1 Mechanical Design

The manipulator is 7R spatial. The design is inspired by the human arm. It only possesses single degree-of-freedom joints, but the three joints representing the shoulder, two joints representing the elbow, and two joints representing the wrist are in close proximity. The shoulder, elbow, and wrist joint groups are separated by longer links. Also, neighboring joint axes intersect. The result is a manipulator like the Kuka LBR iiwa.

The manipulator was modeled in Solidworks. Each link with exception to the base and end are configurations of the same part. Each joint uses the same hardware. This modularity is inspired by the CMU snake robot. The joint is designed so the shaft is fixed with respect to one link and rotates with respect to the other on plain bearings. The shaft is a hex head screw, and the axial play is controlled with a locknut. Table 1 is the bill of materials.

Table 1: Bill of materials.

Line	Description	Quantity
1	Base	1
2	Link 1	1
3	Link 2	3
4	Link 3	2
5	End	1
6	1/4”-20 x 2.25” Hex Head Screw (McMaster-Carr 91257A551)	7
7	1/4”-20 Nylon-Insert Locknut (McMaster-Carr 94945A205)	7
8	Plain Bearing, 1/4” Shaft, 3/8” Housing, Flanged (McMaster-Carr 1677K1)	14

The links were printed on my personal Prusa i3 MK3S+ printer. The links are in alternating colors – including Aggie maroon – to better visualize the joints. The manipulator is currently at my desk in the Research Integration Center (RIC) at TAMUS RELLIS.

Figure 1: Physical manipulator. Right image shows the manipulator at my desk at the RIC.

2 Kinematics

Coordinate frames were defined in Solidworks using modified D-H parameters. These coordinate frames were used with the Solidworks to URDF exporter add-in so the URDF would be defined with the D-H frames. Figure 2 shows the frames. Even and odd frames are shown in different subfigures for better visualization. Table 2 shows the corresponding modified D-H parameters.

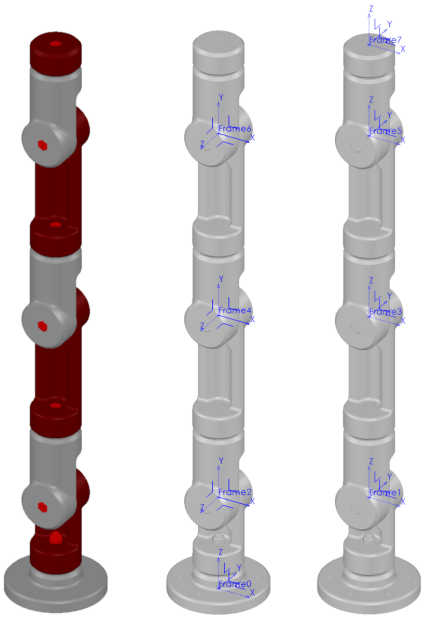

Figure 2: Frame definitions. Middle subfigure shows even frames. Right subfigure shows odd frames.

Table 2: Modified D-H parameters.

Joint i	α_{i-1}	a_{i-1}	d_i	θ_i
1	$0°$	0	$l_1 + b + l_2$	q_1
2	$90°$	0	0	q_2
3	$-90°$	0	$l_2 + b + l_3$	q_3
4	$90°$	0	0	q_4
5	$-90°$	0	$l_2 + b + l_3$	q_5
6	$90°$	0	0	q_6
7	$-90°$	0	$l_2 + b + l_1$	q_7

In Table 2, $b = \frac{1}{16}$" is the plain bearing flange thickness, $l_1 = 1$", $l_2 = 3$", and $l_3 = 5$".

The transforms between each frame necessary for forward kinematics are calculated below.

$$ {}^{0}_{1}T = \begin{bmatrix} cq_1 & -sq_1 & 0 & 0 \\ sq_1 & cq_1 & 0 & 0 \\ 0 & 0 & 1 & l_1 + b + l_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} cq_1 & -sq_1 & 0 & 0 \\ sq_1 & cq_1 & 0 & 0 \\ 0 & 0 & 1 & 4.0625 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

$$ {}^{1}_{2}T = {}^{3}_{4}T = {}^{5}_{6}T = \begin{bmatrix} cq & -sq & 0 & 0 \\ 0 & 0 & -1 & 0 \\ sq & cq & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} cq & -sq & 0 & 0 \\ 0 & 0 & -1 & 0 \\ sq & cq & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2} $$

$$ {}^{2}_{3}T = {}^{4}_{5}T = \begin{bmatrix} cq & -sq & 0 & 0 \\ 0 & 0 & 1 & l_2 + b + l_3 \\ -sq & -cq & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} cq & -sq & 0 & 0 \\ 0 & 0 & 1 & 8.0625 \\ -sq & -cq & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3} $$

$$ {}^{6}_{7}T = \begin{bmatrix} cq_7 & -sq_7 & 0 & 0 \\ 0 & 0 & 1 & l_2 + b + l_1 \\ -sq_7 & -cq_7 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} cq_7 & -sq_7 & 0 & 0 \\ 0 & 0 & 1 & 4.0625 \\ -sq_7 & -cq_7 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4} $$

where $q = q_i$ for ${}^{i-1}_{i}T$ and s and c represent sin and cos, respectively.

3 Control

3.1 Description

A computed torque controller is implemented. Figure 3 is a diagram of the controller. The manipulator

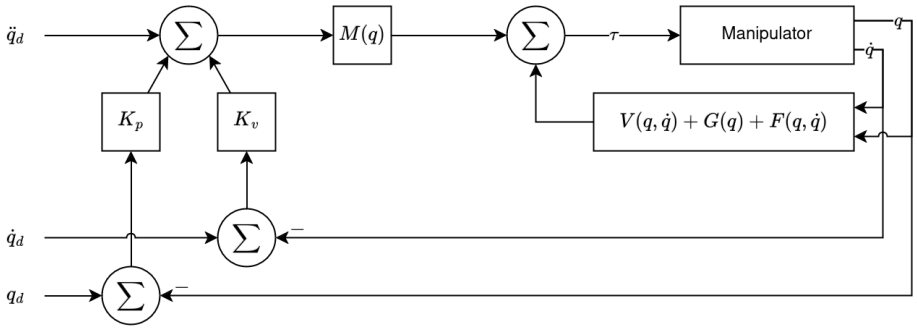

Figure 3: Computed torque diagram.

dynamics are

$$M(q)\ddot{q} + V(q,\dot{q}) + G(q) + F(q,\dot{q}) = \tau \tag{5}$$

where M is the inertia matrix, V contains centrifugal and Coriolis terms, G contains gravity terms, and F contains friction terms. The control law is

$$\tau = M(q)[\ddot{q}_d + K_v(\dot{q}_d - \dot{q}) + K_p(q_d - q)] + V(q,\dot{q}) + G(q) + F(q,\dot{q}) \tag{6}$$

The controller cancels the nonlinear dynamics and adds proportional and derivative error terms. The resulting closed-loop dynamics are

$$\ddot{E} + K_v\dot{E} + K_pE = 0 \tag{7}$$

where $E = q_d - q$. This is a linear second-order system.

3.2 Stability

Since the computed torque controller produces a closed-loop linear second-order system, the system is stable if the gains are chosen such that the roots have negative real components. For the implemented controller, gains are chosen to achieve critical damping:

$$K_v = 2\sqrt{K_p} \tag{8}$$

Stability can also be proven using Lyapunov. A candidate Lyapunov function is

$$v = \frac{1}{2}\dot{E}^\top\dot{E} + \frac{1}{2}E^\top K_pE \succ 0 \tag{9}$$

This function is positive definite since K_p is positive definite. Additionally,

$$v = \begin{bmatrix} E \\ \dot{E} \end{bmatrix}^\top \begin{bmatrix} K_p & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} E \\ \dot{E} \end{bmatrix} > \beta \left\| \begin{matrix} E \\ \dot{E} \end{matrix} \right\|^2 \tag{10}$$

where $\beta = \min(\lambda_{K_p,\min}, 1)$ and $\lambda_{K_p,\min}$ is the minimum eigenvalue of K_p. The derivative of the candidate function is

$$\dot{v} = \dot{E}^\top\ddot{E} + E^\top K_p\dot{E} \tag{11}$$

Plugging in the system dynamics $\ddot{E} = -K_v\dot{E} - K_pE$,

$$\begin{aligned}\dot{v} &= \dot{E}^\top(-K_v\dot{E} - K_pE) + E^\top K_p\dot{E} \\ &= -\dot{E}^\top K_v\dot{E} \preceq 0\end{aligned} \tag{12}$$

The derivative is negative semi-definite since K_v is positive definite. Steady-state analysis showing the system cannot get stuck away from the fixed point is needed to prove asymptotic stability by LaSalle's Theorem. The system can only get stuck if $\ddot{E} = \dot{E} = 0$, which requires $E = 0$ from (7). Therefore, the system is asymptotically stable.

3.3 Simulation

The manipulator was simulated using PyBullet. I wanted to use an environment other than MATLAB; I have used it in industry, but research seems to favor Python and simulators like PyBullet and MuJoCo seem more popular. I used the student version of CoppeliaSim for another robotics course, so I wanted to experiment with something else. I initially tried using the Python Robotic Toolbox developed by the lead developer of MATLAB's Robotics Toolbox, Peter Corke. While most of the functionality worked, the package was unusable because of a bug in getting the mass matrix from a robot imported via URDF. [1] There were other issues too, such as not being compatible with the latest release of SciPy.

The following are links to the flopping videos:

- downflop_bullet.mp4
- upflop_bullet.mp4

The flopping videos were also created using MATLAB before the PyBullet simulation was working.

Figure 4 is an impulse response plot created by perturbing each joint with a large torque at the 2 second mark while under position regulation control. The following is a link to the video of the scenario used for creating the plot: impulse.mp4. The non-zero error at the beginning of the plot is due to the initial

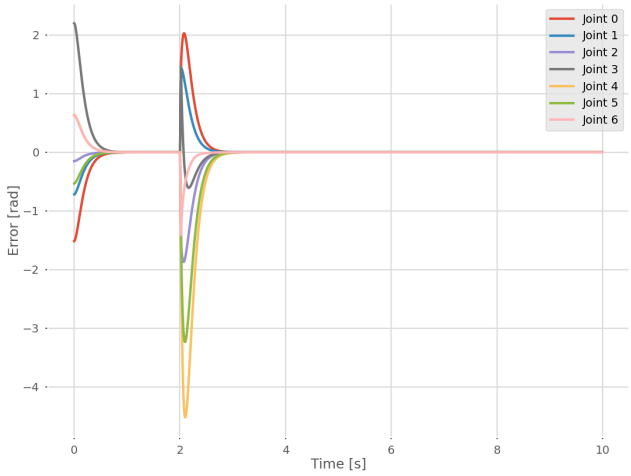

Figure 4: Impulse response.

movement of the manipulator to the desired Cartesian pose.

Figure 5 and Figure 6 are plots of the manipulator's path when trying to draw a pentagram in a plane normal to the x-axis. The following is a link to the video of the scenario used for creating the plot: cartesian_trajectory.mp4 In the figures, the target path is red and the actual path is blue. The error is much greater along the x-axis, but the maximum error is less than 3 mm.

[1] GitHub issue

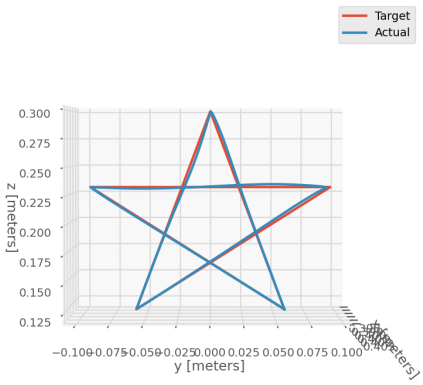

Figure 5: Cartesian path viewed normal to plane of pentagram.

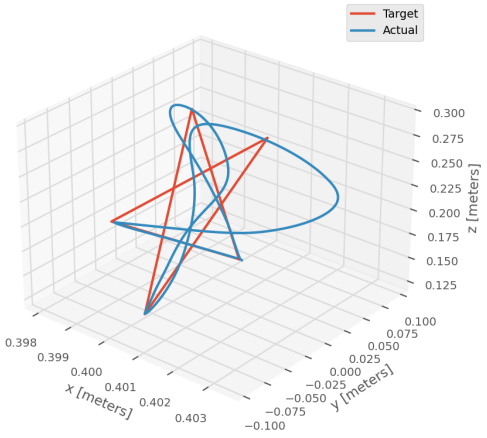

Figure 6: Cartesian path in 3D.