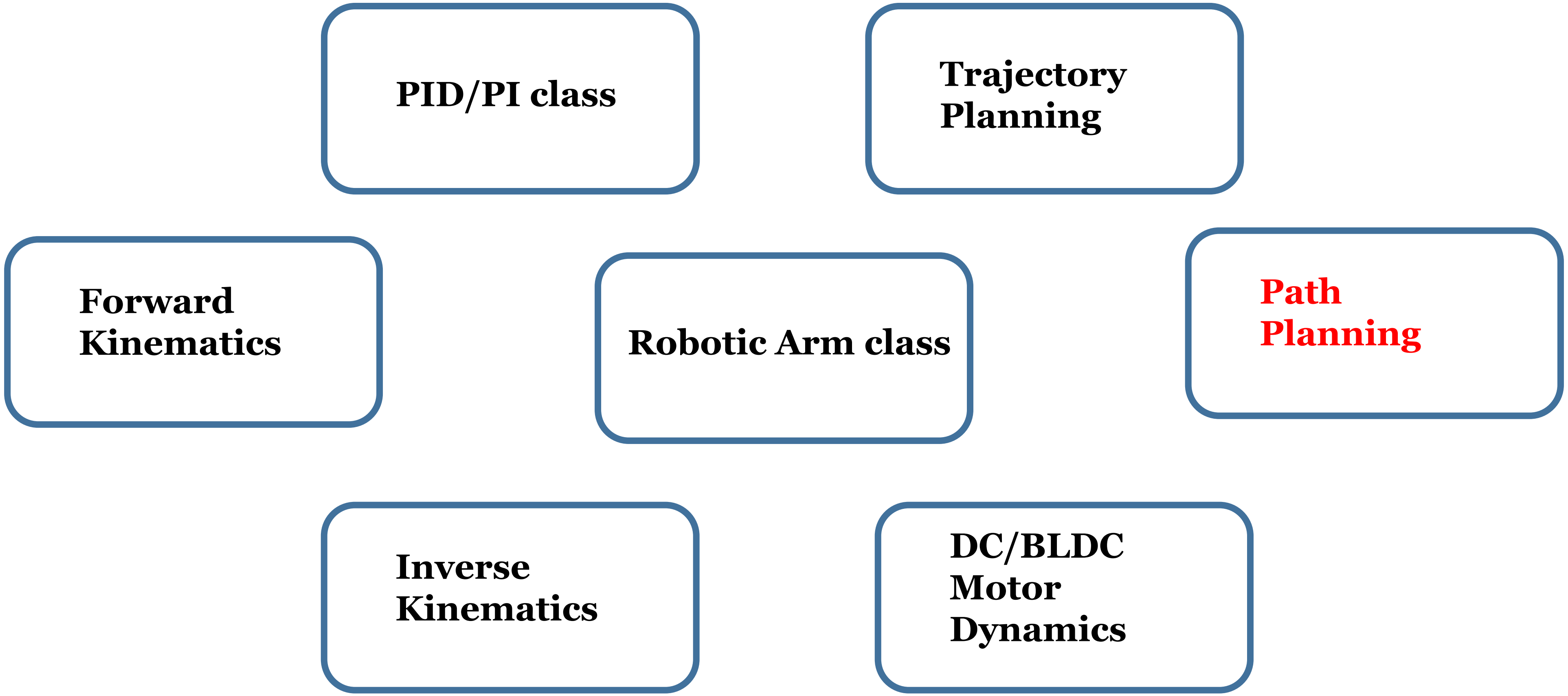
Robotic Manipulator

Kartik Sah

Overview

- To develop a general n-revolute robotic arm class in python.
- To implement the theory of kinematics, velocity analysis, dynamics, PID controller, and trajectory generation - Mark Spong.
- Solve inverse kinematics using Deep Neural Network, as there is no general solution to solve for inverse kinematics of a non-intersecting wrist arm.
- Programming done based on OOPs methodology.

Structure of the overall Schema



Robotic Arm Class

- To initialize a Robotic Manipulator

```
if __name__ == '__main__':
    arm = robotic_arm()
    arm.set_joints(4)
    arm.set_dh_param_dict([[0,0,1,'R'],[-np.pi/2.,0,0,'R'],[0,1,0,'R'],[0,1,0,'R']])
```

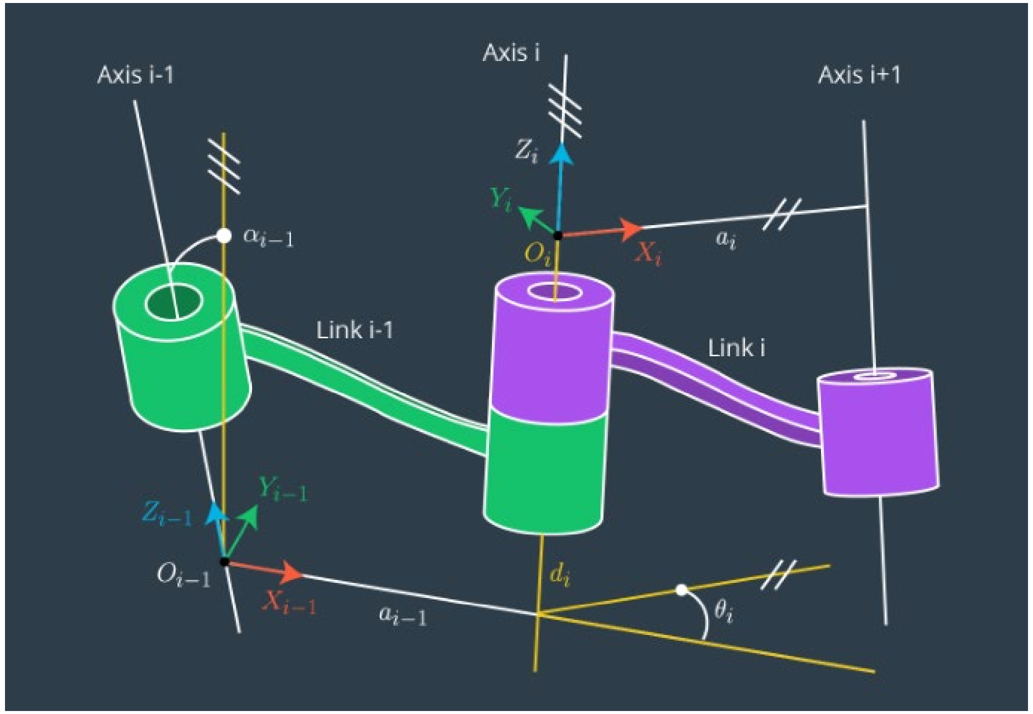
- One can set the Number of joints, and assign DH parameters to initialize a robotic manipulator.

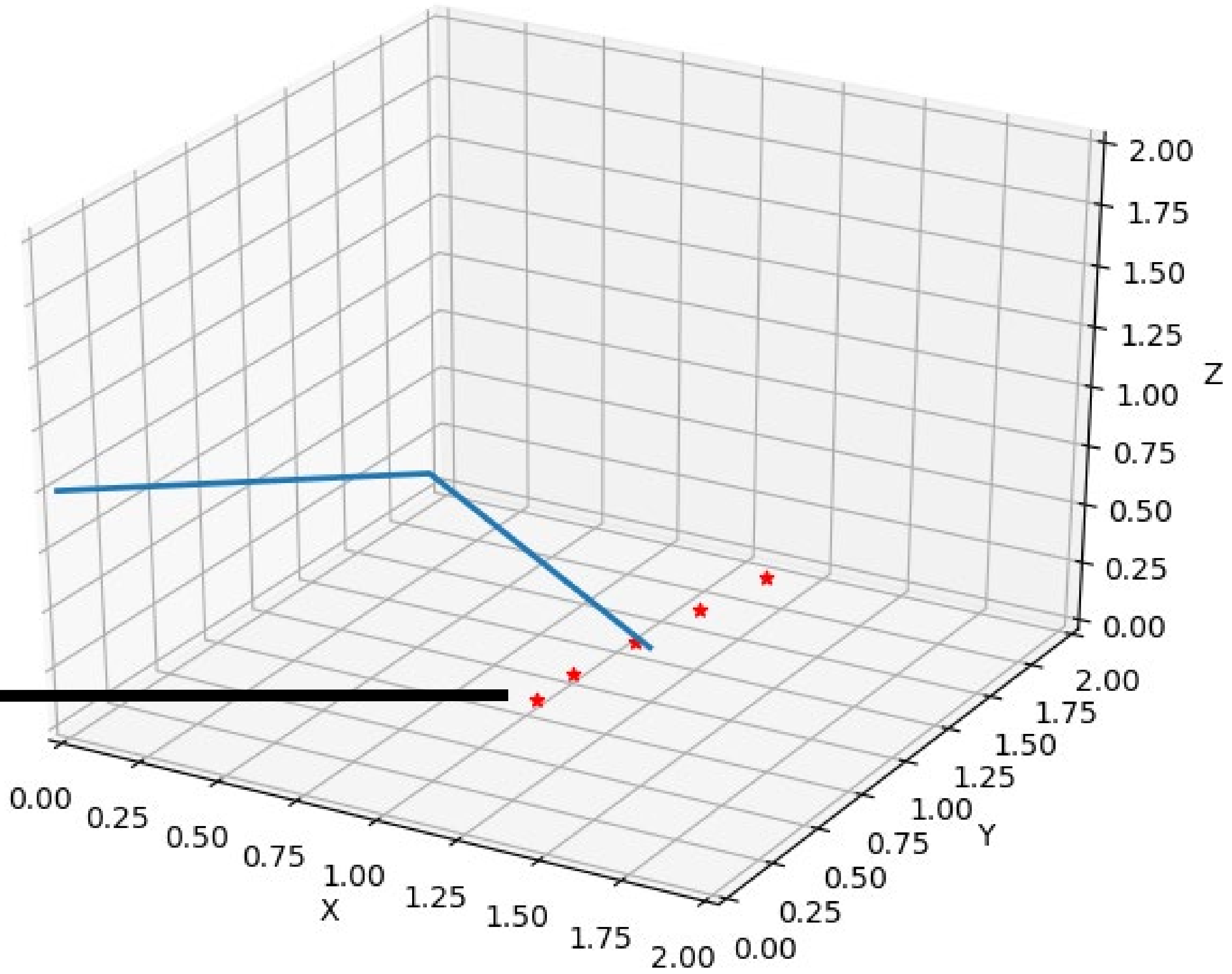
Inverse and Forward Kinematics Class

- Forward Kinematics:
  - Input: Angular Position/Angular Velocity to the joints
  - Output: x,y,z coordinates of the links
  - Implemented by computing the transformation matrices (which transform from)
- Inverse Kinematics:
  - Input: x,y,z coordinates of the final link
  - Output: Angular Position/Angular Velocity to the joints

$$ {}^{0}_{G}T = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T\,{}^{3}_{4}T\,{}^{4}_{5}T\,{}^{5}_{6}T\,{}^{6}_{G}T $$

$$ {}^{i-1}_{\;\;i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$





Target Coordinates

Code Results

- Inverse Kinematics using Gradient Descent
- Learning Rate : 0.2
- Error Tolerance : 0.05

```
[0.55859932 0.06284561 0.29364679 0.        ]
Traget: [[1.6], [1.0], [0.6]] ,  Predicted: [[1.64100609]
 [1.02562881]
 [0.58820643]]
(3, 1)
[ 0.62024949 -0.02462922  0.72444746  0.        ]
Traget: [[1.4], [1.0], [0.4]] ,  Predicted: [[1.43595964]
 [1.02568546]
 [0.38054807]]
(3, 1)
[0.69473827 0.02811737 0.89580148 0.        ]
Traget: [[1.2], [1.0], [0.2]] ,  Predicted: [[1.2309228 ]
 [1.025769  ]
 [0.17391671]]
(3, 1)
[0.77533319 0.13339902 0.96858004 0.        ]
Traget: [[1.0], [1.0], [0.0]] ,  Predicted: [[ 1.03053524]
 [ 1.00999665]
 [-0.02510703]]
(3, 1)
[0.67474356 0.08431536 1.16514516 0.        ]
Traget: [[1.0], [0.8], [0.0]] ,  Predicted: [[ 1.02471782]
 [ 0.81977866]
 [-0.03302986]]
(matrix([[ 0.55859932,  0.06284561,  0.29364679,  0.        ],
        [ 0.62024949, -0.02462922,  0.72444746,  0.        ],
        [ 0.69473827,  0.02811737,  0.89580148,  0.        ],
        [ 0.77533319,  0.13339902,  0.96858004,  0.        ],
        [ 0.67474356,  0.08431536,  1.16514516,  0.        ]]),
```

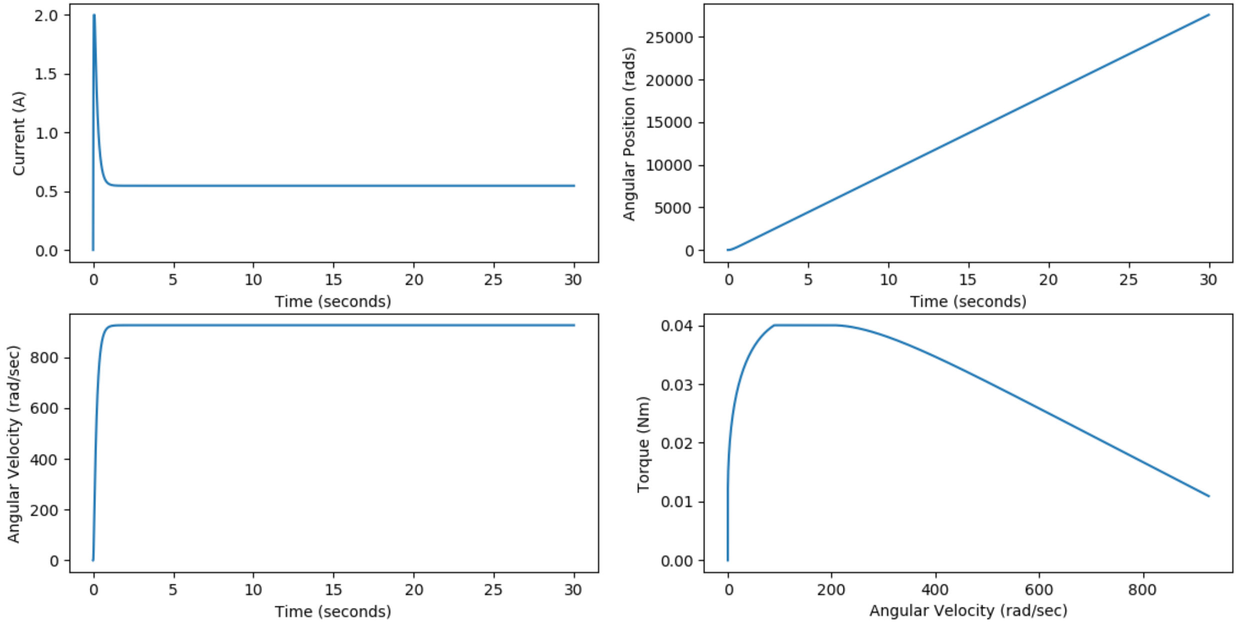
DC Motor Class

- Generalized class for DC motor

```
motor_1 = Dc_motor()
motor_1.set_motor_param(R,L,b,kb,kt,I_motor)
motor_1.set_load_torque(T_load)
motor_1.set_voltage(V)
motor_1.set_duration(30.0)
motor_1.set_controller()
motor_1.set_sample_time(0.001)

pid = PIDController(kp = 0.6, ki = 3, kd = 0.001, max_windup = 20,
        start_time = 0, alpha = 0.75, u_bounds = [0.0, V])
```

- Plots



Trajectory Planning

- Use Linear Parabolic blend to form a continuous trajectory connecting the discrete points.
- Path Planning module will provide discrete points(Based on some algorithm A*, RRT, etc.)
- Using the above points as inputs, along with the required time between two points, and the acceleration required.

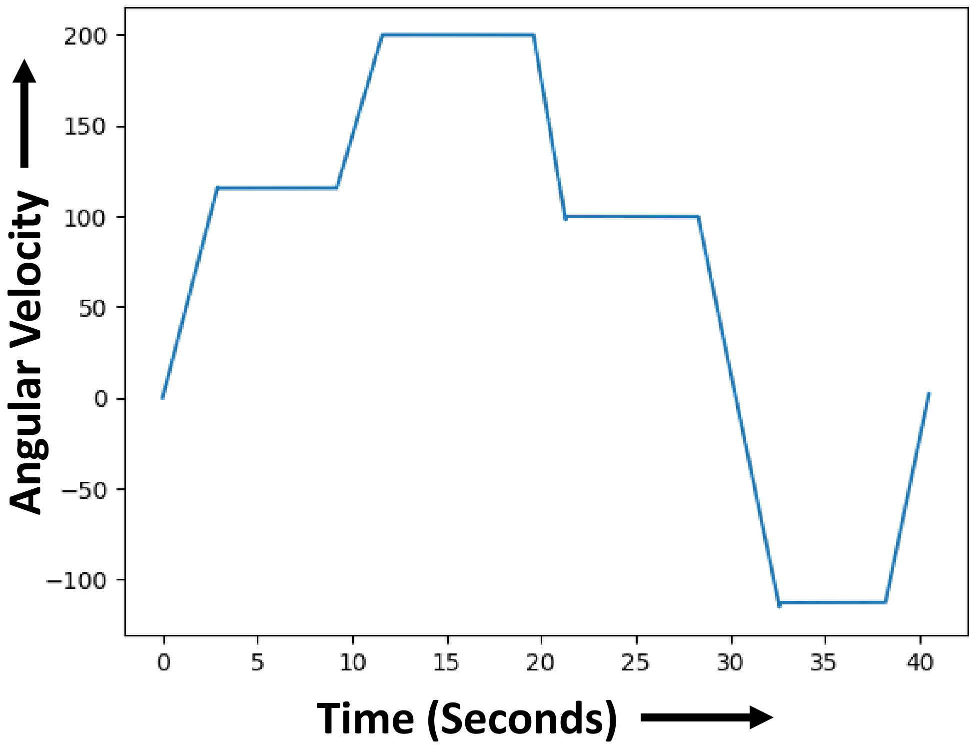
Example:

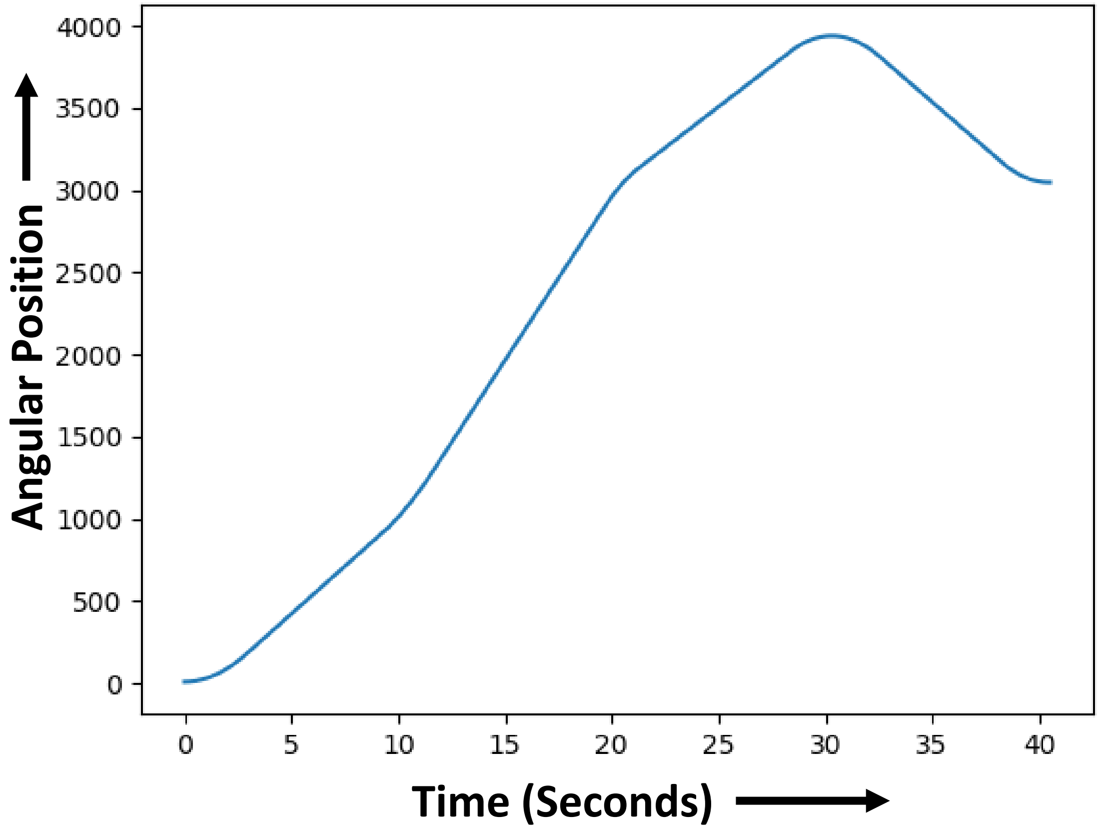
Discrete Points = 10,1000,3000,4000,3000 radian

Time = 10,10,10,10

Acceleration = 40,35,60,50,50

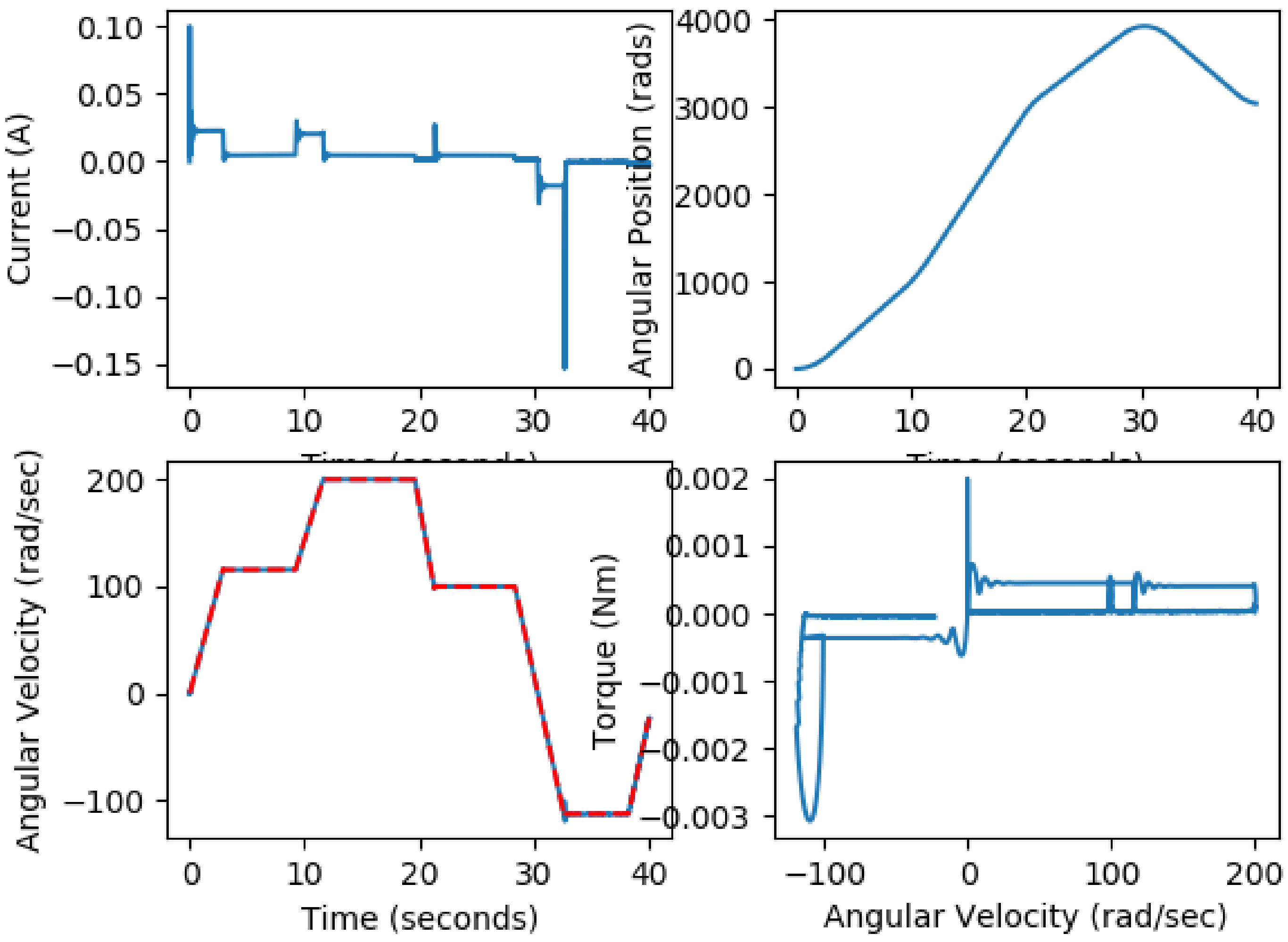
Generated Trajectory as per LPSB





PID Controller

- To make the Manipulator follow the generated trajectory, we feed the controller the reference trajectory and the controller regulates the Voltage to make the manipulator follow the desired trajectory.
- Assumptions: Independent Joint control
- Gain are currently selected through trial and error. Gain and Phase margin plot needed for better selection of gains.



Red line is the desired trajectory

Neural Network for Inverse Kinematics

- To avoid the process of gradient descent every time, another method is to use neural networks to learn the inverse kinematics.
- Advantages are fast computation when the function is learned
- Disadvantages, currently is that point with multiple solution.