpa(simplify(expand(Vi),100));
    Vdi = vpa(simplify(expand(Vdi),100));

    for i = n : -1 : 1
        Fi = transpose(AdTi(:, :, i + 1)) * Fi + Glist(:, :, i) * Vdi(:, i + 1) ...
             - transpose(utils.ad(Vi(:, i + 1))) * (Glist(:, :, i) * Vi(:, i + 1));
        tauSym(i) = transpose(Fi) * Ai(:, i);
    end
    tauSym = simplify(expand(tauSym), 100);
end
```

We would like to emphasize that this function does not return any numerical values (in fact, if you give it actual numbers it will intentionally return an error), but instead it returns a symbolic matrix. This symbolic matrix is then simplified and optimized via the Matlab symbolic toolbox. The program then turns this symbolic matrix into a function handle, and then the user is allowed to replace the symbolic variables with real numbers. Most notably, the function handle only has to be made once at the start of the program.

Software Framework and Implementation

The overall robot framework is object-oriented and programmed in Matab in order to save development time and was implemented on various robots across three classes this semester (RBE 501, 502, and 521). The framework will be translated into beautiful C++ code and released as a library in future work. The framework currently features a symbolic POE and DH forward kinematics solver, a Levenberg-Marquadt implementation (with plans to add a full-fledged Broyden-Fletcher-Goldfarb-Shanno solver), a contact force simulator for our dynamics engine, and the ability to visualize parallel, serial, and hybrid (legged) robots. While outside the scope of this class, the framework also contains a controller interface for utilizing modern control

theory to find optimal inputs to control our robots (including MPC and an adaptive impedance controller).



Challenges and Future Work

One of the main challenges we faced was being unable to complete the robot assembly in time for the final demonstration. During the assembly process, we faced several hardware challenges that delayed the assembly of the robot. For example, the J6 housing part got bent during the pressing process. Unable to install J6, we had to wait until we 3D printed a new part and received the expedited mail delivery of new bearings. Moreover, the assembly process in general took much longer than we anticipated, and we were unable to complete the electronics part of the assembly process on time for the final demonstration.

Despite the challenges, we made major progress in the assembly process of the robot, and most importantly, we learned to apply various theoretical concepts learnt in class. In class, we would often take images of serial arm robots and apply our knowledge of FK and dynamics to solve problem sets or programming assignments. In this project, we got to work hands-on with one of these serial arm robots and follow the procedures learnt in class to derive the forward kinematics, the dynamics (through RNE), as well as develop a software framework to compute the kinematics and dynamics logic of the robot.

Regarding the future work of this project, we hope to complete the electronics portion to be able to interface implement our software framework in C++ to upload to the robot microcontrollers. Then, we can test how well our kinematics and dynamics work. At that point, we hope to develop motion planning and trajectory generation algorithms. Eventually, by combining the motion planning and integrating a computer vision (CV) system to the robot, we hope to run a chess engine AI so to have the robot play chess. There are many areas of research that we hope to explore with this platform, and this class gave us the foundations — in terms of kinematics and dynamics theory — as well as the opportunity to work on assembling the robot for future work.