MEGN544 - Project Part 3 Report

Student: Thong Quoc (Bill) Huynh

November 02, 2023

As discussed in Project Part 1, with the given example points (start of C letter and end of M letter) in both the world frame and the unscaled letter frame, I calculated a vector in each frame and then calculated the ratio of the vector norms to find the scaling factor:

$$\vec{v}_{unscaled\ letter} = \begin{bmatrix}10\\0\\0\end{bmatrix} - \begin{bmatrix}2.5\\4\\0\end{bmatrix} = \begin{bmatrix}7.5\\-4\\0\end{bmatrix}$$

$$|\vec{v}_{unscaled\ letter}| = 8.5$$

$$\vec{v}_{world} = \begin{bmatrix}0.14\\-0.3\\0.4\end{bmatrix} - \begin{bmatrix}-0.01\\-0.3\\0.48\end{bmatrix} = \begin{bmatrix}0.15\\0\\-0.08\end{bmatrix}$$

$$|\vec{v}_{world}| = 0.17$$

$$scalingfactor = \frac{|\vec{v}_{world}|}{|\vec{v}_{unscaled\ letter}|} = 0.02$$

I multiplied this scaling factor to the 2-dimensional letter points' coordinates to get scaled coordinates (X and Y) and added a third Z-axis coordinate of 0 to make the points 3-dimensional. These scaled coordinates could then be considered as coordinates in the scaled letter frame and ready for projection on to the world frame.

I called the "scaled letter frame" "frame 1" and "world frame" "frame 2". I could then begin calculating the transformation matrix as follows:

$${}^1T_2 = \begin{bmatrix}{}^1x_2 & {}^1y_2 & {}^1z_2 & {}^1d_{12}\\0 & 0 & 0 & 1\end{bmatrix}$$

By inspection of the world frame relative to the letter frame:

$${}^1x_2 = {}^1x_1 = \begin{bmatrix}1\\0\\0\end{bmatrix}$$

$${}^1y_2 = -{}^1z_1 = \begin{bmatrix}0\\0\\-1\end{bmatrix}$$

$${}^1z_2 = {}^1y_1 = \begin{bmatrix}0\\1\\0\end{bmatrix}$$

Therefore:

$${}^1T_2 = \begin{bmatrix}1 & 0 & 0 & \\0 & 0 & 1 & {}^1d_{12}\\0 & -1 & 0 & \\0 & 0 & 0 & 1\end{bmatrix}$$

Also:

$$
\begin{aligned}
{}^1d_{12} &= {}^1d_{1C} + {}^1d_{C2} \\
&= {}^1d_{1C} - {}^1R_2\ {}^2d_{2C} \\
&= 0.02 \times \begin{bmatrix} 2.5 \\ 4 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} -0.01 \\ -0.3 \\ 0.48 \end{bmatrix} \\
&= \begin{bmatrix} 0.06 \\ -0.4 \\ -0.3 \end{bmatrix}
\end{aligned}
$$

Therefore:

$$
{}^1T_2 = \begin{bmatrix} 1 & 0 & 0 & 0.06 \\ 0 & 0 & 1 & -0.4 \\ 0 & -1 & 0 & -0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

To project positions from the scaled letter frame (1) to the world frame (2), I needed:

$$
{}^{world}T_{scaled\ letter} = {}^2T_1 = {}^1T_2^{-1} = \begin{bmatrix} 1 & 0 & 0 & -0.06 \\ 0 & 0 & -1 & -0.3 \\ 0 & 1 & 0 & 0.4 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

I multiplied ${}^2T_1$ with the 3D scaled letter frame positions to project all the points to the world frame and double checked to verify that the world positions of the C start point and M end point matched what was given.

With the world positions calculated, I saved the points as file points3D.mat with a field named points3D, using my algorithm already written in Project Part 1.

At each world frame point, I plotted the column vectors of ${}^{world}R_{scaled\ letter}$ with another rotation around Z-axis appended to orient each local X-axis towards the next point. The angle of Z-axis rotation was found through the angle between the previous vector and the next vector at each point (cross product for sine, dot product for cosine, and then atan2).

There is a difference from Project Part 1. The angle in the Z-axis rotation appended to each local coordinate frame in the CSM 3D way points now has an opposite sign compared to Project Part 1. This is because Project Part 1 requires the local Z-axes to point towards the front of the CSM letters (out of the plane), while in this Project Part 3 we need to trace the CSM with the local Z-axes pointing into the plane. This is more natural to represent how the wrist of the robot arm would point into the board as it traces the letters CSM.

The control approach in this part of the project is to use Constant Acceleration Interpolation to interpolate the theta targets between the way points of the 3D CSM trajectory. The way points of the trajectory are calculated using dhInvKine functions, with the link list, desired transform (from 3D points and project Part 1 rotation matrices), and parameter guess list (which could be the last parameter list). The Trajectory Generator, represented as the Trajectory Generator block in the MATLAB Simulink block diagram in Figure 1, has access to the trajectory (as a variable in the MATLAB base workspace). The Trajectory Generator code is implemented to utilize the constAccelInterp() function (written in Project Part 2) to interpolate between the way points and time stamps in the trajectory to generate theta and theta dot targets. The Theta Controller's block diagram is shown in Figure 2 for reference.

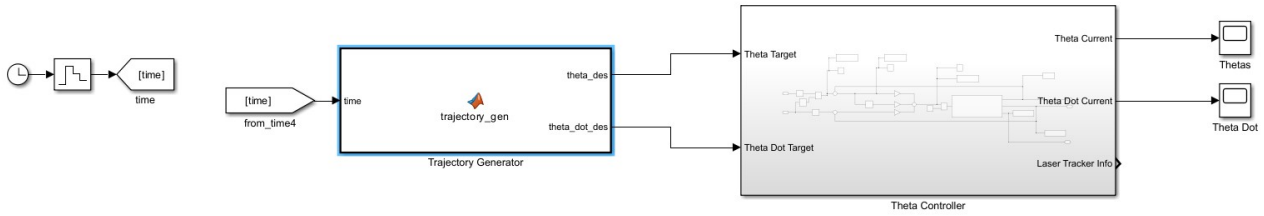

Figure 1: Screenshot of top-level block diagram used in MATLAB Simulink.

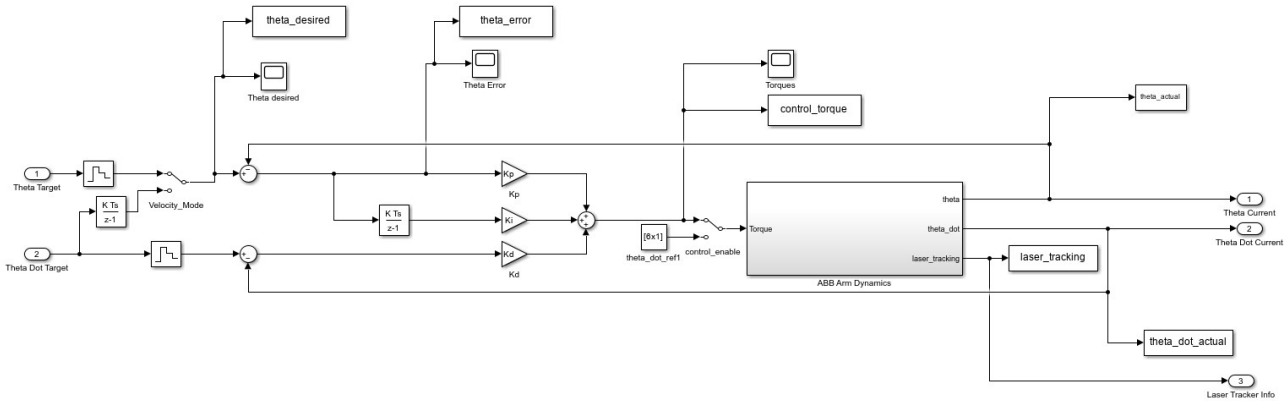

Figure 2: Screenshot of Theta Controller block diagram used in MATLAB Simulink.

Metaphorically, it is as if the encoders ask the Trajectory Generator at each point in time, where the encoders should set the joint angles to be (passing the "time" variable into the Trajectory Generator), and the Trajectory Generator uses the time to interpolate using Constant Acceleration Interpolation to find the answers. From interpolation, the theta and theta dot target outputs from the Trajectory Generator are used as inputs for the Theta Controller block, which controls where and how fast the arm moves (actual theta and actual theta dots), to trace the CSM letters.

Figure 3 and Figure 4 are included to show a comparison between desired and actual theta paths. The actual theta path corresponds to the desired theta path with acceptable amounts of errors and overshoots. All theta's show a significant shift at the beginning near 0 second, where the arm needs to quickly move from initial configuration to the start of the C letter. Another large shift is seen between 14 seconds and 16 seconds for each joint, which is when the arm finishes tracing CSM and moves towards DH Zero configuration as desired. Other than that, θ_4 path shows a discontinuity around 9 seconds, which could be due to joint limit wrapping. θ_6 paths shows many discontinuity points over period of action time. This is due to the wrist being rotated continuously to orient the local x-axis towards the next point in the desired CSM trajectory.

The initial configuration is left as is from Simulink. The CSM points are transformed from the given 2D points. The final configuration is the DH Zero Configuration, which corresponds to zero thetas except for $\theta_2 = \frac{\pi}{2}$ because of the offset in the ABB arm.

The MATLAB code files mentioned are included in the submission package.

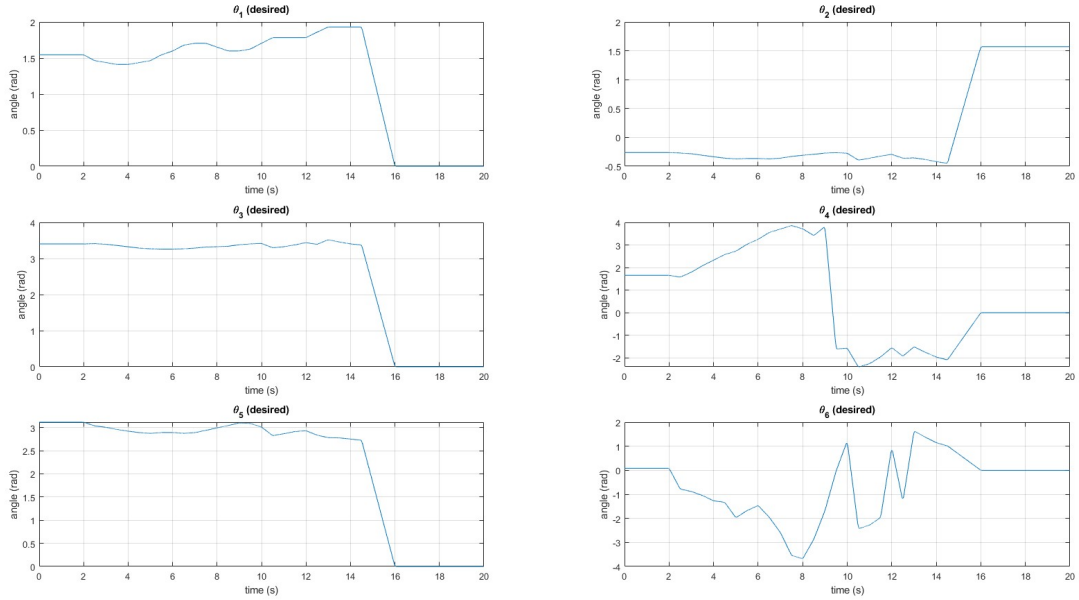

Figure 3: Graph of desired θ's over time, with θ_1 at top left, θ_2 at top right, and so on.

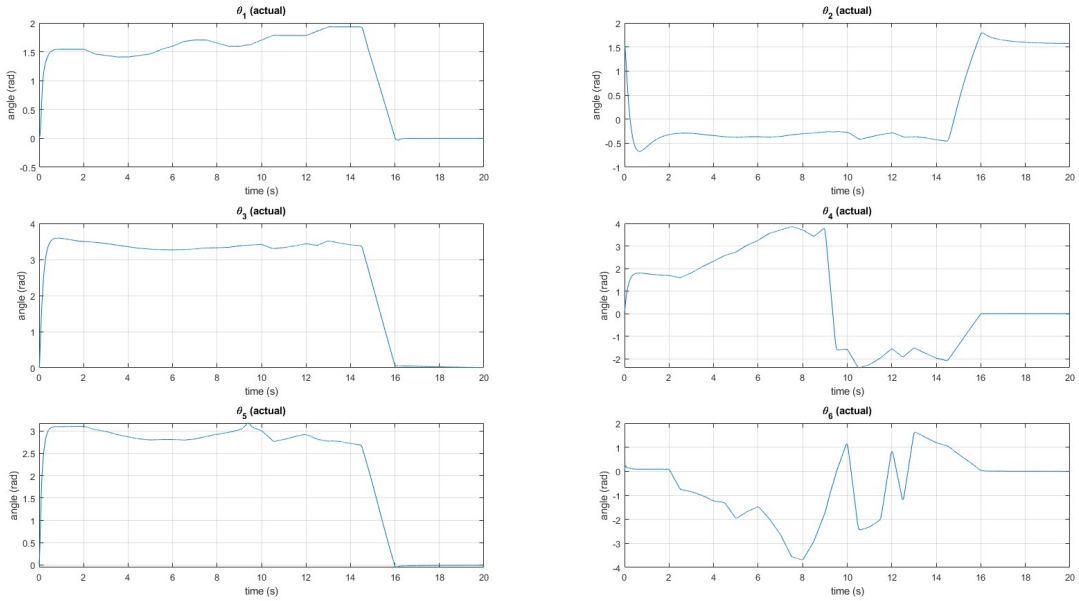

Figure 4: Graph of actual θ's over time, with θ_1 at top left, θ_2 at top right, and so on.

For the CSM trajectory to be completed reliably using the Constant Acceleration Interpolation approach, my trial results show me that it needs a minimum of approximately 1.5 seconds between the arm's initial configuration and the C starting point, 0.5 second stopping in the desired theta here to actually get close to the C starting point.

Then it needs at least 0.2 second between each pair of consecutive trajectory way points in the 26 CSM points. Finally, it requires minimally 1 second to get from the M end point to the DH Zero configuration and another 0.5 seconds to come to a stop at the DH Zero configuration. **This amounts to a total of 8.5 seconds in the desired trajectory - a minimum time** for the arm to trace the CSM letters and move between the start and end configurations reliably. For the movie of the simulated robot arm in action, I set 0.5 second between each pair of consecutive CSM points.

Too high a speed will result in low fidelity of the CSM tracing. This means the actual CSM tracing has too much discrepancy from the ideal CSM plot, because the Constant Acceleration Interpolation is going through the trajectory at a high speed and with sharp turns. Admittedly, Constanst Acceleration Interpolation sacrifices the accuracy of hitting target points for a smoother actual trajectory. However, impractically high speed with make the arm rush through the trajectory in an unstable way, and the result trajectory will no longer resemble the goal trajectory.

From my observation, **passing theta dot targets (from Constant Acceleration Interpolation, instead of just zeros) into the controller helps reduce overshoot in the actual theta's**. Although the differences in the videos of the robot arm motion are not obvious, I can see in the graphs of actual theta's that the changes are sharper and the joint angles seem to be directed towards their next goals more readily.

The resulting tracing, as seen in Figure 5, matches the desired CSM plot fairly well, although not perfect. The arm needs to make big a movement to get from its initial configuration (decided by MATLAB Simulink) to the starting point of the letter C and again when going back to the DH Zero configuration after finishing the letter M. There is also a small looping trace at the top left corner of the S letter. This could be due to some discontinuity in the theta paths, or the arm simply has to wrap around when it reaches a joint limit. This "looping" effect is intensified if I set the theta wrapping in dhInvKine() to be strictly from $-\pi$ to π. Therefore, to achieve the result shown in Figure 5, I increased the limits (before enforcing wrapping) to $-1.3 \times \pi$ and $1.3 \times \pi$. Another observation is that, it seems like the arm's initial configuration (not set by me) coincides with the DH zero configuration set by me for the robot arm to end at, after tracing CSM.

The video of the robot arm in action is included in the submission package as file "arm_motion_CAI.mp4".

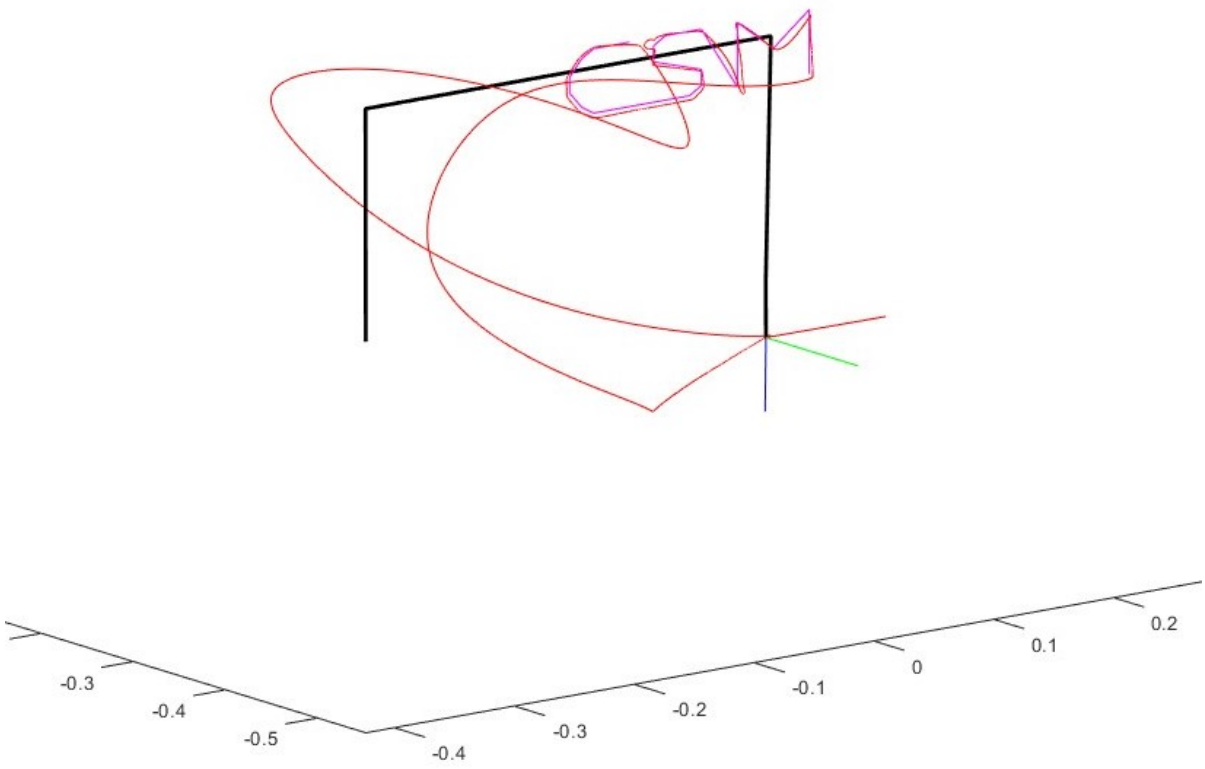

Figure 5: The result tracing of the CSM trajectory.