Robot Arm Build, Winter Quarter 2020
Team: Lindie Burgess, Brita Hill, Vadim Naumchuk

Our project goal was to build an arm, write code to communicate with the arm, and deploy a script to it that would enable the arm to write an arbitrary collection of 5 of the first 10 letters from the English alphabet.

Our robot arm is comprised of 6 robot actuators from Dynamixel: 2 MX-64 actuators and 4 AX-12 actuators. We used SolidWorks to model our links which were 3D printed. The motors were controlled by an OpenCM9.04 Arduino board, scripts were written in the Arduino IDE to ping and control the motors, and scripts to orient the motors and send large sequences of information to the motors were written in and deployed from MATLAB.

Early in the project, we prioritized whiteboarding our plans for the orientation of the motors so we could maximize its configuration space and degrees of freedom to ensure ultimate functionality. We placed the 2 MX-64 actuators at the base of the arm. These were followed by the 4 AX-12 motors at different orientations demonstrated here:

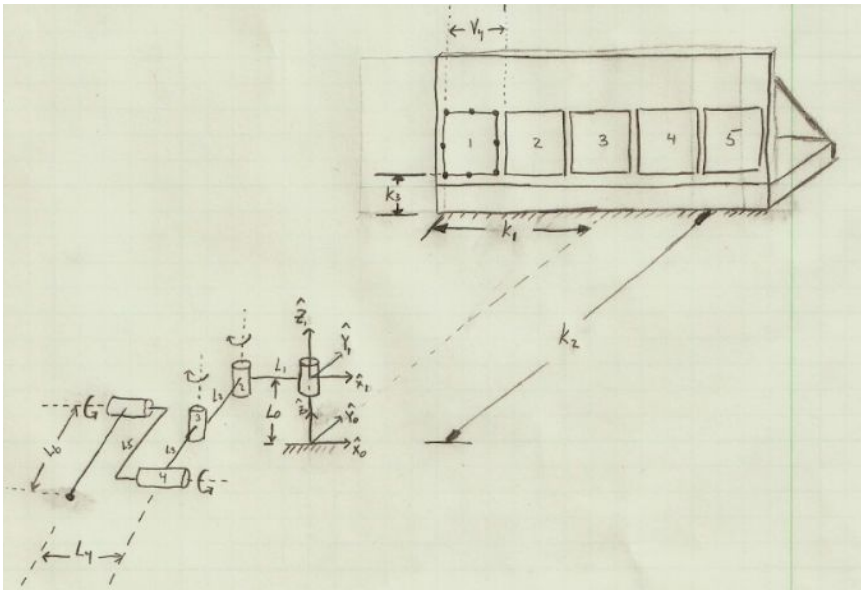

Figure 1: Orientation of robot in home position relative to whiteboard.

The spatial frame is defined such that the x-axis is parallel to the top and bottom of the whiteboard, the y-axis runs parallel to the distance between the robot and the whiteboard, and the z-axis runs parallel to the left and right sides of the whiteboard.

Once the arm was assembled, we brainstormed methods to code the first 10 letters of the English alphabet in a way that could be transmitted reliably to the robot.

We decided to model our font off LED light display letters to maximize readability.

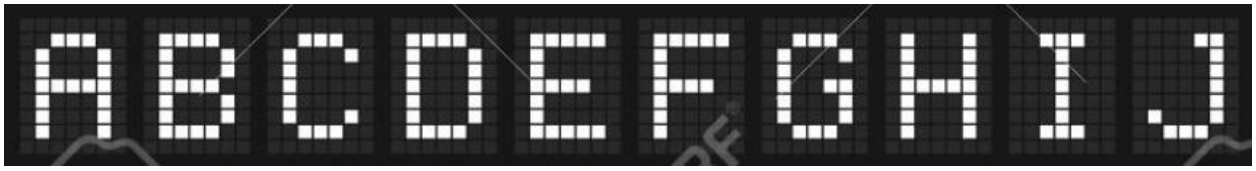

Figure 2: LED light display example

We calculated, in millimeters, the distance from the base of the arm to the center of the leftmost letter space on the whiteboard. Around this centerpoint, we created a rectangle with points along its edge that would allow us to vectorize each letter we planned to write.

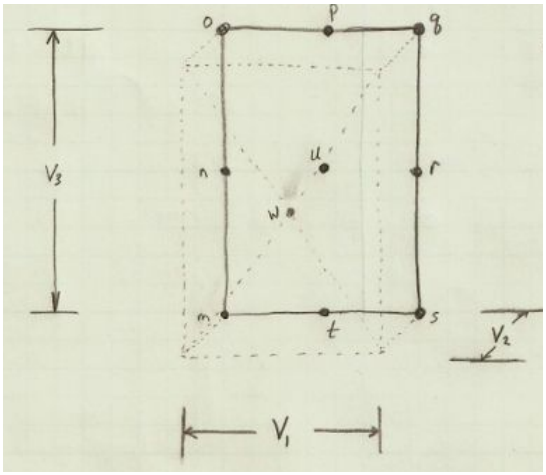

Figure 3: Letter space on the whiteboard with centerpoint u

Vector **u** is defined as the centerpoint of the letter position, while vector **w** exists as point u translated length v2 from the whiteboard. Vector **w** is useful when it comes to letters which require multiple strokes of the pen to create, or when transitioning from one letter position to the next.

Once we had outlined the variables that would describe vectors for each letter we planned to write, we input these vectors to matrices named after the letter they represent.

```
A = [u,w,s,q,o,m,w,n,r,w];

B = [u,w,n,o,q,u,s,m,n,u,w];
```

Figure 4: Matrix of vectors describing letters A and B

Referring to vectors via variables allowed us to experiment with distances and fine-tune the calibration for each component of each letter. It also allowed us to recycle common gestures (i.e. upstroke from bottom left corner of letter box to top left corner of letter box, employed by 8 of the 10 letters).

```
A =

     0     0    15    15   -15   -15     0   -15    15     0
   395   375   395   395   395   395   375   395   395   375
   180   180   150   210   210   150   180   180   180   180
```

Figure 5: Matrix of vectors that describe the letter A

Programming letters this way also allowed us the freedom to use the same letter code regardless of the position the letter was being written on the board. Since all the letter vectors depend on the location of the center point, we simply iterated the centerpoint of each letter box across the board.

Before a letter was written, its matrix was passed through a function, LinSpace, which discretized between each point. To get the most clarity, we tasked LinSpace with creating 16 intermediate points between each original letter vector. Each of these vectors became the 4th column of a T matrix, each of which was stored as a 3D matrix of T's.

```
T =

     1     0     0     0
     0     1     0   395
     0     0     1   180
     0     0     0     1
```

Figure 6: T matrix

Since we knew the location of the end-effector, we used inverse kinematics to calculate the orientation of each motor to accomplish the correct angles to put the end-effector in the positions we needed. We experimented with many initial theta guesses for each of the 5 motors that would dictate the location of the end-effector (the 6th motor simply opened and closed the end-effector clamp that held the pen).

We used an IKinSpace function that processed our T theta guesses using root-finding and either returned success if in 20 iterations or less it had discovered an orientation that would allow us to achieve the correct end-effector location or failure if it was unable to find a root.

Once the inverse kinematics function was successful, we relayed the orientation of each motor to the Arduino board so that the arm would be positioned in the orientation we requested.

As we tested each letter, we discovered that the velocity of each motor needed to be controlled to avoid sending the arm into an oscillating gesture that obscured the clarity of our letters. We controlled the velocity of each motor so they would never have enough power to get stuck in this kind of loop.

Future functionality

We learned a lot about MATLAB, Arduino, and inverse kinematics over the course of this project. As with any project, it's never truly finished, and if we had more time to work on it, we would implement or polish the following aspects of the arm:

1. Augment the code to move the end-effector across the board.
2. Fine-tune the discretization between each letter vector point and remove redundant vectors created in the process of using LinSpace so the letters don't have horizontal lines and instead only move directly from the current point to the next.
3. Clean up unnecessary redundancy in the MATLAB code and find functions that are faster or equivalent to current processes.