**ENSC-488: Introduction to Robotics**

**Simon Fraser University Spring 2017**

**Final Project Report**

Group #7

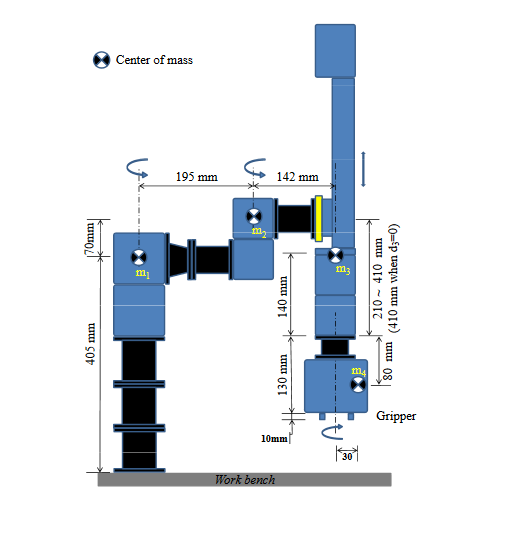
Member 1: Andrew Nichol, 301178942,[amnichol@sfu.ca](mailto:amnichol@sfu.ca)

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

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

# 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. Emulator system and calculations

## 2. Control system and calculations

For the control system we had the following design

Kp

kv

M()

EMULATOR

Figure Rough Control System Diagram

It was designed such that it followed the partitioned trajectory model as described in the class notes. The Kp values were given to us in the assignment to be the following: 174, 110, 40, and 20. This was for joints 1 to 4 respectively. Now 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. Now this controller was intended to run for 10ms for each desired position entered and it would call the emulator 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

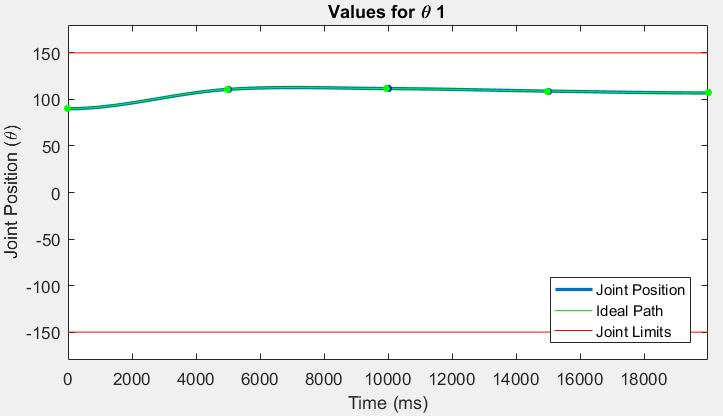
Samples: 0.2ms

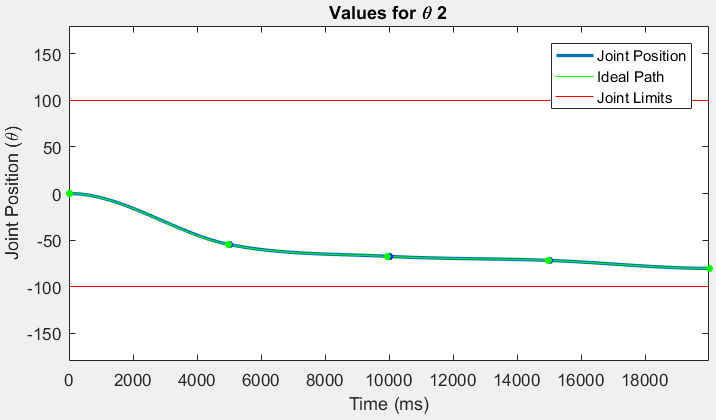
Figure General architecture diagram

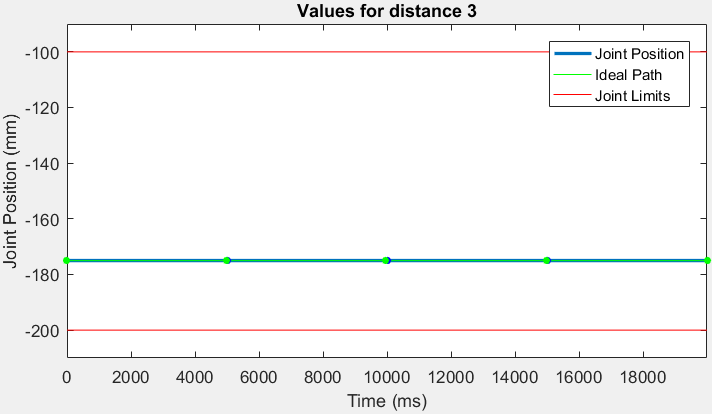
# Plots:

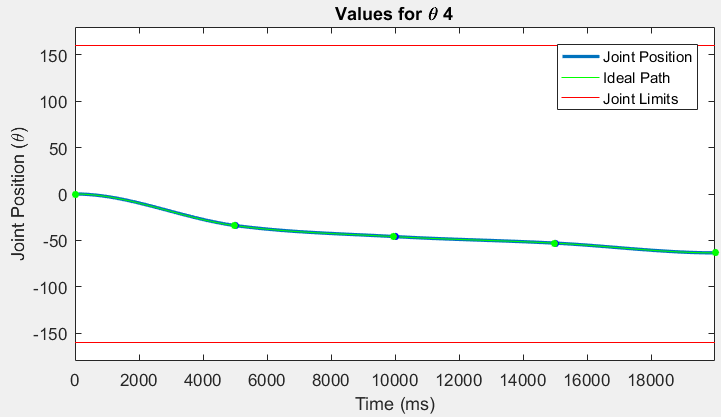
## Part 2 Plots:

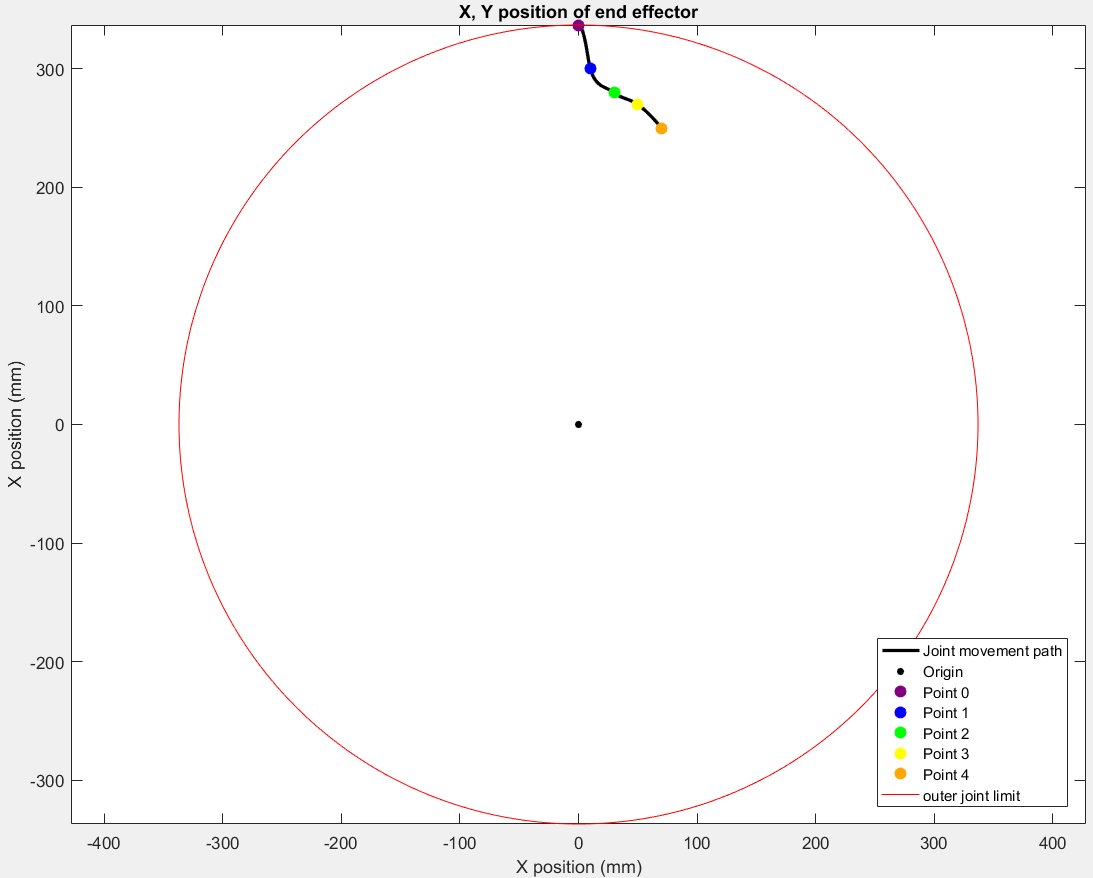
Example plots for joint movement with 4 via points











## Part 3 plots

# Observations and conclusion:

# 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 emulator 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 for the path and torques output from the emulator, and helped to debug the emulator.

## Monica Li:

## Methods of coordination:

To coordinate work effort we met often on Tuesday mornings and we kept in contact over the week on facebook. For sharing work we used github, if interested you can see more details on code work breakdown at https://github.com/reactabean/roboticProject