Development and Analysis of a Planar RRP Drawing Robotic Manipulator with PID Motion Control
Team 8

Ryan Ling
Mechanical and Aerospace Department
UCLA
Los Angeles, US
ryanling521@g.ucla.edu

Wendyl Perez
Mechanical and Aerospace Department
UCLA
Los Angeles, US
wgotos@g.ucla.edu

Shizhuo Zhu
Mechanical and Aerospace Department
UCLA
Los Angeles, US
zhushizhuo98@g.ucla.edu

***Abstract*— A Planar RRP drawing robotic manipulator is engineered to study the effects of decentralized motion control on its performance. The robot drew the 'Cool S' with and without a PID controller to demonstrate its functionality. Without a PID controller, the maximum joint angle error was 50 degrees. With the tuned PID controller, the maximum joint angle error was 3 degrees. From our analysis, we concluded that implementing a feedback control loop significantly improved the performance of the Superbot.**

***Keywords*— PID Control, Decentralized Motion Control, Robotic Manipulator, Joint Space, Position Feedback**

I. Introduction

PID control is a powerful tool that corrects parameter errors in real-time. This allows for the robotic manipulator to be responsive to its environment and have more flexibility in the tasks it can perform. Using sensors, we are able to obtain information about position, velocity, and acceleration.

Our project focused on creating a planar RRP drawing robotic manipulator using PID motion control. There were 5 main parts to this project:

1) Mechanical
2) Kinematics
3) Dynamics
4) Controls
5) Analysis

In this paper, we will demonstrate the functionality of our robotic manipulator, which we will call the Superbot, by drawing a Cool S. In Section II, we discuss the inspiration for this project. Then, we outline the mechanical aspect of our project in Section III. In Section IV, we derive the kinematics of our robotic arm. Next, we evaluate the dynamics of our system in Section V. In Section VI, we explain our controller. After, we analyze our results in Section VII. Lastly, we explore future works and the drawbacks of our project in Section VIII.

II. Related Works

Drawing has a large place in robotics, and most of the existing systems cannot create detailed figures. However, a robotic system created by Adamik et al. [1] demonstrates realistic pencil drawings based on genetic algorithms.

In the first step, a template image is converted to a greyscale bitmap. Then, they started to generate the image on a blank canvas on which they drew straight greyscale line segments of variable lengths, keeping in mind the darkness of certain parts. The robotic system was built on ABB's IRB120-3/0.6 robotic platform, which is a 6 DOF robot. Its end effector is graphite.

Inspired by this paper, our team decided to create a similar but simpler drawing robotic system, which will be described in Section III.

III. Mechanical

A. Design

Our team aimed to develop a planar RRP robotic arm capable of drawing on a 2D plane. We iterated over several designs as shown below.

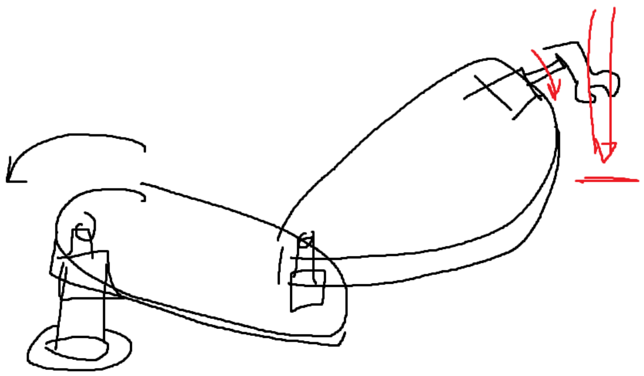


Figure 1: First version of the robot.

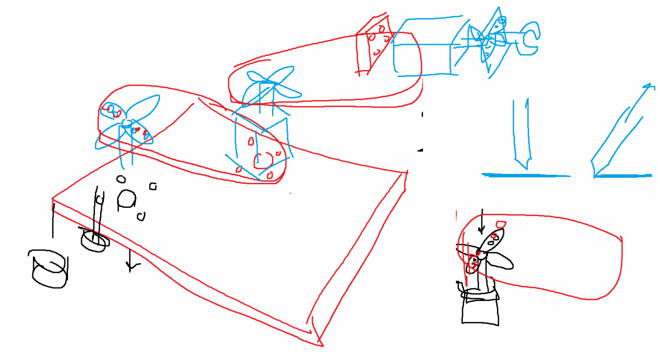


Figure 2: Second version of the robot.

In the end, we slightly modified the prismatic joint in Figure Y and modeled it in SolidWorks.

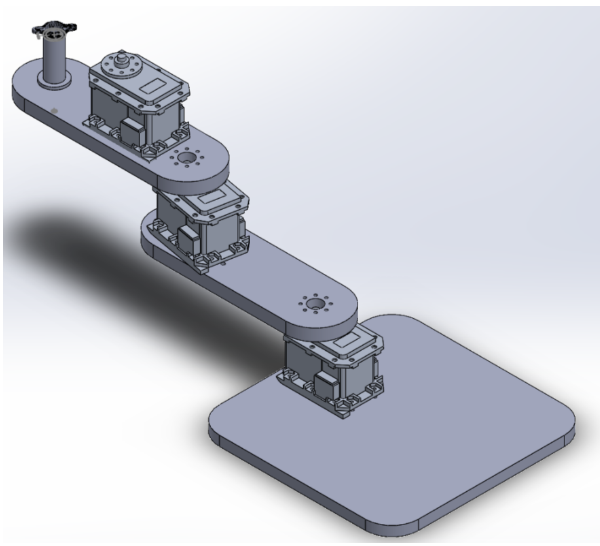


Figure 3: CAD model of the final version of the robot.

The arm consists of two linkages and a base plate. A motor is attached to the base, the end of the first link, and the beginning of the last link. Also, the prismatic joint is a screw with a slot to hold the pen. The sleeve is pushed through a screw hole, rotating the sleeve as it slides up and down the hole.

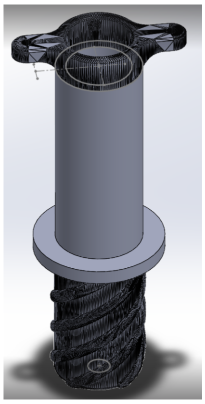


Figure 4: Pen sleeve prismatic joint.

The prismatic joint is controlled by hooking a paper clip to the horn of the motor and the pen sleeve. As the motor pulls the paper clip back, the sleeve is also pulled back, resulting in the sleeve sliding up and lifting the pen. To draw, the motor pushes the paper clip forward, causing the sleeve to rotate in the opposite direction and pushing the pen downwards onto the paper.

B. Manufacturing

All the parts except the motors, paperclip, and screws were 3D printed in PLA with 20% infill and 0.2 mm layer height. Three Dynamixel MX-28AR motors were used to move the system.

We faced tolerancing issues with the 3D printers we used in the UCLA Makerspace. The linkages and base would be several millimeters higher than expected, but this was easily solved by purchasing longer screws to attach our motors.

IV. Kinematics

A. Forward Kinematics

In our project, we chose a planar RRP as the manipulator. The standard DH parameter is used to derive the kinematic model for the purpose of controller design and verification.

Link i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	a_1	d_2	θ_2
3	0	a_2	d_3	0

Table 1: DH parameter

Our links were 3.82 inches each, so $a_1 = a_2 = 3.82$ in and $d_2 = 3$ in. The origins of frames zero and one overlap with each other. The forward kinematic model can be derived from

$$T_3^0 = T_1^0 \cdot T_2^1 \cdot T_3^2$$

Thus, the homogeneous transformation matrix from the base to the end effector is

$$T_3^0 = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 & a2*\cos(\theta_1+\theta_2)+a1*\cos\theta \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & a2*\sin(\theta_1+\theta_2)+a1*\sin\theta \\ 0 & 0 & 1 & d_2+d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

B. Inverse Kinematics

We can derive the inverse kinematics by equating a goal general homogeneous transformation matrix

$$T_3^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

to the homogeneous transformation matrix we derived above. The joint angles can be determined by equating specific matrix elements.

$$d_3 = P_z - d_2$$

$$cos\theta_2 = (P_x^2 + P_y^2 - a_1^2 - a_2^2)/(2*a_1*a_2)$$

$$sin\theta_2 = \sqrt{1 - cos\theta_2{}^2}$$

$$\theta_2 = atan2(sin\theta_2, cos\theta_2)$$

$$\theta_1 = atan2(r_{21}, r_{11}) - \theta_2$$

$$d_3 = P_z - d_2$$

The manipulator inverse kinematics is solved to transform our motion requirements from the operational space into joint space.

V. Dynamics

A. Dynamic Model

One goal of our robot is to follow the given trajectory given the coordinates of our points. This allows the generation of the reference inputs to the motion control system[3]. The dynamic model of the RRP manipulator is

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}_v\dot{\boldsymbol{q}} + \boldsymbol{F}_s\,\mathbf{sgn}\,(\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} - \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{h}_e$$

where matrices $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{g}$ represent inertia matrix, Coriolis matrix, and gravity matrix, respectively. $\boldsymbol{F_v}$ is a matrix of viscous friction coefficients, $\boldsymbol{F_s}$ ***sgn*** denotes the Coulomb friction torque, $\boldsymbol{\tau}$ is the actuation torque matrix, $\boldsymbol{J}$ is the geometric Jacobian matrix, and ***he*** denotes the vector of force and moment exerted by the end effector on the environment[6].

B. Trajectory Generation

To demonstrate our drawing robot, we wanted to draw a Cool S. First, we drew it out on a piece of paper and discretized it as shown below.

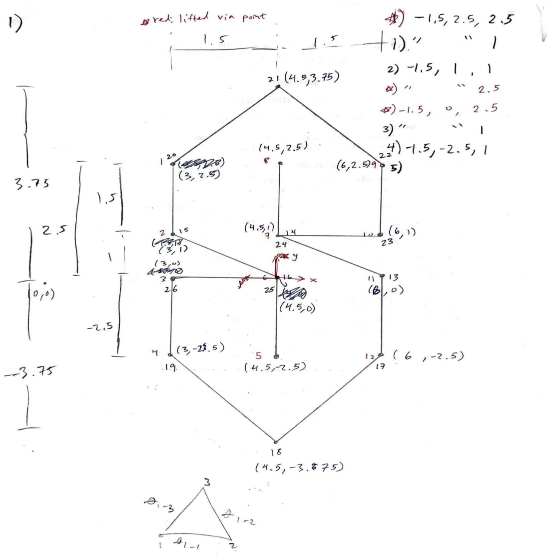


Figure 5: Coordinates of the Cool S.

After getting the coordinates and homogeneous transformation matrices of our Cool S, we fed them into our forward kinematics and inverse kinematics. Then, we interpolated between the points in task space. We used Peter Corke's Robotic Toolbox in Matlab to simulate our trajectory. Looking at just the trajectory on the paper,

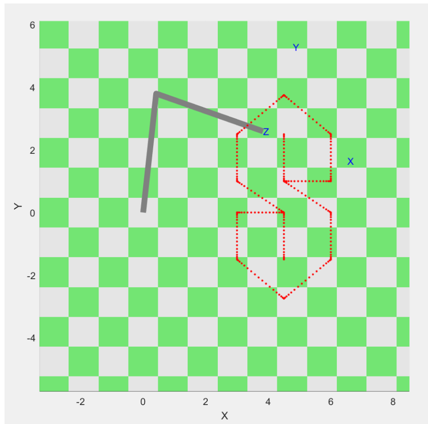


Figure 6: Simulation of drawing the Cool S.

VI. Controls

A. Objectives of Controller Design

Without a controller, each of the three Dynamixel motors of the Superbot moves unregulated with their error propagating into the end-effector space. As a result, the real-time trajectory of the Superbot will be insufficient in creating a well-defined and repeatable trajectory that matches the desired trajectory. The objective of our controller design is to mitigate the errors of each of the motors to achieve a fast, accurate, and precise drawing.

B. Assumptions

We made a few assumptions for our project:

a. All the point coordinates are already known and given to the robot, and we don't include image processing this time.

b. The surface where we place our robot is flat.

c. Smooth writing is implemented and we can neglect the force on the robot from to the forces at the pen tip.

C. Controller Selection

In selecting the controller for Superbot, a couple of assumptions were considered. Since an accurate trajectory was desirable, a controller involving position tracking was of high priority. Ideally, a tracking error of zero is desirable. In addition to the tracking, it was assumed that the robot will move and interact with the environment at a slow rate. Hence, the coupling between the motors and the joints that are heavily weighted in fast, dynamic responses can be neglected in the controller selection[2]. Based on the flow chart depicted in Figure (7), a PID position feedback from decentralized control fits the objective of the controller design.

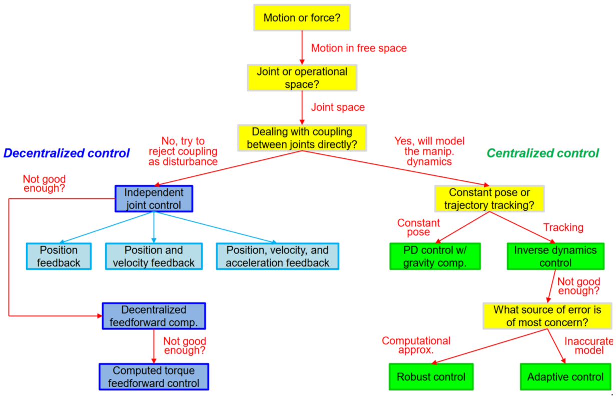


Figure 7: Controller flowchart.

Our control strategy is to regard the manipulator as formed by 3 independent systems and each joint as a single-input/single-output system[5].

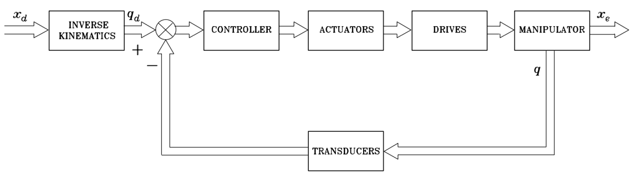


Figure 8: Block diagram of SISO system.

We have solved the inverse kinematics of the manipulator for the transformation from operational space into joint space[2]. Our next step is to design a joint space control scheme that allows the actual motion to track the reference inputs. For the position feedback,

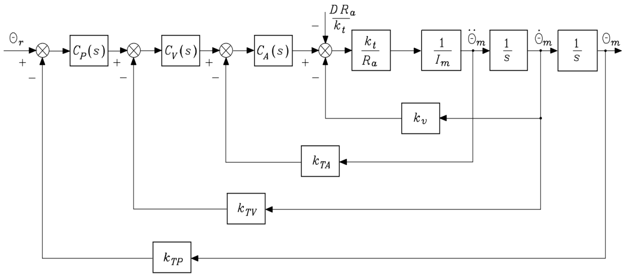


Figure 9: Independent Joint Control

The control action is characterized by

$$C_P(s) = K_P \frac{1 + sT_P}{s} \qquad C_V(s) = 1 \qquad C_A(s) = 1$$

$$k_{TV} = k_{TA} = 0$$

The closed-loop input/output transfer function is

$$P(s) = \frac{k_m K_P (1 + sT_P)}{s^2(1 + sT_m)}$$

The closed-loop disturbance/output transfer function is

$$\frac{\Theta_m(s)}{D(s)} = -\frac{\dfrac{sR_a}{k_t K_P k_{TP}(1 + sT_P)}}{1 + \dfrac{s^2(1 + sT_m)}{k_m K_P k_{TP}(1 + sT_P)}}$$

In selecting the controller for Superbot, a couple of assumptions were considered. Since an accurate trajectory was desirable, a controller involving position tracking was of high priority. Ideally, a tracking error of zero is desirable. In addition to the tracking, it was assumed that the robot will move and interact with the environment at a slow rate. Hence, the coupling between the motors and the joints that are heavily weighted in fast, dynamic responses can be neglected in the controller selection[2]. Based on the flow chart depicted in Figure (7), a PID position feedback from decentralized control fits the objective of the controller design.

From the equations above, we could say that increasing Kp to reduce the effect of disturbance on the output during the transient is worthwhile[7].

D. PID Tuning

In choosing the suitable K_P, K_I, and K_D gains for the decentralized position feedback control, the following method was used:

a. Start with an arbitrary value for K_P and observe the system's response by inputting a goal position. For a fast response time, a large K_P value should be used.

b. From the system's response, a common issue with high K_P values will be an overshoot. To lessen its magnitude, a K_D value should be added.

c. In some cases, a small steady-state error may be present in the system. To eliminate it, a low-value K_I is required.

VI. Analysis

A. Workspace

We performed a Monte Carlo simulation to generate the workspace of our robotic manipulator.

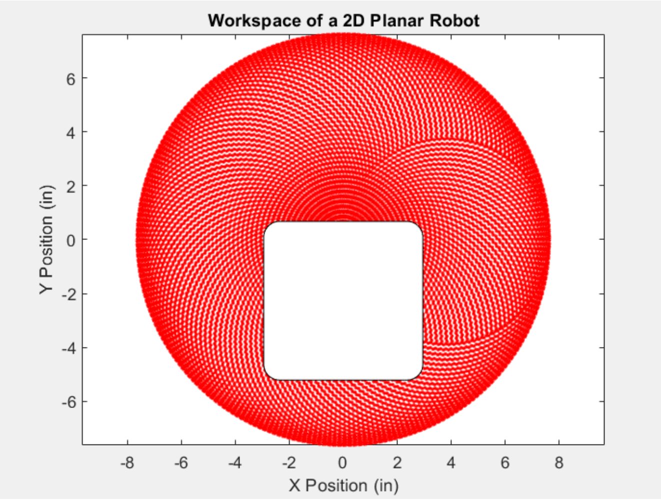


Figure 10: Workspace of the robot from Monte Carlo simulation.

Note that since we have physical constraints such as the motors and base, the workspace should have an open square in the middle. Furthermore, we technically have a 3D cylindrical workspace, but since we only care about what we can draw on a piece of paper, our 2D workspace is sufficient.

B. Singularities

Due to the mechanical interferences of our system, we avoid an elbow singularity where the last link folds into the first link[6]. To avoid the elbow-out singularity, we can generate our trajectory close to the base of the robot.

C. PID Tuning Results

As a baseline for comparison, proportional gains for the PID decentralized position control were set to zero or close to zero. The joint angle values are given in Figure (11) in the Appendix. It can be observed that the actual position value attempts to follow the goal position values; however, it lacks the response to do so.

To better demonstrate the discrepancy between the actual and goal trajectory, the joint angle errors are plotted in Figure(12) in the Appendix. The error reaches up to a magnitude of 50 degrees which significantly affects the trajectory tracking of the SuperBot.

Based on the non-controlled performance, it is evident that a PID must be implemented to reduce the error and create better trajectory tracking. After trial and error, the following controller gain values were obtained:

a. $K_P = 700$
b. $K_I = 200$
c. $K_D = 1500$

Utilizing these values, the following joint angle values are shown in Figure (13); The actual and goal position is superimposed on one another conveying that the controller is accomplishing its purpose. The corresponding error graph is depicted in Figure(14). Judging from these results, the controller values have significantly reduced the error from a magnitude of 50 degrees to almost zero degrees. There are minor oscillations away from zero degrees partly due to the fact that there are vibrations from the surroundings and from the manipulator's movement.

VII. Discussion

A. Limitations of Decentralized Position Control

There are several limitations to the controller that has been chosen. The most apparent is the inability to directly control the velocity and the acceleration of the manipulator. Only controller gain values were tuned for position. Moreover, decentralized control is not suited for fast movements as the dynamics of the motors and the linkages will start to affect the performance of the manipulator. To prevent this issue, a new control under the category of centralized control must be chosen where the dynamics of the manipulator are modeled and taken into account. To add to the issue of speed, the feedback control driving the correction is based on reaction and is insufficient. For fast movements, the system has to plan ahead; therefore, a feedforward control will be beneficial to further reduce tracking errors.

B. Challenges

a. Data Transmission: The first major hurdle was the initial read and write script's inability to transmit data to the three Dynamixel motors concurrently and efficiently. This tremendously affected the trajectory of the manipulator as it created a 'shakiness' while it was drawing. To solve the issue, a new script using sync read/write was implemented that involved packing all the data into one command which allowed for efficient transmission of data.

b. Prismatic Joint Screw Mechanism: In terms of the 3D-printed framework that allowed the manipulator to function, the screw responsible for lifting the drawing tool above the paper underperformed. It was a tolerancing issue in which it proved too loose for it to only translate up and down. It also translated from side to side which caused the pen to write unwanted lines as it retracted vertically.

c. Structural Integrity: The structure was not leveled due to the mass of each component. As the manipulator fully extends its linkages, the moment from its weight caused the linkages to bend, creating an uneven workspace that forced Superbot to draw even when its tool was fully retracted.

C. Future Work

Most, if not all, challenges that were faced during the development could be prevented through a better motor mounting system and an alternative rotation-to-linear motion mechanism. Furthermore, to fully automate the trajectory generation process, we can incorporate image processing. The automated process reduces the number of inputs the user would have to provide to SuperBot. For instance, a person with no knowledge of the mechanics of SuperBot can draw dots with a specified number associated with it to guide the robot from one dot to another.

VIII. Conclusions

The Superbot is a simple, but effective way to explore the design process of a robotic manipulator from its initial modeling steps to the tuning of its controller. It provides insight into the importance of a controller in terms of trajectory tracking. Without a controller, the Superbot would not have accomplished its goal of maneuvering its path across a piece of paper while drawing lines from one dot to another. The functionality of Superbot is not limited to only drawing the Cool S. It can also provide the opportunity to be an educational kit in its future iterations. Due to its simplicity and modular design, it can easily be built at home where it can function as a teacher for enforcing the geometry of letters and shapes!

References

[1] Adamik, M., Goga, J., Pavlovicova, J., Babinec, A., & Sekaj, I. (2021, October 21). *Fast robotic pencil drawing based on image evolution by means of genetic algorithm*. Robotics and Autonomous Systems. Retrieved June 7, 2022.

[2] E. Y. L. Gu, "Configuration manifolds and their applications to robot dynamic modeling and control," in *IEEE Transactions on Robotics and Automation*, vol. 16, no. 5, pp. 517-527, Oct. 2000.

[3] M. Petko, G. Karpiel, D. Prusak and A. Martowicz, "A new 3-DOF parallel manipulator," *Proceedings of the Fourth International Workshop on Robot Motion and Control (IEEE Cat. No.04EX891)*, 2004.

[4] J. Guo, B. Xian, F. Wang and X. Zhang, "Development of a three degree-of-freedom testbed for an unmanned helicopter and attitude control design," *Proceedings of the 32nd Chinese Control Conference*, 2013.

[5] T. F. Arciuolo and M. Faezipour, "Dynamically Adjustable PID for Adaptive Motion Control: PID Algorithm Introduction and Applications," *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2020.

[6] B. Yao and C. Jiang, "Advanced motion control: From classical PID to nonlinear adaptive robust control," *2010 11th IEEE International Workshop on Advanced Motion Control (AMC)*, 2010.

[7] Y. Kume, Y. Hirata, K. Kosuge, H. Asama, H. Kaetsu and K. Kawabata, "Decentralized control of multiple mobile robots transporting a single object in coordination without using force/torque sensors," *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, 2001.

[8] T. Mukaiyama, Kyunghwan Kim and Y. Hori, "Implementation of cooperative manipulation using decentralized robust position/force control," *Proceedings of 4th IEEE International Workshop on Advanced Motion Control - AMC '96 - MIE*, 1996.

Appendix

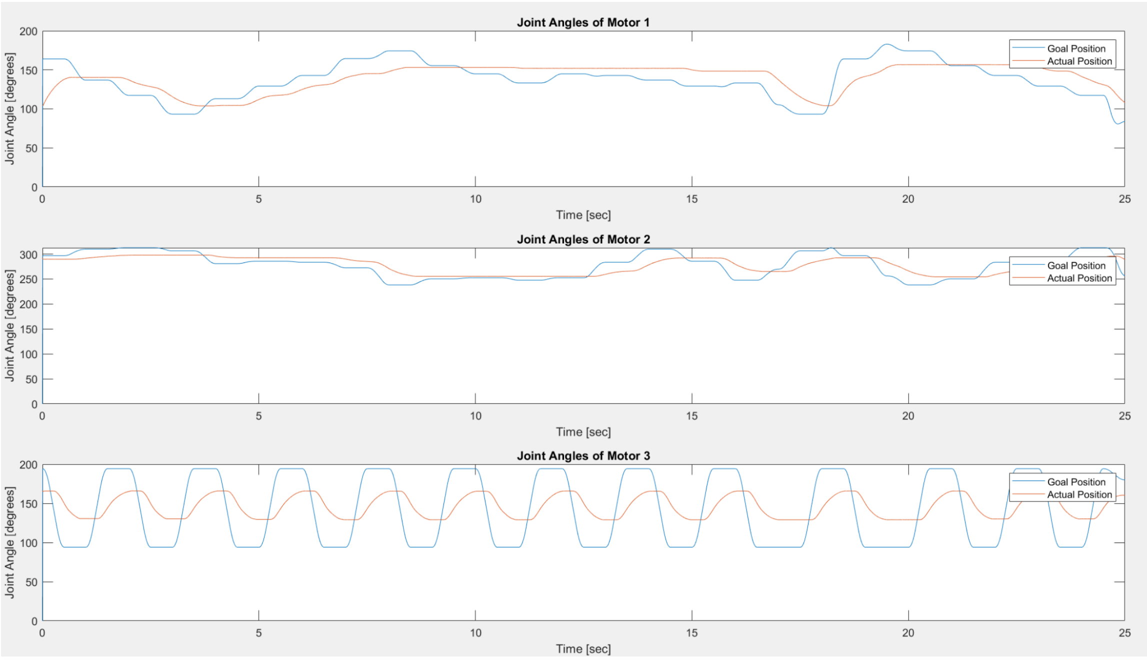


Figure 11: Actual trajectory (red) plotted against the ideal trajectory (blue) with the uncontrolled manipulator

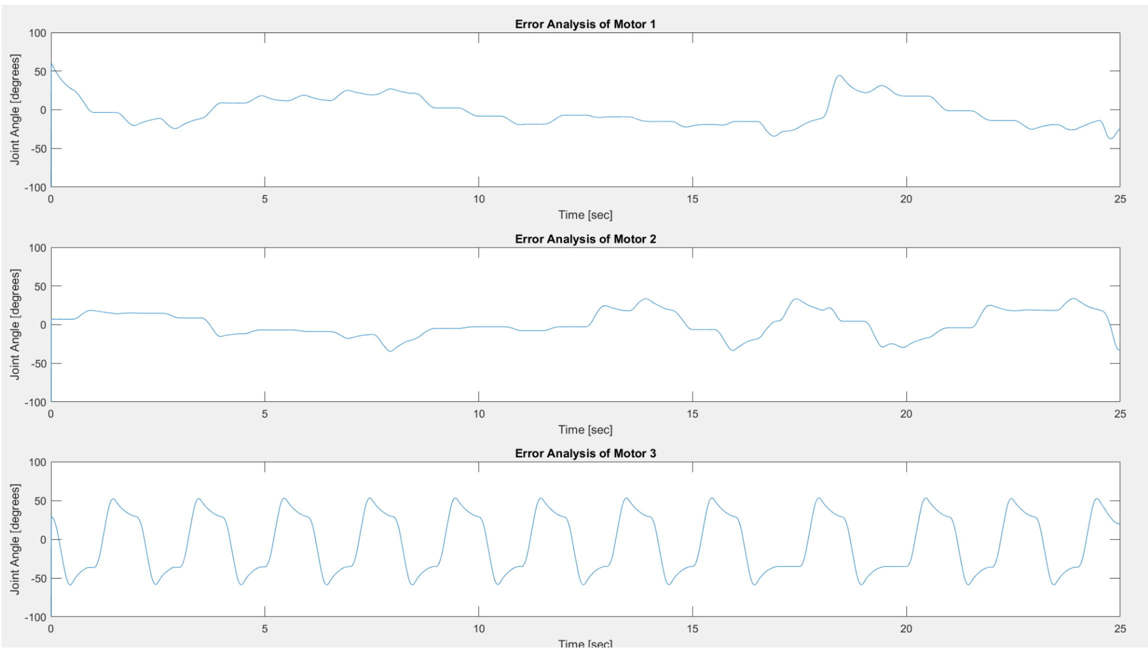


Figure 12: The error plot for each of the manipulator's motors in its uncontrolled state

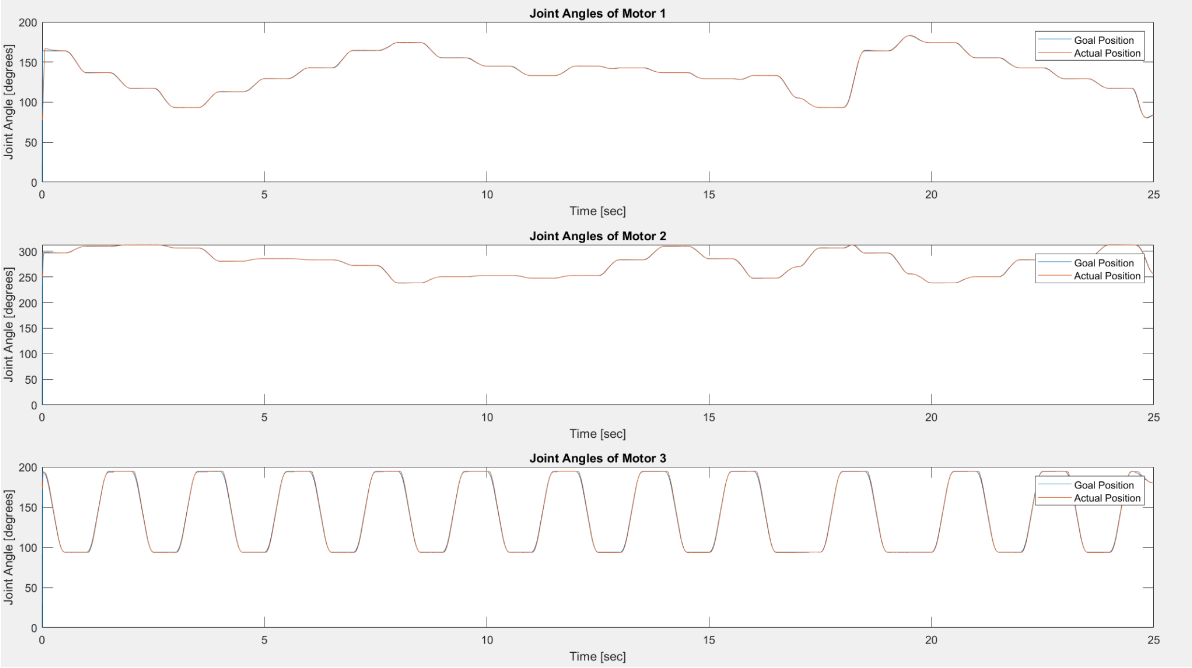


Figure 13: Actual trajectory (red) plotted against the ideal trajectory (blue) with the PID controlled manipulator

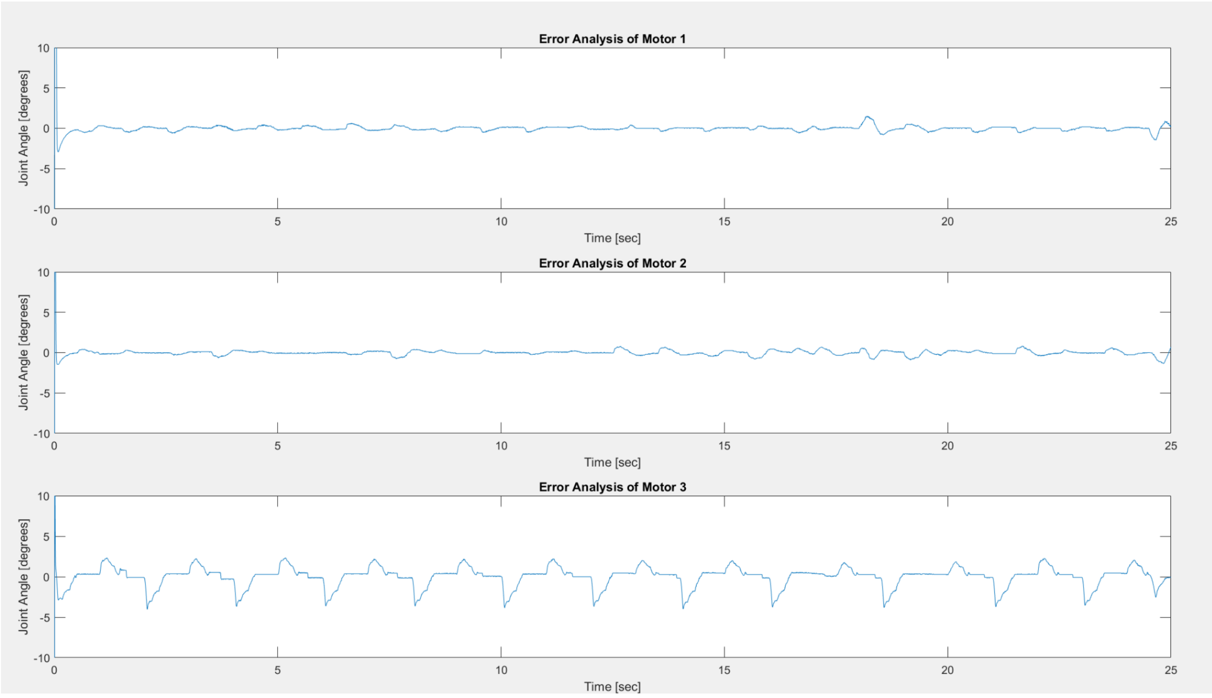


Figure 14: The error plot for each of the manipulator's motors in its tuned controller state

Contributions:

Ryan Ling	Wendyl Perez	Shizhuo Zhu
CAD Model	CAD Model	Script Development
Script Development	Script Development	Dynamics
Assembly	Assembly	Assembly
Controller Design	Controller Design	Controller Design

MATLAB Script:

Main.m

```
% MAE C163A/C263A Project
% Team X
clear;clc;
sync_read_write();
%% Close Port
closePort(port_num);
clc;
%%
h = 20;
current_time = 0;
tic
syms a1 a2 d2 t1 t2 d3
% need to change paramters such as the height
a1 = 3.82; a2 = 3.621; d2 = 3;
L1 = Link('revolute','d', 0, 'a', 0,'alpha', 0, 'modified');
L2 = Link('revolute','d', d2, 'a', a1,'alpha', 0, 'modified');
L3 = Link('prismatic','theta', 0, 'a', a2, 'alpha', 0, 'modified');
% make the robot
tool = transl(0, 0, -3);
Ryan = SerialLink([L1 L2 L3], 'name', 'Ryan', 'tool', tool);
% via point for each point (p is on paper, v is above)
height = 3.25;
down = 1.5;
v1 = traj_pt(3,2.5,height);
p1 = traj_pt(3,2.5,down);
p2 = traj_pt(3,1,down);
v2 = traj_pt(3,1,height);
v3 = traj_pt(3,0,height);
```

```
p3 = traj_pt(3,0,down);
p4 = traj_pt(3,-1.5,down);
v4 = traj_pt(3,-1.5,height);
v5 = traj_pt(4.5,-1.5,height);
p5 = traj_pt(4.5,-1.5,down);
p6 = traj_pt(4.5,0,down);
v6 = traj_pt(4.5,0,height);
v7 = traj_pt(4.5,1,height);
p7 = traj_pt(4.5,1,down);
p8 = traj_pt(4.5,2.5,down);
v8 = traj_pt(4.5,2.5,height);
v9 = traj_pt(6,2.5,height);
p9 = traj_pt(6,2.5,down);
p10 = traj_pt(6,1,down);
v10 = traj_pt(6,1,height);
v11 = traj_pt(6,0,height);
p11 = traj_pt(6,0,down);
p12 = traj_pt(6,-1.5,down);
v12 = traj_pt(6,-1.5,height);
v13 = v11;
p13 = p11;
p14 = p7;
v14 = v7;
v15 = v2;
p15 = p2;
p16 = p6;
v16 = v6;
v17 = v12;
p17 = p12;
p18 = traj_pt(4.5,-2.75,down);
p19 = p4;
v19 = v4;
v20 = v1;
p20 = p1;
p21 = traj_pt(4.5,3.75,down);
p22 = p9;
v22 = v9;
v23 = v10;
p23 = p10;
p24 = p7;
v24 = v7;
v25 = v6;
p25 = p6;
p26 = p3;
% move away to see the paper
v26 = v3;
v27 = traj_pt(3,-5,3);
```

```
points =
cat(1,v1,p1,p2,v2,v3,p3,p4,v4,v5,p5,p6,v6,v7,p7,p8,v8,v9,p9,p10,v10,v11,p11,p12,v12,v13,p13,p14,v14,
v15,p15,p16,v16,v17,p17,p18,p19,v19,v20,p20,p21,p22,v22,v23,p23,p24,v24,v25,p25,p26,v26,v27);
% Ctraj
ii = 1;
for i = 1:4:length(points)-7
   Traj{ii} = ctraj(points(i:i+3,:),points(i+4:i+7,:),h);
   ii = ii + 1;
end
for j = 1:length(Traj)
   for i = 1:h
       temp_1 = ik_test(Traj{j}(:,:,i))*(4096/(2*pi));
       theta1(i) = temp_1(1)+2048;
       theta2(i) = temp_1(2)+2048;
       distance(i) = temp_1(3)+2048;
   end
   theta1_new{j} = theta1;
   theta2_new{j} = theta2;
   distance_new{j} = distance;
end
theta1_full = 0;
theta2_full = 0;
distance_full = 0;
Traj_full = diag([0 0 0 0]);
for i = 1:length(Traj)
   theta1_full = cat(2,theta1_full,theta1_new{i});
   theta2_full = cat(2,theta2_full,theta2_new{i});
   distance_full = cat(2,distance_full,distance_new{i});
   for k = 1:h
   Traj_full = cat(1,Traj_full,Traj{i}(:,:,k));
   end
end
theta_list = [theta1_full; theta2_full; distance_full]';
%%
qt3 = theta_list*2*pi/4096;
kk = 1;
for j = 1:length(qt3)
   Ryan.plot(qt3(j,:),'workspace',[-10 10 -10 10 1 2])
   [rot,pos] = tr2rt(Traj_full(kk:kk+3,1:4));
   plot3(pos(1),pos(2),pos(3),'r.')
   kk = kk + 4;
   hold on
end
%% Sync
   theta1_actual = zeros(1,length(theta_list));
   theta2_actual = zeros(1,length(theta_list));
   theta3_actual = zeros(1,length(theta_list));
   % Add parameter storage for Dynamixel#0 present position value
```

```
  dxl_addparam_result = groupSyncReadAddParam(groupread_num, DXL0_ID);

  % Add parameter storage for Dynamixel#1 present position value
  dxl_addparam_result = groupSyncReadAddParam(groupread_num, DXL1_ID);
  % Add parameter storage for Dynamixel#2 present position value
  dxl_addparam_result = groupSyncReadAddParam(groupread_num, DXL2_ID);
for kk = 1:length(theta_list)
  Theta1 = theta_list(kk,1);%+init_pos(1);
  Theta2 = theta_list(kk,2);%+init_pos(2);
  distance3 = theta_list(kk,3); %+init_pos(3);

  % Add Dynamixel#0 goal position value to the Syncwrite storage
  dxl_addparam_result = groupSyncWriteAddParam(groupwrite_num, DXL0_ID,
typecast(int32(Theta1), 'uint32'), LEN_PRO_GOAL_POSITION);

  % Add Dynamixel#1 goal position value to the Syncwrite storage
  dxl_addparam_result = groupSyncWriteAddParam(groupwrite_num, DXL1_ID,
typecast(int32(Theta2), 'uint32'), LEN_PRO_GOAL_POSITION);

  % Add Dynamixel#2 goal position value to the Syncwrite parameter storage
  dxl_addparam_result = groupSyncWriteAddParam(groupwrite_num, DXL2_ID,
typecast(int32(distance3), 'uint32'), LEN_PRO_GOAL_POSITION);
  % Syncwrite goal position
  groupSyncWriteTxPacket(groupwrite_num);
  dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);

  % Clear syncwrite parameter storage
  groupSyncWriteClearParam(groupwrite_num);

  % Syncread present position (COMMENT IT OUT WHEN DEMO)
  groupSyncReadTxRxPacket(groupread_num);

  dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
  dxl_getdata_result = groupSyncReadIsAvailable(groupread_num, DXL0_ID,
ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
  dxl_getdata_result = groupSyncReadIsAvailable(groupread_num, DXL1_ID,
ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
  dxl_getdata_result = groupSyncReadIsAvailable(groupread_num, DXL2_ID,
ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
  % Get Dynamixel#0 present position value
  theta1_actual(kk) = groupSyncReadGetData(groupread_num, DXL0_ID,
ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
  % Get Dynamixel#1 present position value
  theta2_actual(kk) = groupSyncReadGetData(groupread_num, DXL1_ID,
ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
  % Get Dynamixel#2 present position value
  theta3_actual(kk) = groupSyncReadGetData(groupread_num, DXL2_ID,
ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
```

```
end
%% Plots
theta1_actual_T = theta1_actual';
theta2_actual_T = theta2_actual';
theta3_actual_T = theta3_actual';
time_actual = linspace(0,25,length(theta1_actual_T));
err1 = rad2deg((theta_list(:,1) - theta1_actual_T)*((2*pi)/4096));
err2 = rad2deg((theta_list(:,2) - theta2_actual_T)*((2*pi)/4096));
err3 = rad2deg((theta_list(:,3) - theta3_actual_T)*((2*pi)/4096));
figure(1)
subplot(3,1,1)
plot(time_actual,err1)
title('Error Analysis of Motor 1')
axis([0 25 -100 100])
% axis([0 25 -10 10])
xlabel('Time [sec]'); ylabel('Joint Angle [degrees]')
subplot(3,1,2)
plot(time_actual,err2)
title('Error Analysis of Motor 2')
axis([0 25 -100 100])
% axis([0 25 -10 10])
xlabel('Time [sec]'); ylabel('Joint Angle [degrees]')
subplot(3,1,3)
plot(time_actual,err3)
axis([0 25 -100 100])
% axis([0 25 -10 10])
title('Error Analysis of Motor 3')
xlabel('Time [sec]'); ylabel('Joint Angle [degrees]')
figure(2)
subplot(3,1,1)
plot(time_actual,rad2deg((theta_list(:,1))*((2*pi)/4096)))
hold on
plot(time_actual,rad2deg((theta1_actual_T)*((2*pi)/4096)))
xlabel('Time [sec]'); ylabel('Joint Angle [degrees]')
title('Joint Angles of Motor 1')
legend('Goal Position','Actual Position')
subplot(3,1,2)
plot(time_actual,rad2deg((theta_list(:,2))*((2*pi)/4096)))
hold on
plot(time_actual,rad2deg((theta2_actual_T)*((2*pi)/4096)))
xlabel('Time [sec]'); ylabel('Joint Angle [degrees]')
title('Joint Angles of Motor 2')
legend('Goal Position','Actual Position')
subplot(3,1,3)
plot(time_actual,rad2deg((theta_list(:,3))*((2*pi)/4096)))
hold on
plot(time_actual,rad2deg((theta3_actual_T)*((2*pi)/4096)))
xlabel('Time [sec]'); ylabel('Joint Angle [degrees]')
```

```
title('Joint Angles of Motor 3')
legend('Goal Position','Actual Position')
%% Function
function A = traj_pt(x,y,z)
  A = [1 0 0 x; 0 1 0 y; 0 0 1 z;0 0 0 1];
end
function t = ik_test(T_g)
  a1 = 3.82; a2 = 3.6210; d2 = 3;
  [rot,pos] = tr2rt(T_g);
  x = pos(1); y = pos(2); z = pos(3);
  d3 = z - d2;
  q2 = acos((x^2+y^2-a1^2-a2^2)/(2*a1*a2));
  q1 = atan2(y,x) - atan2(a2*sin(q2),a1+a2*cos(q2));
  t = [q1 q2 d3];
end
function [theta1,theta2,distance] = traj_pt_gen(a1,a2,d2,h,p1,p2)
  Traj = ctraj(p1,p2,h);

  for i = 1:h
     temp = ik_test(a1,a2,d2,Traj(:,:,i))*(4096/(2*pi));
     theta1(i) = temp(1)+2048;
     theta2(i) = temp(2)+2048;
     distance(i) = temp(3)+2048;
  end
end


clc; close all;

syms a1 a2 d2 t1 t2 d3

% need to change paramters such as the height
a1 = 3.82; a2 = 3.82; d2 = 3;
L1 = Link('revolute','d', 0, 'a', 0,'alpha', 0, 'modified');
L2 = Link('revolute','d', d2, 'a', a1,'alpha', 0, 'modified');
L3 = Link('prismatic','theta', 0, 'a', a2, 'alpha', 0, 'modified');
% make the robot
tool = transl(0, 0, -3);
Ryan = SerialLink([L1 L2 L3], 'name', 'Ryan', 'tool', tool);

% th = [t1 t2 d3];
th = [pi/2 pi 2];

% forward kinematics
FK = Ryan.fkine(th);

th1 = [1 0 0 3; 0 1 0 -3; 0 0 1 2; 0 0 0 1];
th2 = [1 0 0 3; 0 1 0 -4; 0 0 1 2; 0 0 0 1];
```

```
Traj = ctraj(th1,th2,10); %same number
 for i = 1:10
   temp = Ryan.ikine(Traj(:,:,i),'mask',[1 1 1 0 0 0])*180/pi;
   theta1(i) = temp(1);
   theta2(i) = temp(2);
 end

thetas = [theta1; theta2]'

% inverse kinematics
IK = Ryan.ikine(FK,'mask',[1 1 1 0 0 0]);

%%
 for j = 1:10 %same number
   qt3(j,:) = Ryan.ikine(Traj(:,:,j),'mask',[1 1 1 0 0 0])
   Ryan.plot(qt3(j,:),'workspace',[-10 10 -10 10 -5 10]);
   [rotation, position] = tr2rt(Traj(:,:,j));
   plot3(position(1),position(2),position(3),'r*')
   hold on
 end
%% plot robot
figure();
Ryan.plot([0 0 0],'workspace',[-10 10 -10 10 -5 10]);
%% Monte Carlo Simulation
for t1 = .01:0.05:2*pi
   for t2 = .01:0.05:2*pi
      forward = Ryan.fkine([t1 t2 0]);
      [R0T, P0T] = tr2rt(forward);
      plot(P0T(1),P0T(2),'r.');
      hold on
   end
end
hold on
rectangle('Position',[-2.95 -5.22 5.9 5.9],'FaceColor','w','Curvature',0.2)
xlabel('X Position (in)')
ylabel('Y Position (in)')
title('Workspace of a 2D Planar Robot')
axis equal;
```

sync_read_write.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Copyright 2017 ROBOTIS CO., LTD.
%
% Licensed under the Apache License, Version 2.0 (the "License");
```

```
% you may not use this file except in compliance with the License.
% You may obtain a copy of the License at
%
%     http://www.apache.org/licenses/LICENSE-2.0
%
% Unless required by applicable law or agreed to in writing, software
% distributed under the License is distributed on an "AS IS" BASIS,
% WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
% See the License for the specific language governing permissions and
% limitations under the License.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author: Ryu Woon Jung (Leon)
%
% *********     Sync Read and Sync Write Example      *********
%
%
% Available Dynamixel model on this example : All models using Protocol 2.0
% This example is designed for using two Dynamixel PRO 54-200, and an USB2DYNAMIXEL.
% To use another Dynamixel model, such as X series, see their details in E-Manual(emanual.robotis.com)
and edit below variables yourself.
% Be sure that Dynamixel PRO properties are already set as %% ID : 1 / Baudnum : 1 (Baudrate : 57600)
%
clc;
clear all;
lib_name = '';
if strcmp(computer, 'PCWIN')
 lib_name = 'dxl_x86_c';
elseif strcmp(computer, 'PCWIN64')
 lib_name = 'dxl_x64_c';
elseif strcmp(computer, 'GLNX86')
 lib_name = 'libdxl_x86_c';
elseif strcmp(computer, 'GLNXA64')
 lib_name = 'libdxl_x64_c';
elseif strcmp(computer, 'MACI64')
 lib_name = 'libdxl_mac_c';
end
% Load Libraries
if ~libisloaded(lib_name)
   [notfound, warnings] = loadlibrary(lib_name, 'dynamixel_sdk.h', 'addheader', 'port_handler.h',
'addheader', 'packet_handler.h', 'addheader', 'group_sync_write.h', 'addheader', 'group_sync_read.h');
end
% Control table address
ADDR_PRO_TORQUE_ENABLE          = 64;%24;                 % Control table address is different in
Dynamixel model
ADDR_PRO_GOAL_POSITION          = 116; %30;
ADDR_PRO_PRESENT_POSITION       = 132; %36;
% Data Byte Length
```

```
LEN_PRO_GOAL_POSITION       = 4;
LEN_PRO_PRESENT_POSITION    = 4;
% Protocol version
PROTOCOL_VERSION            = 2.0;          % See which protocol version is used in the Dynamixel
% Default
DXL0_ID                     = 0;
DXL1_ID                     = 1;            % Dynamixel#1 ID: 1
DXL2_ID                     = 2;            % Dynamixel#2 ID: 2
BAUDRATE                    = 57600; %1000000; % 57600;
DEVICENAME                  = 'COM3';       % Check which port is being used on your controller
                                            % ex) Windows: 'COM1'   Linux: '/dev/ttyUSB0' Mac: '/dev/tty.usbserial-*'
TORQUE_ENABLE               = 1;            % Value for enabling the torque
TORQUE_DISABLE              = 0;            % Value for disabling the torque
DXL_MINIMUM_POSITION_VALUE  = 0;%-10000; %-150000;       % Dynamixel will rotate between this value
DXL_MAXIMUM_POSITION_VALUE  = 2000; %10000; %150000;       % and this value (note that the Dynamixel would not move when the position value is out of movable range. Check e-manual about the range of the Dynamixel you use.)
DXL_MOVING_STATUS_THRESHOLD = 10; %20;          % Dynamixel moving status threshold
ESC_CHARACTER               = 'e';          % Key for escaping loop
COMM_SUCCESS                = 0;            % Communication Success result value
COMM_TX_FAIL                = -1001;        % Communication Tx Failed
% Initialize PortHandler Structs
% Set the port path
% Get methods and members of PortHandlerLinux or PortHandlerWindows
port_num = portHandler(DEVICENAME);
% Initialize PacketHandler Structs
packetHandler();
% Initialize Groupsyncwrite Structs
groupwrite_num = groupSyncWrite(port_num, PROTOCOL_VERSION, ADDR_PRO_GOAL_POSITION, LEN_PRO_GOAL_POSITION);
% Initialize Groupsyncread Structs for Present Position
groupread_num = groupSyncRead(port_num, PROTOCOL_VERSION, ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
index = 1;
dxl_comm_result = COMM_TX_FAIL;           % Communication result
dxl_addparam_result = false;              % AddParam result
dxl_getdata_result = false;               % GetParam result
dxl_goal_position = [DXL_MINIMUM_POSITION_VALUE DXL_MAXIMUM_POSITION_VALUE];         % Goal position
dxl_error = 0;                            % Dynamixel error
dxl1_present_position = 0;                % Present position
dxl2_present_position = 0;
% Open port
if (openPort(port_num))
    fprintf('Succeeded to open the port!\n');
else
    unloadlibrary(lib_name);
```

```
  fprintf('Failed to open the port!\n');
  input('Press any key to terminate...\n');
  return;
end
% Set port baudrate
if (setBaudRate(port_num, BAUDRATE))
  fprintf('Succeeded to change the baudrate!\n');
else
  unloadlibrary(lib_name);
  fprintf('Failed to change the baudrate!\n');
  input('Press any key to terminate...\n');
  return;
end
% Enable Dynamixel#1 Torque
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL0_ID, ADDR_PRO_TORQUE_ENABLE,
TORQUE_ENABLE);
dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
if dxl_comm_result ~= COMM_SUCCESS
  fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
elseif dxl_error ~= 0
  fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
else
  fprintf('Dynamixel #%d has been successfully connected \n', DXL0_ID);
end
% Enable Dynamixel#1 Torque
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL1_ID, ADDR_PRO_TORQUE_ENABLE,
TORQUE_ENABLE);
dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
if dxl_comm_result ~= COMM_SUCCESS
  fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
elseif dxl_error ~= 0
  fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
else
  fprintf('Dynamixel #%d has been successfully connected \n', DXL1_ID);
end
% Enable Dynamixel#2 Torque
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL2_ID, ADDR_PRO_TORQUE_ENABLE,
TORQUE_ENABLE);
dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
if dxl_comm_result ~= COMM_SUCCESS
  fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
elseif dxl_error ~= 0
  fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
else
  fprintf('Dynamixel #%d has been successfully connected \n', DXL2_ID);
```

```
end
%
% % Add parameter storage for Dynamixel#0
% dxl_addparam_result = groupSyncReadAddParam(groupread_num, DXL0_ID);
%
% % Add parameter storage for Dynamixel#1 present position value
% dxl_addparam_result = groupSyncReadAddParam(groupread_num, DXL1_ID);
% % if dxl_addparam_result ~= true
% %   fprintf('[ID:%03d] groupSyncRead addparam failed', DXL1_ID);
% %   return;
% % end
%
% % Add parameter storage for Dynamixel#2 present position value
% dxl_addparam_result = groupSyncReadAddParam(groupread_num, DXL2_ID);
% % if dxl_addparam_result ~= true
% %   fprintf('[ID:%03d] groupSyncRead addparam failed', DXL2_ID);
% %   return;
% % end
%
%
% while 1
%     if input('Press any key to continue! (or input e to quit!)\n', 's') == ESC_CHARACTER
%         break;
%     end
%
%     %Add Dynamixel#1 goal position value to the Syncwrite storage
%     dxl_addparam_result = groupSyncWriteAddParam(groupwrite_num, DXL0_ID,
typecast(int32(dxl_goal_position(index)), 'uint32'), LEN_PRO_GOAL_POSITION);
%     if dxl_addparam_result ~= true
%         fprintf('[ID:%03d] groupSyncWrite addparam failed', DXL0_ID);
%         return;
%     end
%
%     % Add Dynamixel#1 goal position value to the Syncwrite storage
%     dxl_addparam_result = groupSyncWriteAddParam(groupwrite_num, DXL1_ID,
typecast(int32(dxl_goal_position(index)), 'uint32'), LEN_PRO_GOAL_POSITION);
%     if dxl_addparam_result ~= true
%         fprintf('[ID:%03d] groupSyncWrite addparam failed', DXL1_ID);
%         return;
%     end
%
%     % Add Dynamixel#2 goal position value to the Syncwrite parameter storage
%     dxl_addparam_result = groupSyncWriteAddParam(groupwrite_num, DXL2_ID,
typecast(int32(dxl_goal_position(index)), 'uint32'), LEN_PRO_GOAL_POSITION);
%     if dxl_addparam_result ~= true
%         fprintf('[ID:%03d] groupSyncWrite addparam failed', DXL2_ID);
%         return;
%     end
```

```
%
%     % Syncwrite goal position
%     groupSyncWriteTxPacket(groupwrite_num);
%     dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
%     if dxl_comm_result ~= COMM_SUCCESS
%         fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
%     end
%
%     % Clear syncwrite parameter storage
%     groupSyncWriteClearParam(groupwrite_num);
%
%     while 1
%         % Syncread present position
%         groupSyncReadTxRxPacket(groupread_num);
%         dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
%         if dxl_comm_result ~= COMM_SUCCESS
%             fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
%         end
%
%         % Check if groupsyncread data of Dynamixel#1 is available
%         dxl_getdata_result = groupSyncReadIsAvailable(groupread_num, DXL1_ID,
ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
%         if dxl_getdata_result ~= true
%           fprintf('[ID:%03d] groupSyncRead getdata failed', DXL1_ID);
%           return;
%         end
%
%         % Check if groupsyncread data of Dynamixel#2 is available
%         dxl_getdata_result = groupSyncReadIsAvailable(groupread_num, DXL2_ID,
ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
%         if dxl_getdata_result ~= true
%           fprintf('[ID:%03d] groupSyncRead getdata failed', DXL1_ID);
%           return;
%         end
%
%         % Get Dynamixel#1 present position value
%         dxl1_present_position = groupSyncReadGetData(groupread_num, DXL1_ID,
ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
%
%         % Get Dynamixel#2 present position value
%         dxl2_present_position = groupSyncReadGetData(groupread_num, DXL2_ID,
ADDR_PRO_PRESENT_POSITION, LEN_PRO_PRESENT_POSITION);
%
%         fprintf('[ID:%03d] GoalPos:%03d  PresPos:%03d\t[ID:%03d] GoalPos:%03d  PresPos:%03d\n',
DXL1_ID, dxl_goal_position(index), typecast(uint32(dxl1_present_position), 'int32'), DXL2_ID,
dxl_goal_position(index), typecast(uint32(dxl2_present_position), 'int32'));
%
```

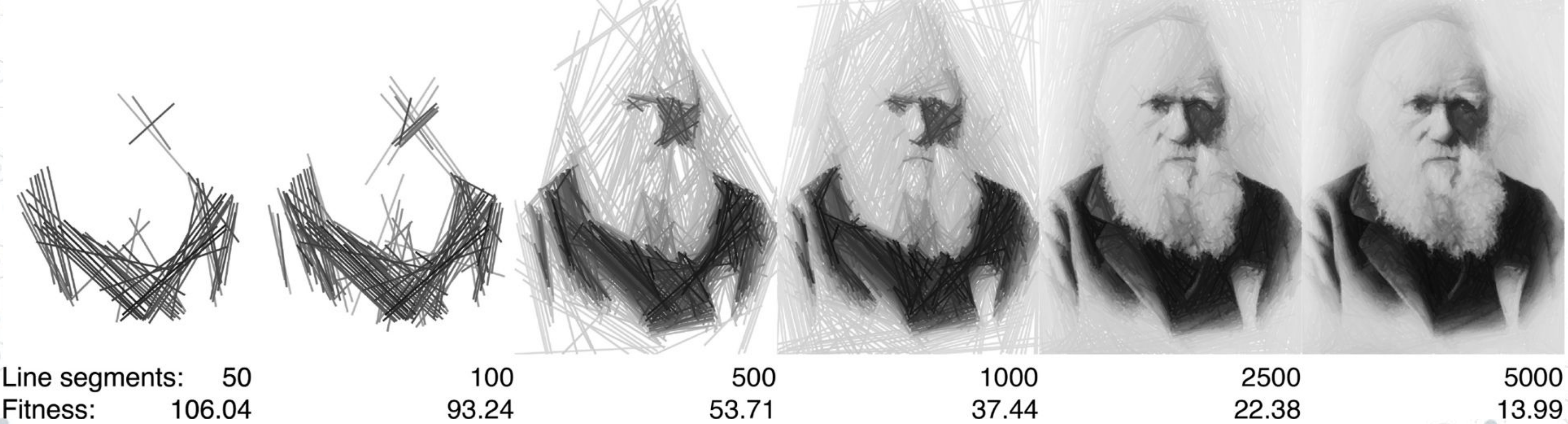
```
%         if ~((abs(dxl_goal_position(index) - typecast(uint32(dxl1_present_position), 'int32')) > DXL_MOVING_STATUS_THRESHOLD) || (abs(dxl_goal_position(index) - typecast(uint32(dxl2_present_position), 'int32')) > DXL_MOVING_STATUS_THRESHOLD))
%             break;
%         end
%     end
%
%     % Change goal position
%     if index == 1
%         index = 2;
%     else
%         index = 1;
%     end
% end
%
%
% % Disable Dynamixel#1 Torque
% write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL1_ID, ADDR_PRO_TORQUE_ENABLE, TORQUE_DISABLE);
% dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
% dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
% if dxl_comm_result ~= COMM_SUCCESS
%     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
% elseif dxl_error ~= 0
%     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
% end
%
% % Disable Dynamixel#2 Torque
% write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL2_ID, ADDR_PRO_TORQUE_ENABLE, TORQUE_DISABLE);
% dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
% dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
% if dxl_comm_result ~= COMM_SUCCESS
%     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
% elseif dxl_error ~= 0
%     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
% end
%
% % Close port
% closePort(port_num);
%
% Unload Library
% unloadlibrary(lib_name);
%
% close all;
% clear all;
```

SuperBot

Team #8

Ryan Ling, Wendyl Perez, Shizhuo Zhu

Literature Review

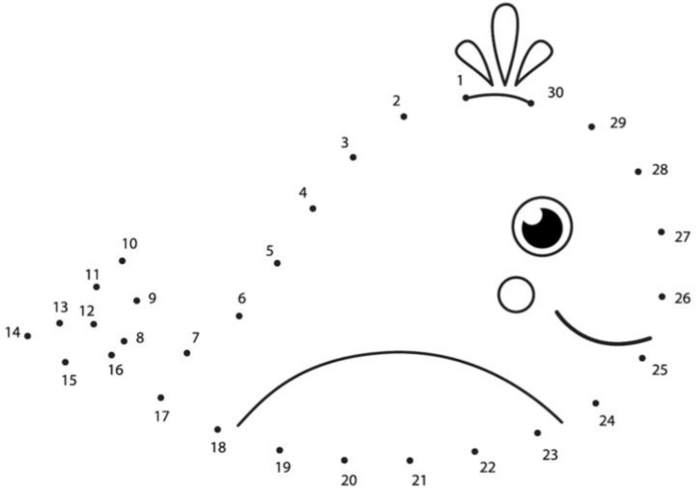


Drawing improvement with increased line segments [1].

- Pencil drawing
- Load-cell sensor to control pressure of pencil

Overview

- We wanted to drawing anything on a 2D plane.
    - Connect the dots puzzles to help children learn numbers and counting
    - Draw the Cool S (impress your friends)



Kinematics of Robot

- 3DOF: 2 revolute joints + 1 prismatic joint
- DH parameter

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	a_1	d_2	θ_2
3	0	a_2	d_3	0

- Inverse Kinematics:

$$T_{0\_e} = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 & a2*\cos(\theta_1+\theta_2)+a1*\cos\ \theta_1 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & a2*\sin(\theta_1+\theta_2)+a1*\sin\ \theta_1 \\ 0 & 0 & 1 & d_2+d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$d_3 = P_z - d_2$$

$$cos\theta_2 = (P_x^2 + P_y^2 - a_1^2 - a_2^2)/(2*a_1*a_2)$$

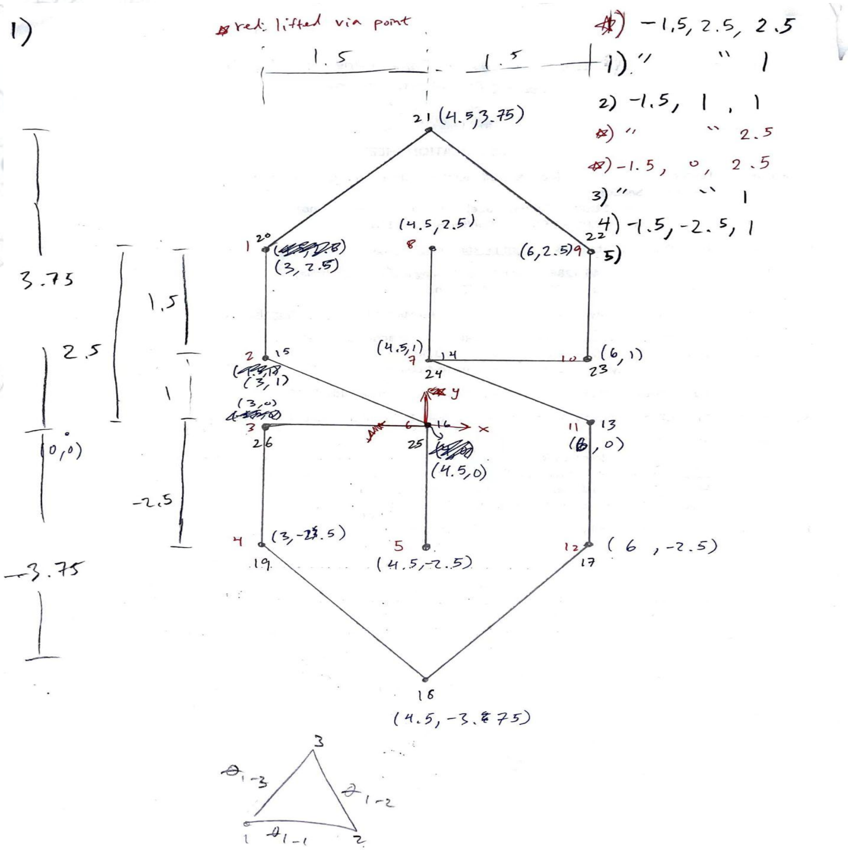
$$sin\theta_2 = \sqrt{1 - cos\theta_2{}^2}$$

$$\theta_2 = atan2(sin\theta_2, cos\theta_2)$$

$$\theta_1 + \theta_2 = atan2(r_{21}, r_{11})$$

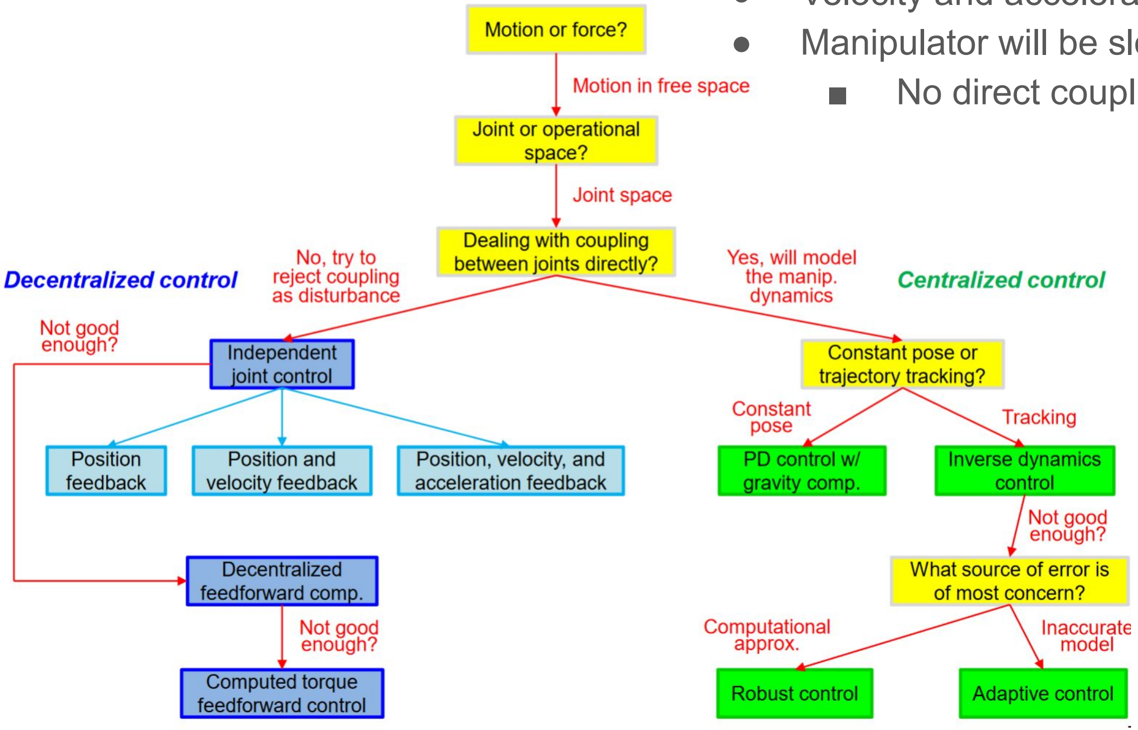
Objectives

- Trajectory Generation



- Controller Design
  - Control each joint as a single-input/single-output system
  - Position Feedback
  - Assume a smooth writing implementation and neglect the force on the robot due to the forces at the pen tip, so no need for force sensor.

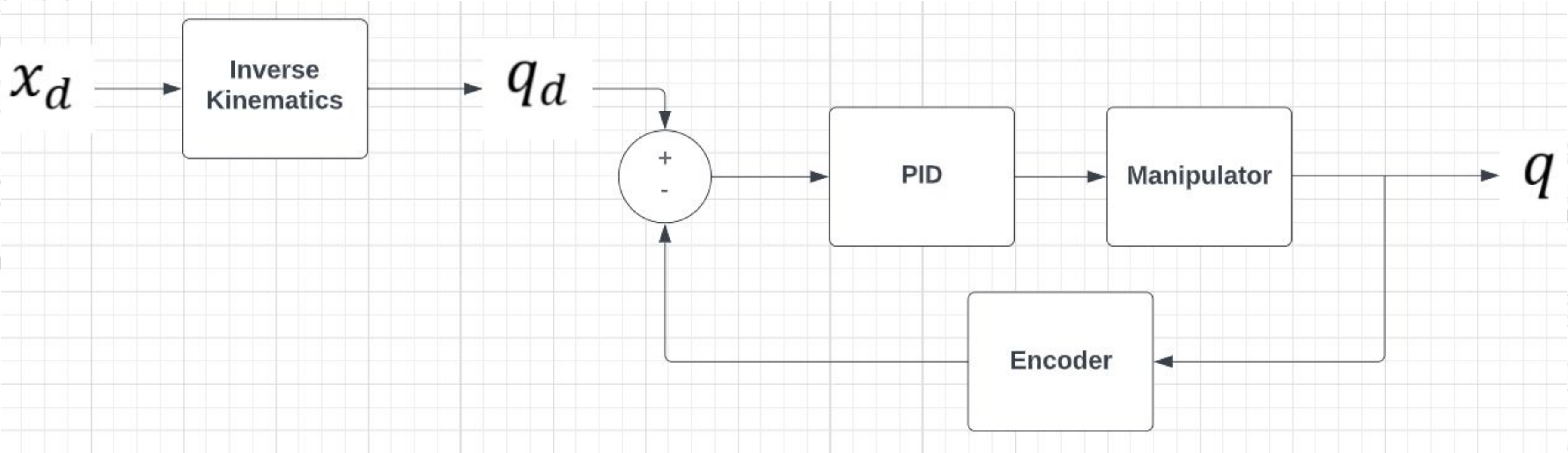
Decentralized Control: Position Control



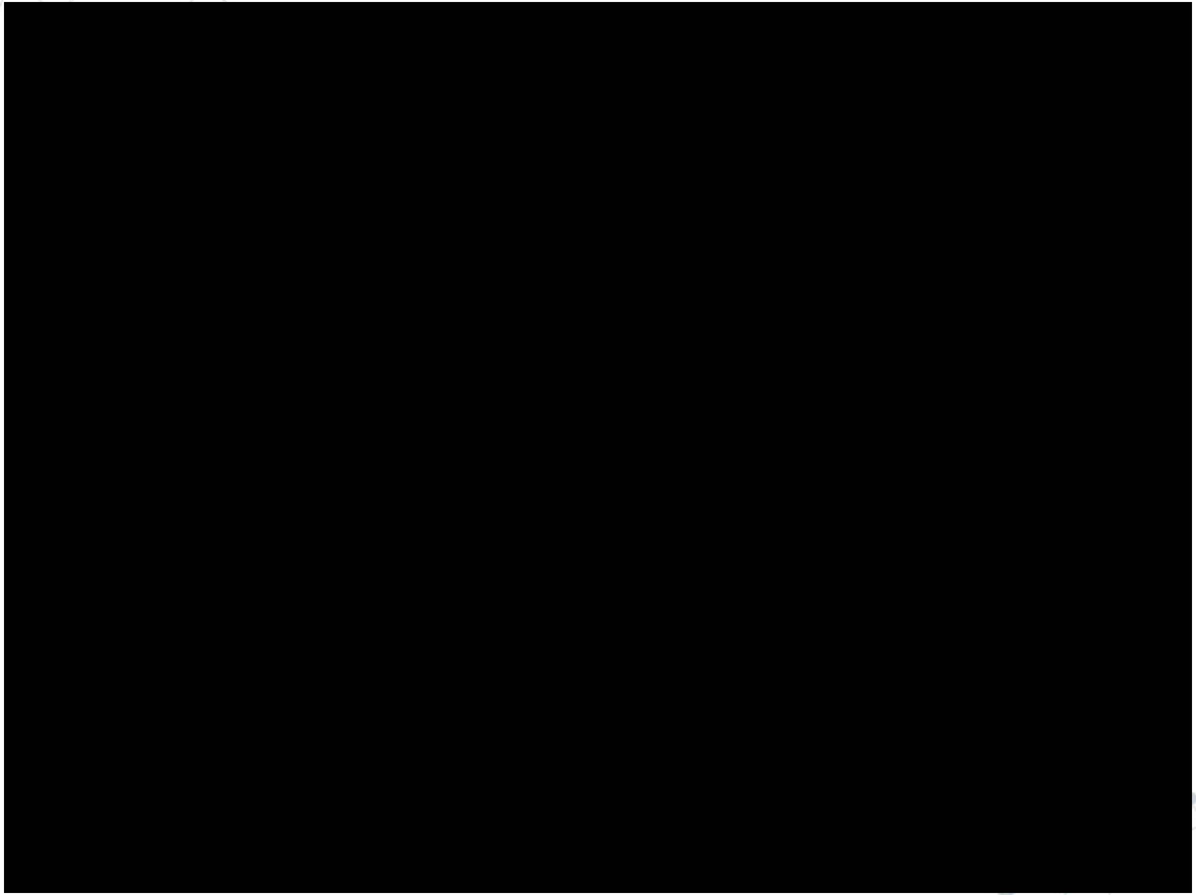
- Velocity and acceleration are of no interest => position
- Manipulator will be slow moving so we can assume:
    - No direct coupling between joints => decentralized

Decentralized Control: Tuning

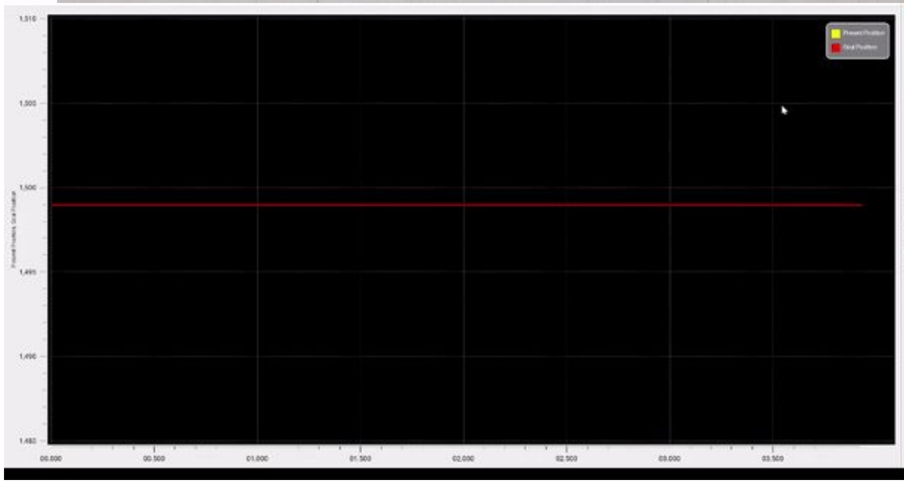
- Choosing suitable K_P, K_I, and K_D values
  - Started with K_P
  - Overshoot => Add K_D
  - Steady State => K_I

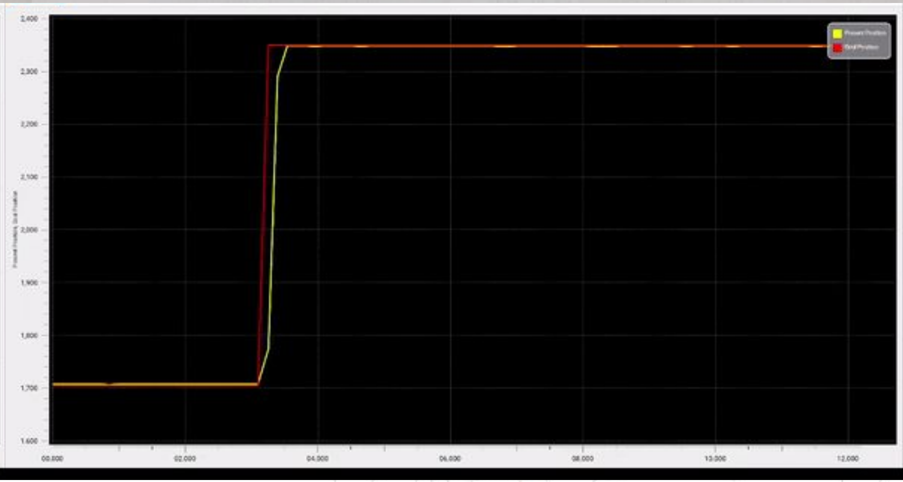


Backup Video

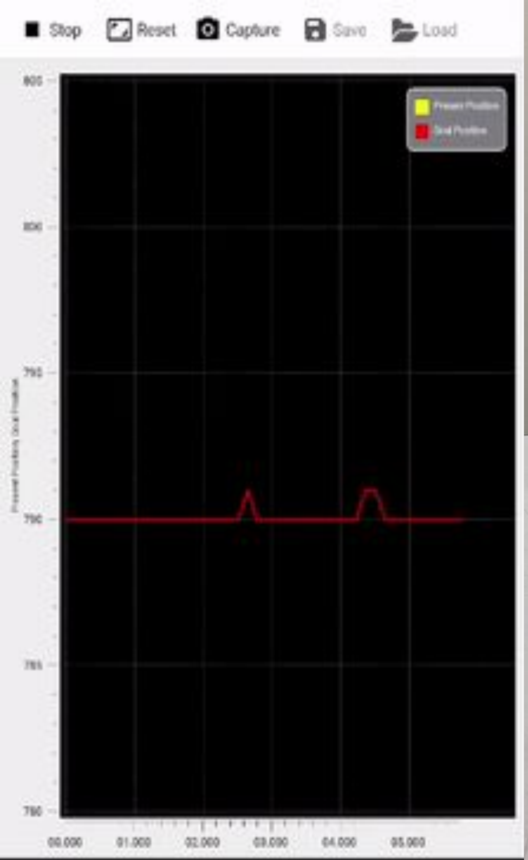


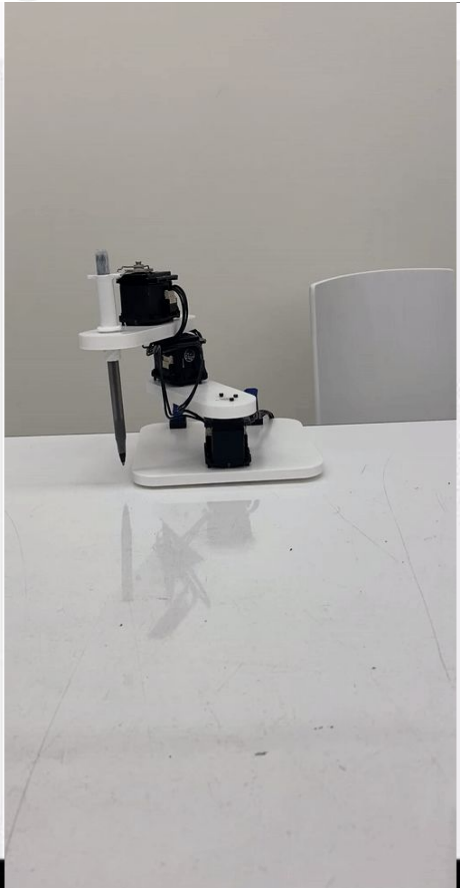
$K_P=K_I=K_D=0$ (gifs are not synced)

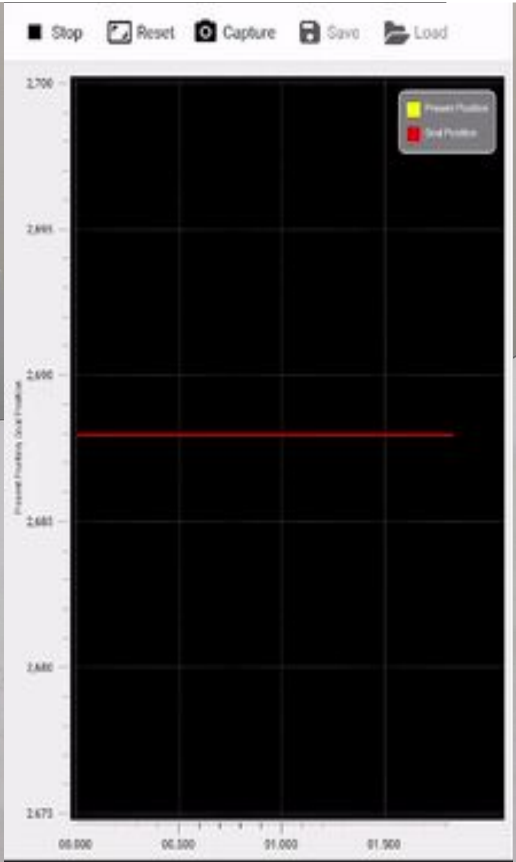


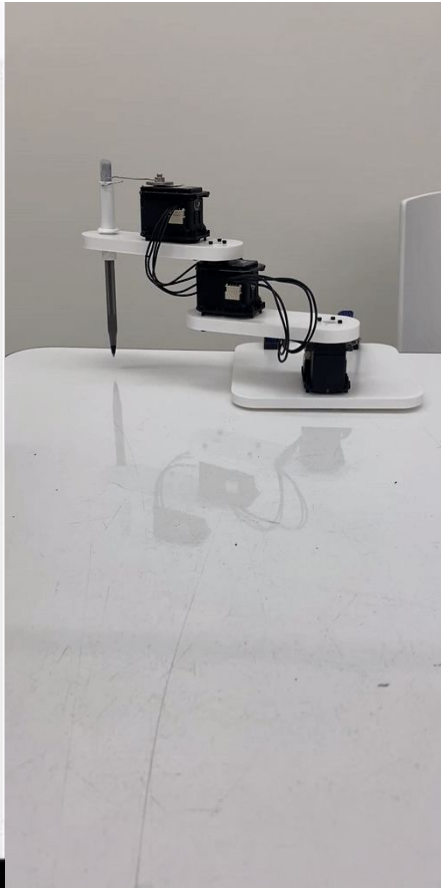


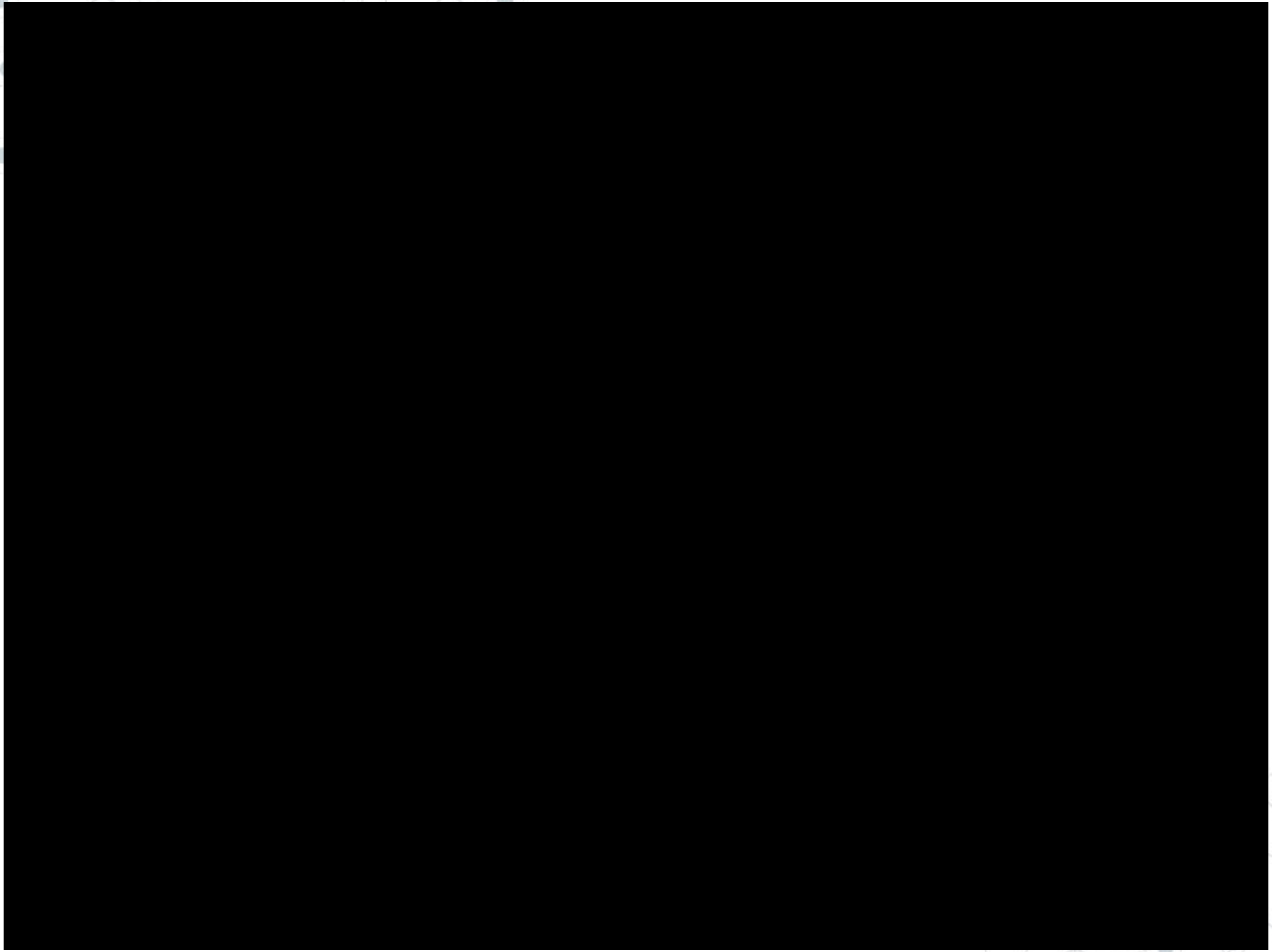
K_P=700, K_I=200, K_D=1500 (gifs not synced)



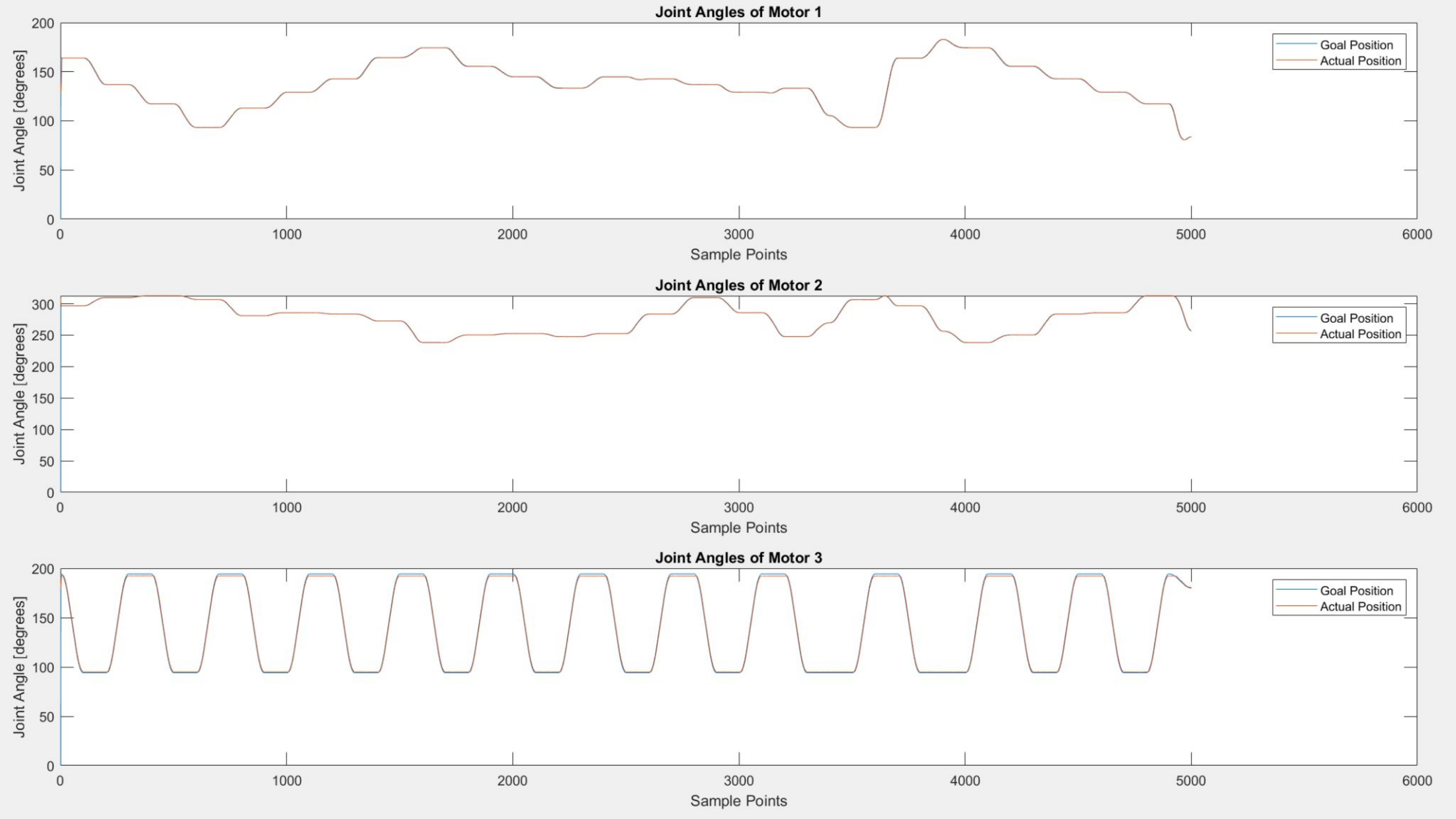




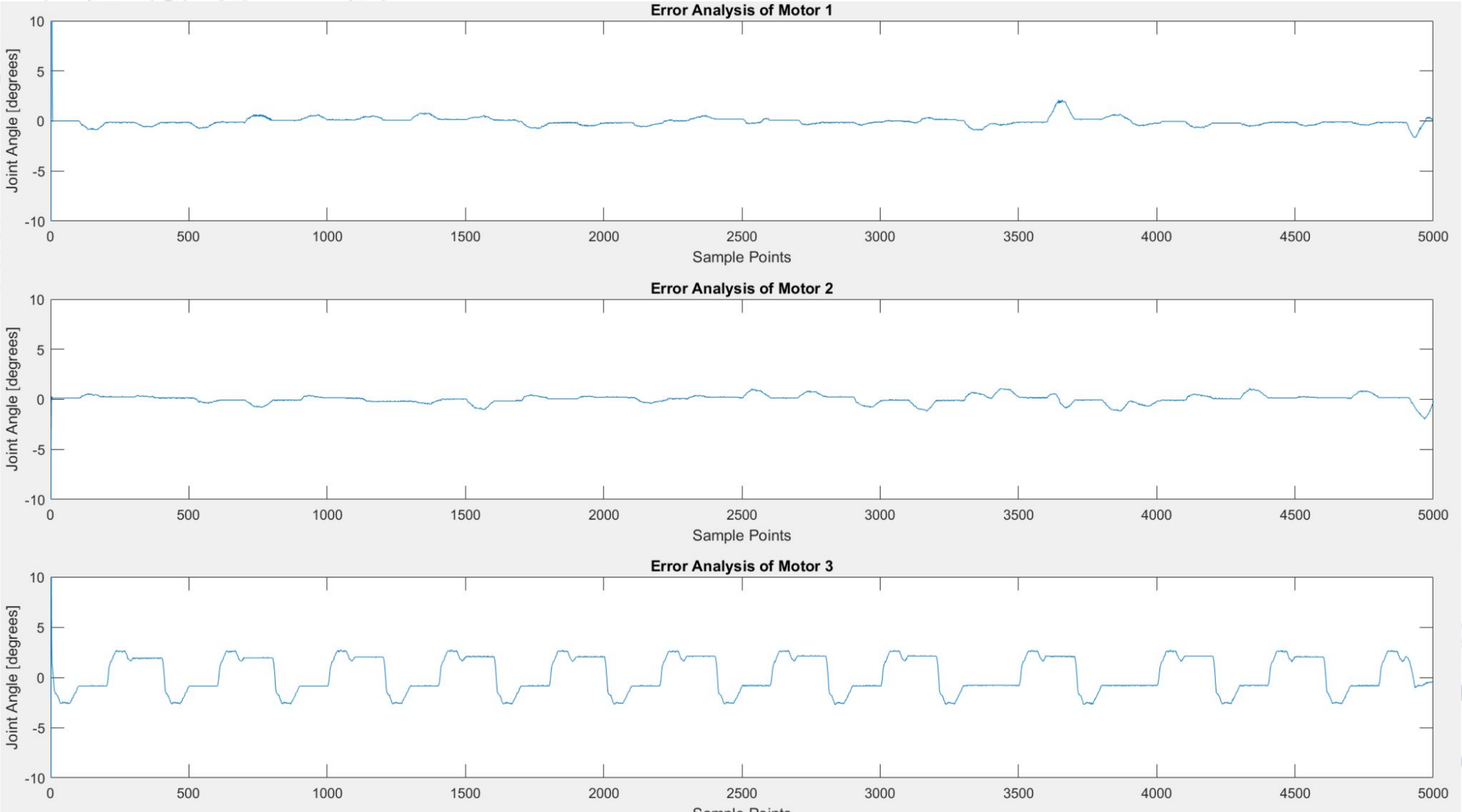




Joint Angles



Joint Angle Errors



Discussion: Limitations of Position Control

- Unable to control velocity and acceleration
- Limited in speed
  - Faster => coupling is noticeable
- Feedback control
  - No planning ahead; reactive
  - For high speeds, feedforward is beneficial

Discussion: Challenges

- Data transmitted to the 3 motors was not being read fast enough
  - Caused ‘shakiness’ while drawing
  - Switched to a sync read and write to send data much faster
- Screw mechanism for the prismatic joint is too loose
  - Causes unwanted lines while the pen is pulled up
- Structural integrity
  - High moment arm causes manipulator to have an uneven drawing plane

Thanks

References

[1] Adamik, M., Goga, J., Pavlovicova, J., Babinec, A., & Sekaj, I. (2021, October 21). *Fast robotic pencil drawing based on image evolution by means of genetic algorithm*. Robotics and Autonomous Systems. Retrieved June 7, 2022, from https://www.sciencedirect.com/science/article/pii/S0921889021001974