**ENSC-488: Introduction to Robotics**

**Simon Fraser University**

**Spring 2017**

**Final Project Report**

Group #7

Member 1: Andrew Nichol, 301178942,[amnichol@sfu.ca](mailto:amnichol@sfu.ca)

Member 2: Adrian Fettes, 301201391, [adfettes@sfu.ca](mailto:adfettes@sfu.ca)

Member 3: Monica Li, 301226320, [xmli@sfu.ca](mailto:xmli@sfu.ca)

Contents

[Part one: 3](#_Toc479545557)

[1. Frame assignment 3](#_Toc479545558)

[2. Homogenous Matrix Transformations 4](#_Toc479545559)

[3. Position and Orientation of Tool Frame 5](#_Toc479545560)

[4. Possible inverse kinematic solutions 5](#_Toc479545561)

[5. Method of position choice 5](#_Toc479545562)

[Part two: 6](#_Toc479545563)

[1. Description of trajectory planner 6](#_Toc479545564)

[2. Cubic spline interpolation coefficients 6](#_Toc479545565)

[Part three: 8](#_Toc479545566)

[1. Dynamic simulator and calculations 8](#_Toc479545567)

[2. Control system and calculations 10](#_Toc479545568)

[3. Overall system architecture 10](#_Toc479545569)

[Plots: 11](#_Toc479545570)

[1. Part 2 11](#_Toc479545571)

[2. Part 3 12](#_Toc479545572)

[Observations: 13](#_Toc479545573)

[Team Contributions: 14](#_Toc479545574)

[1. Andrew Nichol 14](#_Toc479545575)

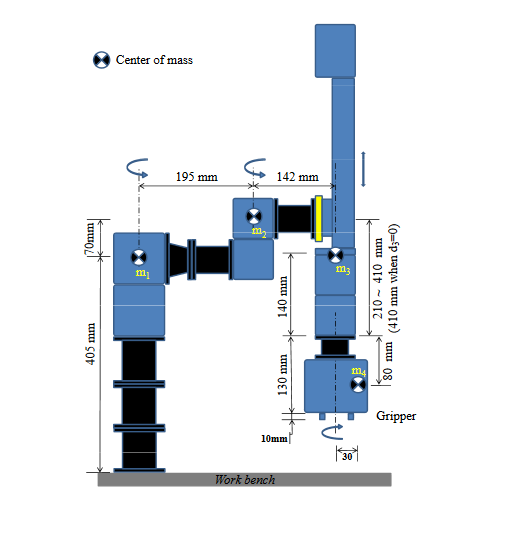
[2. Adrian Fettes 14](#_Toc479545576)

[3. Monica Li 14](#_Toc479545577)

[4. Methods of coordination: 14](#_Toc479545578)

# Part one:

## Frame assignment



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | alphai-1 | ai-1 | di | thetai |
| 1 | 0 | 0 | L1 | θ1 |
| 2 | 0 | L3 | L2 | θ2 |
| 3 | 0 | L4 | -(Lmax+d3-L5) | 0 |
| 4 | 180 | 0 | L5+L6 | θ4 |
| 5 | 0 | 0 | (L7-L6+L8/2) | 0 |

|  |  |
| --- | --- |
| Label | Length(mm) |
| L1 | 405 |
| L2 | 70 |
| L3 | 195 |
| L4 | 142 |
| L5 | 140 |
| L6 | 80 |
| L7 | 130 |
| L8 | 10 |
| L9 | 30 |
| Lmax | 410 |

{5}

{4}

{3}

{2}

{0}

{1}

## Homogenous Matrix Transformations

## Position and Orientation of Tool Frame

, φ =

## Possible inverse kinematic solutions

Step 1: There are two results for

Step 2: have 2 possible solutions because of

Step 3: d3 has one solution

## Method of position choice

To find the solution that is closest to our current position:

D3 will have to move to the same position regardless of solution, it only has one solution.

Find the combined distance each revolute joint will have to move

For joints 1, 2, and 4. We choose the final position that minimizes this distance.

# Part two:

## Description of trajectory planner

Our trajectory planner was a joint space trajectory planner, not cartesian. It worked by finding the positions of each joint at the via points via inverse kinematics, and moving to those.

The interpolation method we used was clamped cubic spline, with velocity and acceleration at the beginning and the end each set to 0.

Because it is a cubic spline, we guarantee smoothness of both position and velocity, as well as continuous position, velocity, and acceleration

Our method of cubic spline coefficient calculation is shown below

## Cubic spline interpolation coefficients

For velocity at via points:

-Average of the slope of 2 linear segments

-There are 4 constraints at each via point

For V0 and Vf at each point:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Points | Start | 1 | 2 | 3 | Final |
|  | 0 |  |  |  |  |
|  | 0 |  |  |  |  |
|  | 0 |  |  |  |  |
|  | 0 |  |  |  |  |

We scale tf (∆T) to 1 =>

At each sample time tx

After calculating these spline coefficients, we simply follow the calculated trajectory with a sampling rate of 10ms

# Part three:

## 1. Dynamic simulator and calculations

In order to derive the force-acceleration and the torque-angular acceleration formulas, we used the outwards and inwards Newton-Euler equations as detailed in equations 6.45-6.53 of the textbook. In order to account for the prismatic joint, we also added a term containing the linear velocity and acceleration into equation 6.47.

We used MATLAB to multiply our matrices and vectors together in order to form our set of formulae. It takes as inputs the rotation matrices for each frame, the current orientations and velocities, the desired accelerations, the masses of each joint, and the moments of inertia of each joint. The torque polynomials below were found using this MATLAB program.

**(1) Torque polynomials**

**τ1** = M4\*l9\*(-sin(theta4)\*(l4\*(thetadot1\*thetadot1)+l4\*(thetadot2\*thetadot2)-l3\*thetadotdot1\*sin(theta2)+l3\*(thetadot1\*thetadot1)\*cos(theta2)+l4\*thetadot1\*thetadot2\*2.0)+cos(theta4)\*(l4\*thetadotdot1+l4\*thetadotdot2+l3\*thetadotdot1\*cos(theta2)+l3\*(thetadot1\*thetadot1)\*sin(theta2))+l9\*(thetadotdot1+thetadotdot2-thetadotdot4))+M4\*(l9\*l9)\*(thetadotdot1+thetadotdot2-thetadotdot4)+M3\*l4\*(l4\*thetadotdot1+l4\*thetadotdot2+l3\*thetadotdot1\*cos(theta2)+l3\*(thetadot1\*thetadot1)\*sin(theta2))+M2\*(l3\*l3)\*thetadotdot1;

**τ2** = M3\*l4\*(l4\*(thetadotdot1+thetadotdot2)+l3\*thetadotdot1\*cos(theta2)+ l3\*(thetadot1\*thetadot1)\*sin(theta2))+M4\*l9\*(cos(theta4)\*(l4\*(thetadotdot1+thetadotdot2)+l3\*thetadotdot1\*cos(theta2)+l3\*(thetadot1\*thetadot1)\*sin(theta2))-sin(theta4)\*(l4\*pow(thetadot1+thetadot2,2.0)-l3\*thetadotdot1\*sin(theta2)+l3\*(thetadot1\*thetadot1)\*cos(theta2))+l9\*(thetadotdot1+thetadotdot2-thetadotdot4))+M4\*(l9\*l9)\*(thetadotdot1+thetadotdot2-thetadotdot4);

**f3** = (Ddotdot3-gravity)\*(M3+M4);

**τ4** =-M4\*l9\*(cos(theta4)\*(l4\*(thetadotdot1+thetadotdot2)+l3\*thetadotdot1\*cos(theta2)+ l3\*(thetadot1\*thetadot1)\*sin(theta2))-sin(theta4)\*(l4\*pow(thetadot1+thetadot2,2.0)-l3\*thetadotdot1\*sin(theta2)+l3\*(thetadot1\*thetadot1)\*cos(theta2))+l9\*(thetadotdot1+thetadotdot2-thetadotdot4))-M4\*(l9\*l9)\*(thetadotdot1+thetadotdot2-thetadotdot4);

**(2) Mass matrix**

Once we found the torque and force polynomials, we could find the closed form inverse dynamics solutions for the acceleration, to calculate the acceleration due to torque at each joint. The Mass matrixes, G vector, and V vector found via MATLAB are shown below.

**M[0][0]** = M2\*(l3\*l3)+M4\*(l9\*l9)+M4\*l9\*(l9+cos(theta4)\*(l4+l3\*cos(theta2))+ l3\*sin(theta2)\*sin(theta4))+M3\*l4\*(l4+l3\*cos(theta2))

**M[0][1]** =M3\*(l4\*l4)+M4\*(l9\*l9)+M4\*l9\*(l9+l4\*cos(theta4))

**M[0][2]** = 0

**M[0][3]** = -M4\*(l9\*l9)\*2.0

**M[1][0]** = M4\*(l9\*l9)+M4\*l9\*(l9+cos(theta4)\*(l4+l3\*cos(theta2))+l3\*sin(theta2)\*sin(theta4))+ M3\*l4\*(l4+l3\*cos(theta2))

**M[1][1]** = M3\*(l4\*l4)+M4\*(l9\*l9)+M4\*l9\*(l9+l4\*cos(theta4))

**M[1][2]** = 0

**M[1][3]** = -M4\*(l9\*l9)\*2.0

**M[2][0]** = 0

**M[2][1]** = 0

**M[2][2]** = M3+M4

**M[2][3]** = 0

**M[3][0]** = -(M4\*(l9\*l9)+M4\*l9\*(l9+cos(theta4)\*(l4+l3\*cos(theta2))+l3\*sin(theta2)\*sin(theta4)))

**M[3][1]** = -(M4\*(l9\*l9)+M4\*l9\*(l9+l4\*cos(theta4)))

**M[3][2]** = 0

**M[3][3]** = M4\*(l9\*l9) \*2.0

**(3) G vector**

**G[0]** = 0

**G[1]** = 0

**G[2]** = -gravity\*(M3+M4)

**G[3]** = 0

**(4) V vector**

**V[0]** = -M4\*l9\*(sin(theta4)\*(l4\*(thetadot1\*thetadot1)+l4\*(thetadot2\*thetadot2)+ l3\*(thetadot1\*thetadot1)\*cos(theta2)+l4\*thetadot1\*thetadot2\*2.0)-l3\*(thetadot1\*thetadot1)\*cos(theta4)\*sin(theta2))+M3\*l3\*l4\*(thetadot1\*thetadot1)\*sin(theta2)

**V[1]** = -M4\*l9\*(sin(theta4)\*(l4\*pow(thetadot1+thetadot2,2.0)+ l3\*(thetadot1\*thetadot1)\*cos(theta2))-l3\*(thetadot1\*thetadot1)\*cos(theta4)\*sin(theta2))+ M3\*l3\*l4\*(thetadot1\*thetadot1)\*sin(theta2)

**V[2]** = 0

**V[3]** = M4\*l9\*(sin(theta4)\*(l4\*pow(thetadot1+thetadot2,2.0)+ l3\*(thetadot1\*thetadot1)\*cos(theta2))-l3\*(thetadot1\*thetadot1)\*cos(theta4)\*sin(theta2))

**(5) Calculating the real-time acceleration, velocity and position**

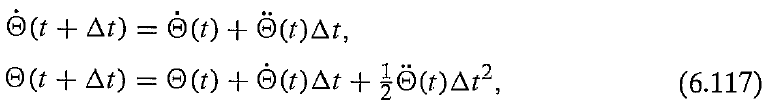
**Acceleration:**



**Initial Conditions:**



**Velocity and position:**



## 2. Control system and calculations

For the control system, we used the following design:

Kp

kv

M()

SIMULATOR

Figure Rough Control System Diagram

It was designed to follow the partitioned trajectory model as described in the class notes. The Kp values were given to be: 174, 110, 40, and 20 for joints 1 to 4 respectively. To have the system be critically damped we required that the kv be equal to 2\*sqrt(kp). Calculating this we got 26.5, 10.5, 12.65, and 9 for joints 1 to 4 kv values respectively. This controller was intended to run for 10ms for each desired position entered and call the simulator every 2ms to simulate the applied torque to the robot.

## 3. Overall system architecture

Inv kin

Traj planner

Controller

Simulator

Lasts: according to user settings

Samples: 10ms

Lasts: 10ms

Samples: 2ms

Lasts: 2ms

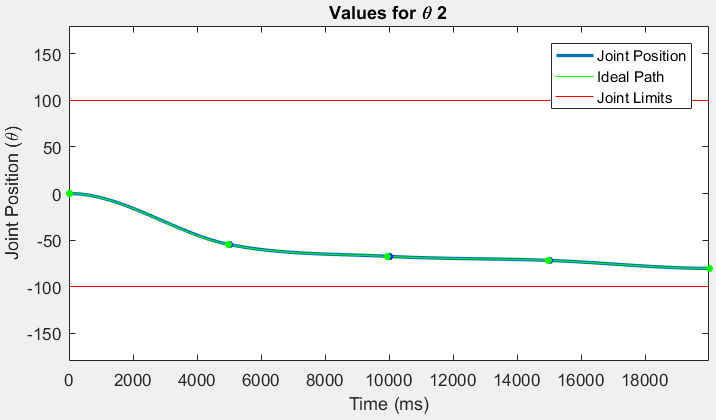
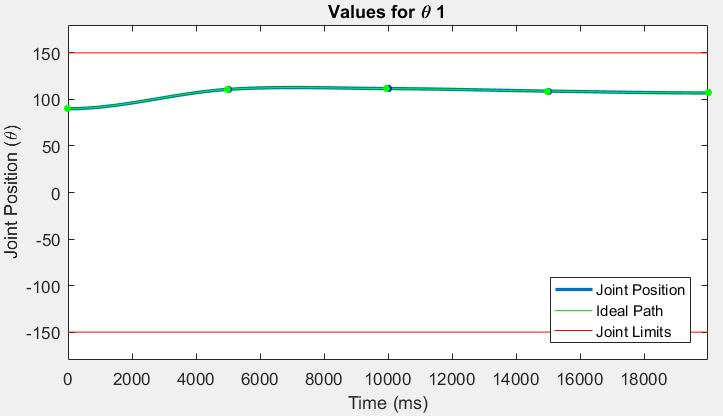
Samples: 0.2ms

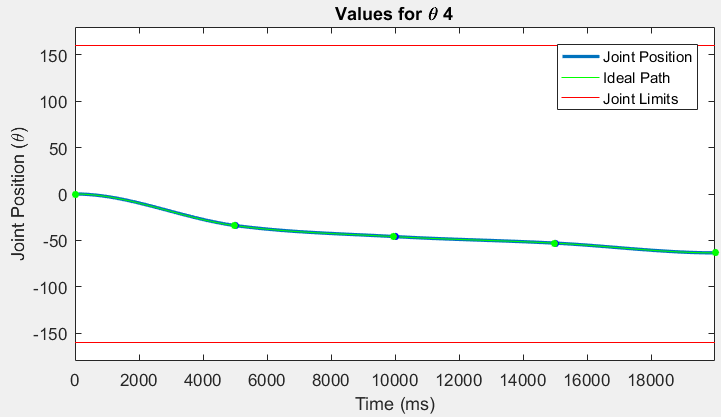
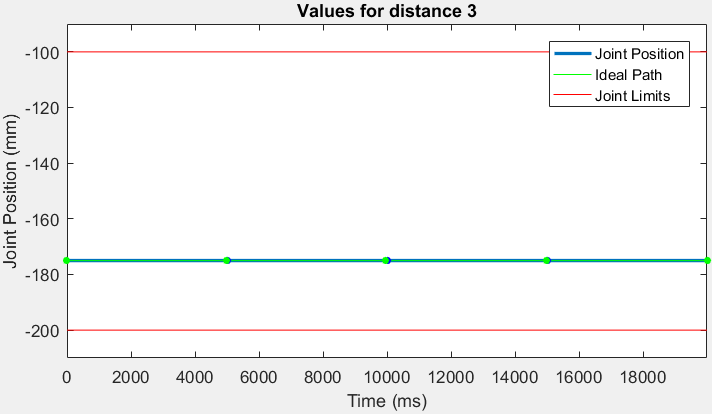
Figure General architecture diagram

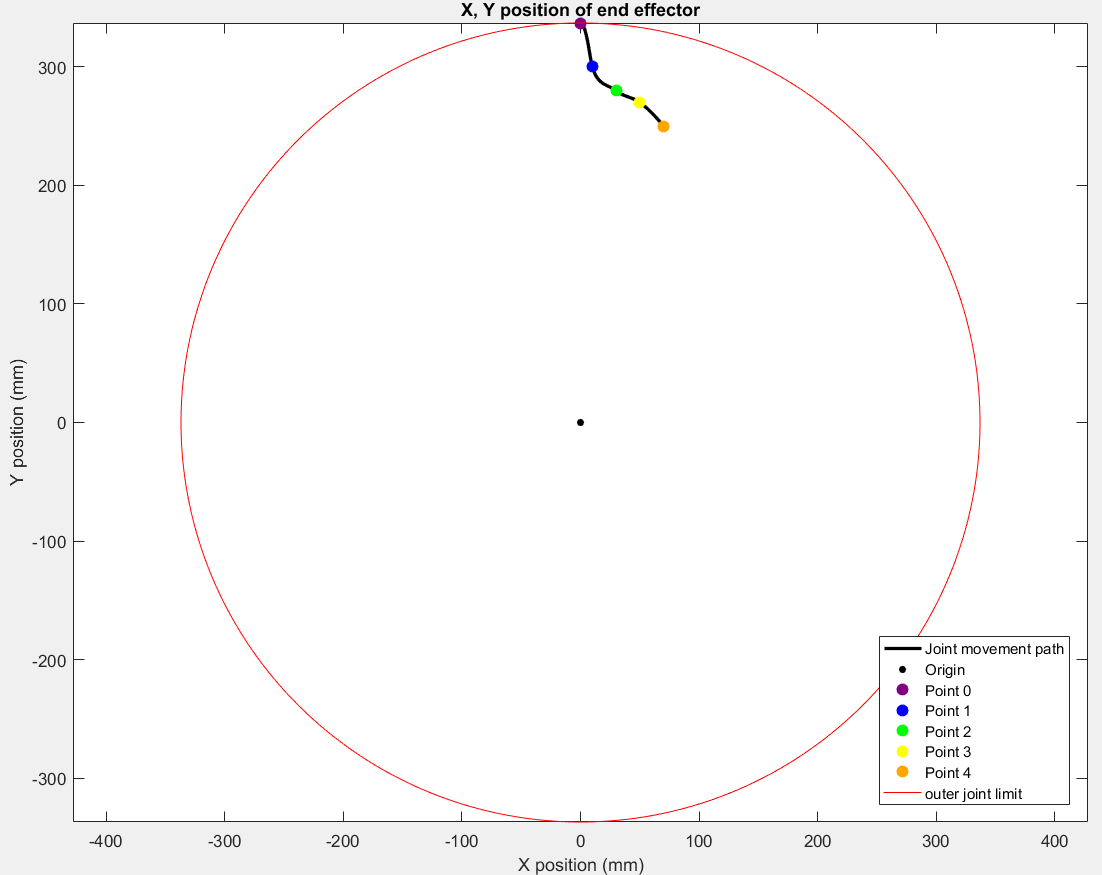
# Plots:

## Part 2

Example plots for joint movement with 4 via points

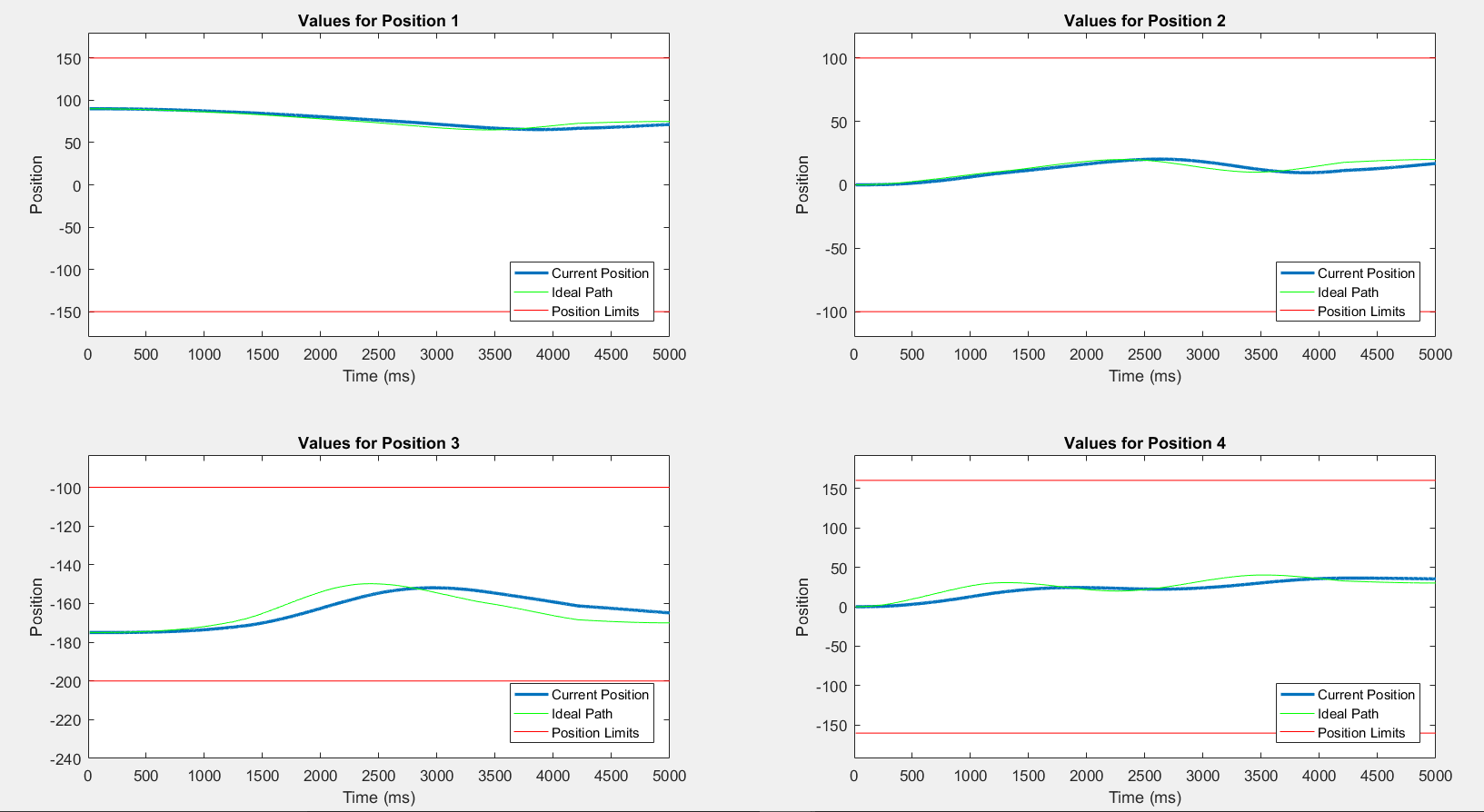


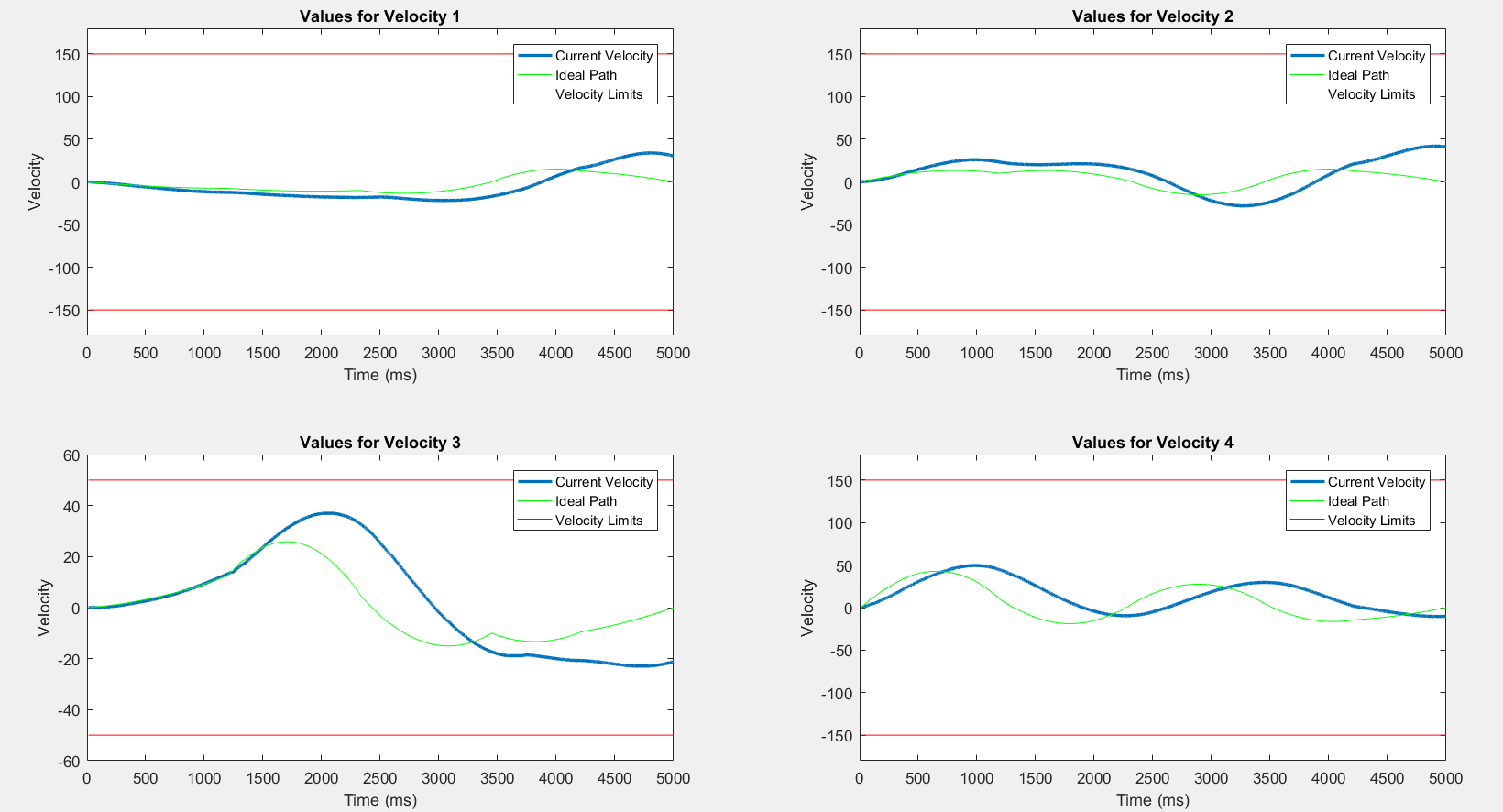


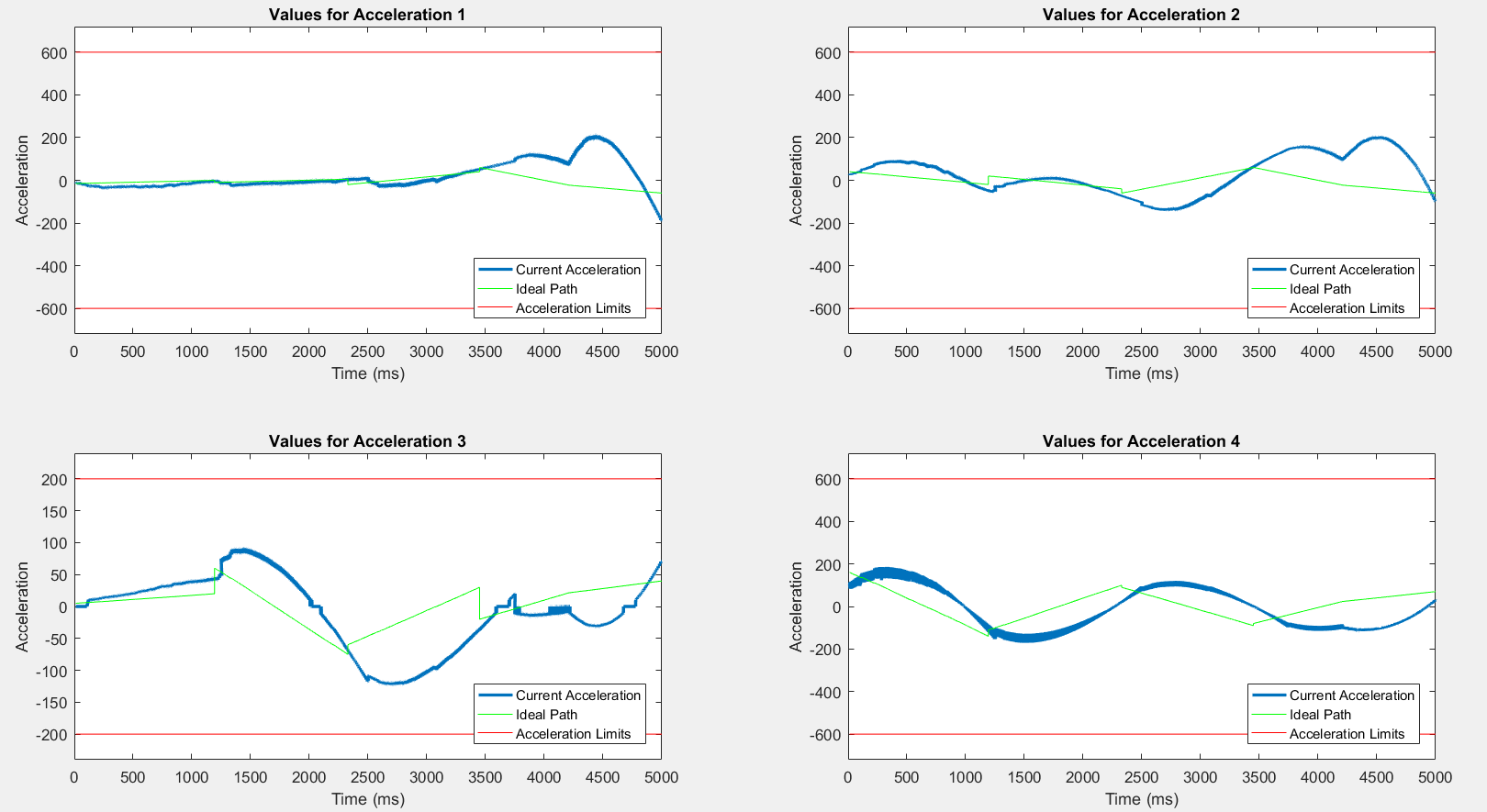


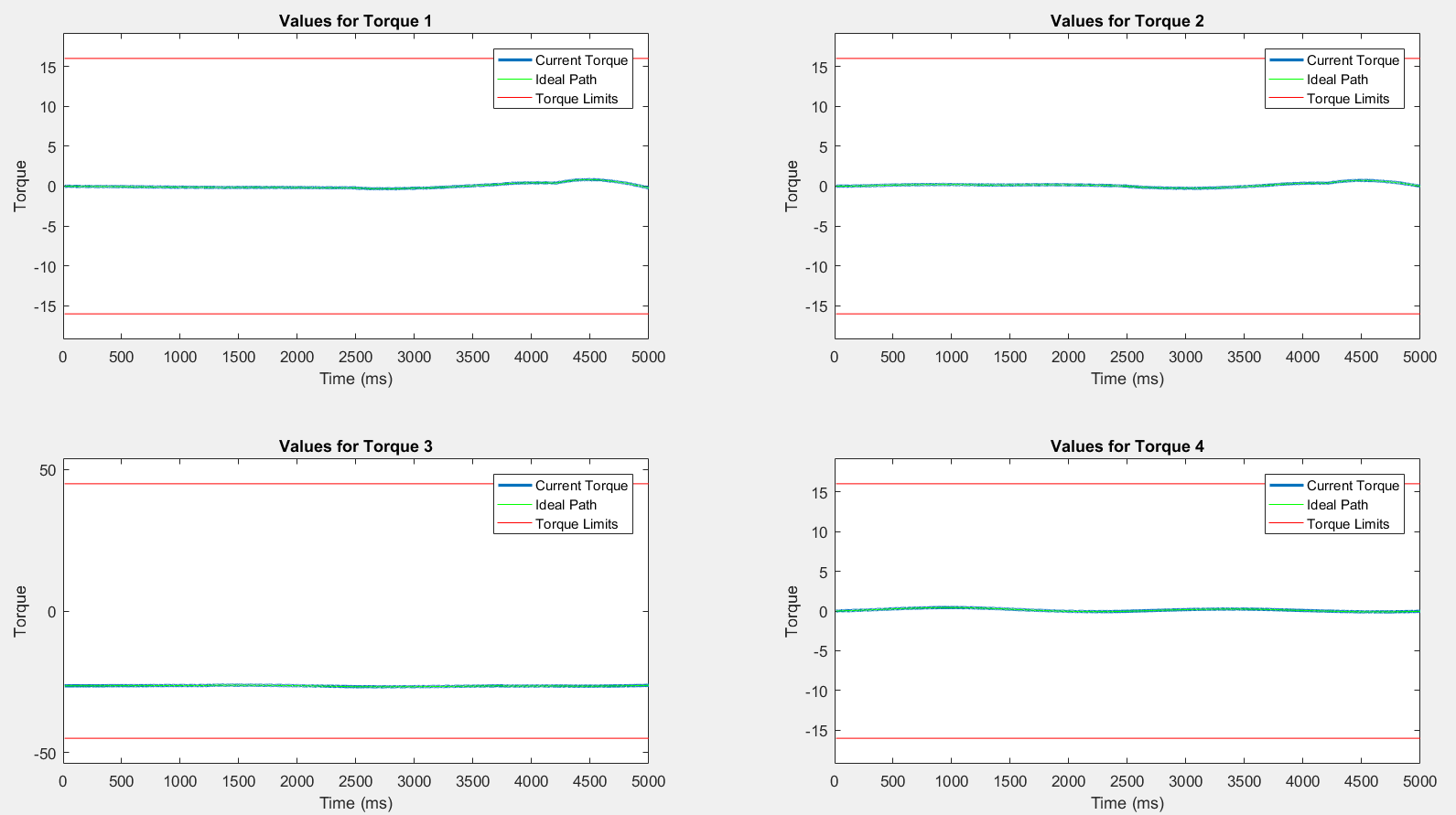
## Part 3

Example plots for joint position, velocity, acceleration, and torque are shown as follow









# Anomalous Observations:

In our final emulator, we see some lag in velocity and acceleration when compared to the optimal result. This is likely due to the real-time constraints of our emulator and controller, and our choice of kv and kp values.

# Team Contributions:

## Andrew Nichol

For the first section, Andrew worked on the demo script that added a menu to control the robot. He also worked on a library that allows for manipulation of the homogeneous matrix. On the second section Andrew developed the demo script to add new options to the control menu; he also added script to check the input from this menu. Also he wrote the underlying structure of the trajectory planner. For the third part of the project he was involved with adding more functionality to the demo script and the control menu. He also wrote a big part of the controller script and debugged the simulator script.

## Adrian Fettes

For the first section, Adrian worked on the inverse kinematics derivation and coding. Using the techniques worked on in class, he derived the inverse kinematics solutions for joints 1-4 of the robot. Then, he wrote the library function which, when given a desired cartesian position, would provide the possible joint configurations. For the second part of the project, he was involved with mainly the MATLAB portion and helping to debug the trajectory planner. He wrote the code to generate the MATLAB plots, including ideal and actual paths, and to display these graphically. He tested and helped find bugs with the cubic spline calculation and the demo. For the third part of the project, he wrote the MATLAB function to calculate the torque acceleration equations, by doing the outwards and inwards Newton-Euler equations. He also wrote the plotting functions for the path and torques output from the simulator, and helped to debug the simulator.

## Monica Li

For the first section, Monica worked on the forward kinematics derivation and coding. She also helped debug the overall test for this section. In the second section, she worked on the coefficient calculation of the cubic spline for trajectory planner. She also developed and debugged the trajectory planner on the top of Andrew’s underlying structure. She handwrote the team report for this section. In the third section, she developed dynamic simulator part. Taking the torque polynomial generated by Adrian, she used MATLAB extract the entries for mass matrix, gravity vector and V vector. She also helped debug the integrated trajectory planning code with controller and simulator.

## Methods of coordination:

To coordinate work effort, we met often on Tuesday mornings and kept in contact over the week on Facebook. For sharing work, we used GitHub. More details on code work breakdown at https://github.com/reactabean/roboticProject