MAE 204: Final Project

Motion Control of a Mobile Manipulator Robot

Arnav Wayade
Department of Mechanical and Aerospace Engineering
University of California San Diego
awayade@ucsd.edu

Chaitanya Tambat
Department of Mechanical and Aerospace Engineering
University of California San Diego
ctambat@ucsd.edu

Abstract*—The interest in robotics has been on the rise because of the practical value it provides. Due to their very nature, robots are used for tasks that are repetitive, require a high degree of precision, and for tasks where it is unsafe for humans to work. The problem of localization and control of a robot to perform certain tasks has many novel uses in Modern Robotics. A specific utility of robots is for picking up and placing an object in a specified location and orientation. A robot that can perform this task of pick-up-and-place is the KUKA youBot. This project discusses and provides a solution to control the motion of this robot to perform a certain task. This project will run the simulation of the robot control in CoppeliaSIM.

***Keywords*— KUKA youBot, CoppeliaSIM, Trajectory, End-effector, Transformation matrices, Feedforward, Feedback, Proportional Gain, Integral Gain**

I. Introduction

Pick and place robots are commonly used in modern manufacturing environments. Pick and place automation speeds up the process of picking up parts or items and placing them in other locations. Pick and place robots handle repetitive tasks while freeing up human workers to focus on more complex work. The KUKA youBot is one such type of robot designed to perform the above specified task. The youBot is a mobile robotic arm developed by KUKA. Its arm has five degrees of freedom and a linear gripper. Its base has four mecanum wheels allowing for omnidirectional movement. These wheels are efficiently modeled using asymmetric friction.

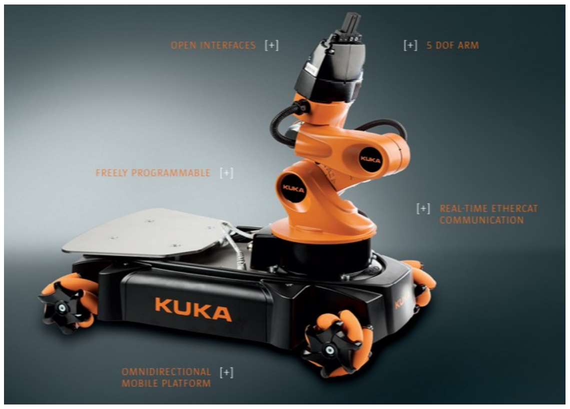

OPEN INTERFACES [+]
[+] 5 DOF ARM
FREELY PROGRAMMABLE [+]
[+] REAL-TIME ETHERCAT COMMUNICATION
KUKA
OMNIDIRECTIONAL [+] MOBILE PLATFORM

Figure 1: KUKA youBot

II. Problem Formulation

The robot movement needs to be planned such that the robot picks up and places the object, cube in our case, in the specified location. But during this formulation, the robot movement might be affected by control input noise or other external factors and the accuracy of the end-effector configuration might be lost and the cumulative error may lead to high errors in performing the desired task.

So to make up for these errors, the project discusses implementation of a feedback loop to correct the end-effector configuration at each step and hence obtain a feedback loop controlled function which gives us values of the configuration of the robot position and orientation and the pick-up arm angles at each timestep. The problem can be solved in separate steps.

Hence, the problem can be broken into four parts, namely

1. Trajectory Generation (Milestone 2)

In this part of the problem, we will test the performance of the end-effector of the youBot in the CoppeliaSIM youBot scene 8. The CoppeliaSIM youBot scene 8 takes a csv (comma separated values) file as input which contains the Transformation matrix T_{se} which is the transformation matrix of the end-effector in the world fixed frame {s}. If we take matrix T_{se} as shown below

$$T = [\,[R \quad p], \\ [0 \quad 1]\,]$$

Where R is the rotation matrix, and p is the position vector of the end effector frame.

We supply the csv file to CoppeliaSIM in the following format

($R_{11}, R_{12}, R_{13}, R_{21}, R_{22}, R_{23}, R_{31}, R_{32}, R_{33}, p_1, p_2, p_3$, gripper)

We generate this csv file using the TrajectoryGenerator function in python.

2. Determination of Next State of the Robot (Milestone 1)

In this step, we generate the function in python which will give us the next state of the robot in a csv file which is fed to Coppelia youBot scene 6. The function that gives us the state is NextState function and gives us csv files in the format given below.

(chassis phi, chassis x, chassis y, J1, J2, J3, J4, J5, W1, W2, W3, W4, gripper state)
Where chassis phi is the orientation of the chassis, chassis x and y are the chassis locations in x-y plane, J1-J5 are joint angles of the arm of the youBot and W1-W4 are the wheel angles turned at that particular timestep.

3. Feedback Control to determine angular velocities of the wheel and arm joints
In feedback loop, we determine the error between desired position and actual position of the end effector and find out the required error compensatory velocities at each time step and give the required velocities to the NextState function to determine the new state of the robot.
We will check the performance of the robot for different Proportional and integral gains of the controller using

$$\dot{q}(t) = \dot{q}_d(t) + K_p\big(q_d(t) - q(t)\big) + K_i \int_0^t \big(q_d(t) - q(t)\big)\, dt$$

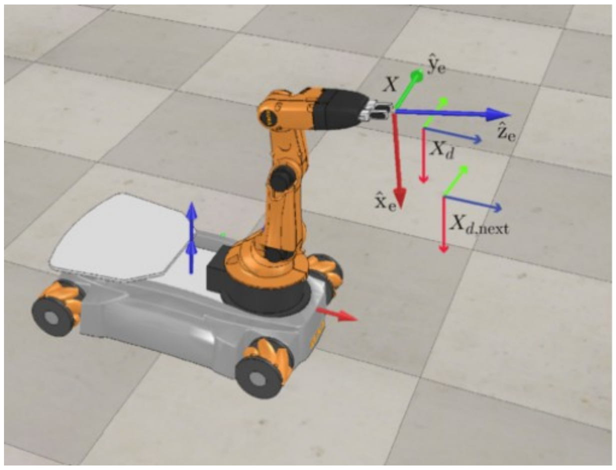


Figure 2: youBot in the CoppeliaSIM environment

4. Wrapper Script
In the wrapper script we put together all three functions in a loop for the program to calculate corrected end effector location and therefore the corrected robot position and orientation alongwith the joint angles at each timestep.
The wrapper script gives out a csv file containing the following data.
(chassis phi, chassis x, chassis y, J1, J2, J3, J4, J5, W1, W2, W3, W4, gripper state)
Where chassis phi is the orientation of the chassis, chassis x and y are the chassis locations in x-y plane, J1-J5 are joint angles of the arm of the youBot and W1-W4 are the wheel angles turned at that particular timestep.
The wrapper script generates a csv file which runs each step of the pick-up and place program goal within the specified amount of time.

III. Technical Approach

1. Trajectory Generation
In this part of the problem, we specify the initial configuration of the cube given by T_scube_initial and the final desired configuration of the cube T_scube_final.
Secondly, we specify the relative configuration of the end effector while grasping and in standoff position in the frame of the cube, viz. T_ce_grasp and T_ce_standoff.
To find the end effector configuration while grasping and standoff in the world fixed frame {s} we use the inverse of transformation formula to find T_se_grasp and T_se_standoff as shown below.

$$X_{start,end} = X_{start,s}X_{s,end} = X^{-1}_{s,start}X_{s,end}$$

Similarly, we find the end effector configuration in the world frame using the final desired configuration of the cube and the grasp and standoff matrices using the same above formula.

Then we use the screwtrajectory function from the Modern Robotics library to generate interpolated transformation matrices for the end effector to move from one location to the next desired location.
The csv file is stored in the following format.
(R_{11}, R_{12}, R_{13}, R_{21}, R_{22}, R_{23}, R_{31}, R_{32}, R_{33}, p_1, p_2, p_3, gripper)

2. Determination of the Next State of the Robot
From the current state of the robot we get its orientation and location as well as the angles of the arms and wheels.
In the NextState function we update the angles of the wheels and arms using the following formula
New angles=current angles + velocities*timestep
We thus get the new angles of the arm and the wheels of the robot.
Now to get to this new position of the chassis we find the chassis body twist using the following formula

$$\mathcal{V}_b = F\Delta\theta = \frac{r}{4}\begin{bmatrix} -\frac{1}{l+w} & \frac{1}{l+w} & \frac{1}{l+w} & -\frac{1}{l+w} \\ 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix}\Delta\theta$$

Where delta theta is

$$\Delta\theta = H(0)\mathcal{V}_b$$

and F is the force matrix in which l is the length of the robot, w is the wheel base. The values for the youBot are given as r = 0.0475, l = 0.47 / 2, w = 0.3 / 2.
Using the above formula we get the twist that is required to go from current state to next state. And we calculate the new state by using the same above formula.

$$X_{start,end} = X_{start,s}X_{s,end} = X^{-1}_{s,start}X_{s,end}$$

From the next state function we thus get the new state of the robot required to determine the configuration.

3. Feedback Control to determine angular velocities of the wheel and arm joints

In the feedback control, we use the desired configuration of the end effector from the ideal trajectory generated using the trajectory generator function and the actual configuration of the end effector and find out the error twist that is required to compensate for this error. In addition we also consider the ideal twist that is required to go from current desired end effector configuration to the next desired end effector configuration. The formulation is as shown below.

$$X(q,\theta) = T_{se}(q,\theta) = T_{sb}(q)\, T_{b0}\, T_{0e}(q) \in SE(3)$$

$$\mathcal{V}_e = J_e(\theta)\begin{bmatrix} u \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} J_{base}(\theta) & J_{arm}(\theta) \end{bmatrix}\begin{bmatrix} u \\ \dot{\theta} \end{bmatrix}$$

Now once we find the error twist and the required twist, we find the proportional error and integral error using the desired and actual end effector configuration. The integral error is calculated totaling the cumulative errors of each timestep.

Then we find the corrected twist required to go to next state using.

Corrected error twist = Ve + Kp*Error + Ki*Integral_error

From the corrected error twist we obtain the wheel and angular velocities of the arm using

$$\mathcal{V}_e = J_e(\theta)\begin{bmatrix} u \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} J_{base}(\theta) & J_{arm}(\theta) \end{bmatrix}\begin{bmatrix} u \\ \dot{\theta} \end{bmatrix}$$

To obtain the velocities we Moore Penrose Pseudoinverse of the Jacobian Je.

$$\left[Ad_{T_{eb}(\theta)}\right]\mathcal{V}_{b6} = \left[Ad_{T_{0e}^{-1}(\theta)\, T_{b0}^{-1}(\theta)}\right]\mathcal{V}_{b6} = \left[Ad_{T_{0e}^{-1}(\theta)\, T_{b0}^{-1}(\theta)}\right]F_6 u = J_{base}(\theta)u$$

These linear and angular velocities are then given to the NextState function to determine the next configuration of the youBot.

4. Wrapper Script

In the wrapper script we put together all three functions in a loop for the program to calculate corrected end effector location and therefore the corrected robot position and orientation alongwith the joint angles at each timestep.

The wrapper script gives out a csv file containing the following data.

(chassis phi, chassis x, chassis y, J1, J2, J3, J4, J5, W1, W2, W3, W4, gripper state)

Where chassis phi is the orientation of the chassis, chassis x and y are the chassis locations in x-y plane, J1-J5 are joint angles of the arm of the youBot and W1-W4 are the wheel angles turned at that particular timestep.

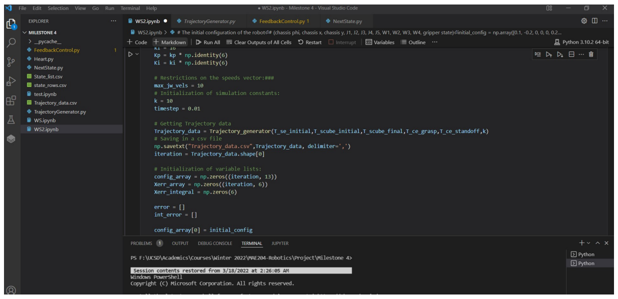
IV. Software Orientation

1. Python (VS Code)

All the above-mentioned codes have been scripted using the Python language, wherein we use the Numpy library for matrix multiplications and matrix operations.

In addition to the Numpy library we use the Modern Robotics library for functions like MatrixLog6, se3ToVec which are the Forward Kinematics functions that convert our matrices to desired form.

The GUI of the VS code which is the coding software is shown below.



WS2.ipynb > # The initial configuration of the robot# (chassis phi, chassis x, chassis y, J1, J2, J3, J4, J5, W1, W2, W3, W4, gripper state)initial_config = np.array([0.1, -0.2, 0, 0, 0, 0.2...

```
Kp = kp * np.identity(6)
Ki = ki * np.identity(6)

# Restrictions on the speeds vector:###
max_jw_vels = 10
# Initialization of simulation constants:
k = 10
timestep = 0.01

# Getting Trajectory data
Trajectory_data = Trajectory_generator(T_se_initial,T_scube_initial,T_scube_final,T_ce_grasp,T_ce_standoff,k)
# Saving in a csv file
np.savetxt("Trajectory_data.csv",Trajectory_data, delimiter=',')
iteration = Trajectory_data.shape[0]

# Initialization of variable lists:
config_array = np.zeros((iteration, 13))
Xerr_array = np.zeros((iteration, 6))
Xerr_integral = np.zeros(6)

error = []
int_error = []

config_array[0] = initial_config
```

```
PS F:\UCSD\Academics\Courses\Winter 2022\MAE204-Robotics\Project\Milestone 4>

 Session contents restored from 3/18/2022 at 2:26:05 AM
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
```

Figure 3: VS Code GUI (Python language)

2. CoppeliaSIM Edu

CoppeliaSim, formerly known as V-REP, is a robot simulator used in industry, education and research.

It is built around a distributed control architecture having Python and Lua scripts, or C/C++ plug-ins acting as individual, synchronous controllers. Additional asynchronous controllers can execute in another process, thread or machine via various middleware solutions (ROS, remote API, ZeroMQ) with programming languages such as C/C++, Python, Java and Matlab.

CoppeliaSim uses a kinematics engine for forward and inverse kinematics calculations, and several physics simulation libraries. CoppeliaSIM takes csv files in the desired form for the robot configuration, and takes the robot to each configuration while the simulation is running.

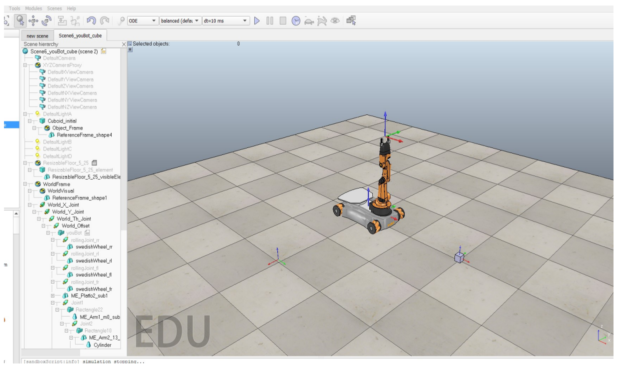


Figure 4: CoppeliaSIM software Interface

V. Results

For all the tasks we have used the Feedforward plus PI controller with varying gains. We observed the transient response for lower gains and tried multiple variations of gain values as we plotted all the six error values for each task.

- PI Control:

Digital controllers are implemented with discrete sampling periods and a discrete form of the PI equation is needed to approximate the integral of the error. This modification replaces the continuous form of the integral with a summation of the error and uses Δt as the time between sampling instances and nt as the number of sampling instances. Here N is determined using the time for each trajectory and Δt is the timestep we have given to interpolate the motion in CoppeliaSIM where Δt = 0.01 in our case.

- Implementation of Joint velocities' limits:

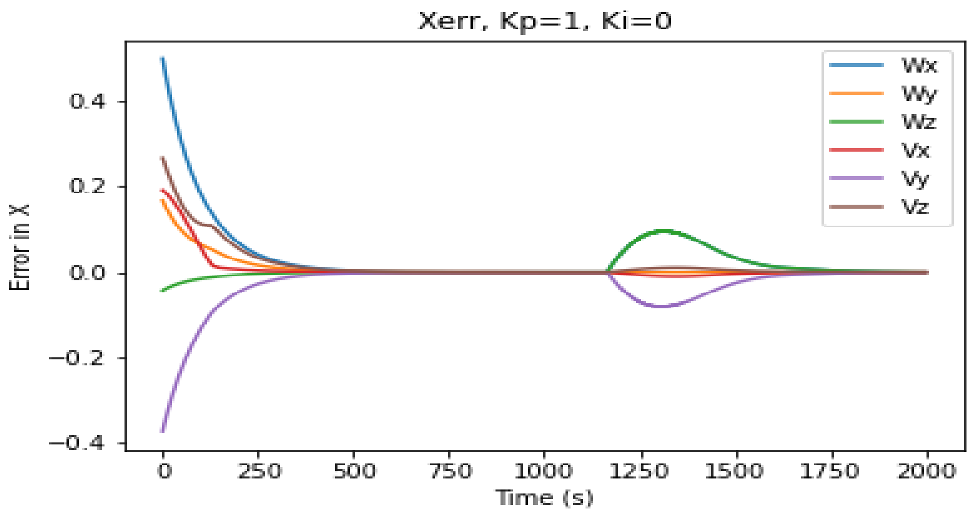
In our project, we implemented joint velocity limits to each joint as the Pseudoinverse of Jacobian might lead to very high values of joint velocities. We did this by comparing joint and wheel velocities obtained from the Feedback function to a maximum angular velocity value which we specified. We then made these velocity values to the maximum value by implementing the following in Python.

```
for i, jw_vel in enumerate(jw_vels):
        if abs(jw_vel) > max_jw_vels:
            jw_vels[i] = jw_vel / abs(jw_vel) * max_jw_vels
    vel_arm = jw_vels [:5]
    vel_wheels = jw_vels [5:]
```

To preserve the sign of the velocity, we multiply the max velocity by the absolute of the compared velocity and divide by itself to get the maximum velocity with desired sign.

- Feedforward only:

We tried using only the feedforward Control using Kp = 1 and Ki = 0 and got the error plot as shown below:



We observed that the feedforward-only control uses only desired end effector configurations and does not compensate for errors that accumulate during the motion of the end effector from one target position to another.

VideoLink:
https://drive.google.com/file/d/1-IB6yNNuVSuH9h8LGucdx4fIfGO8oqsm/view?usp=sharing

1. Best Task:

Type of Controller: Feedforward plus PI
Gain values: Kp = 20; Ki = 15

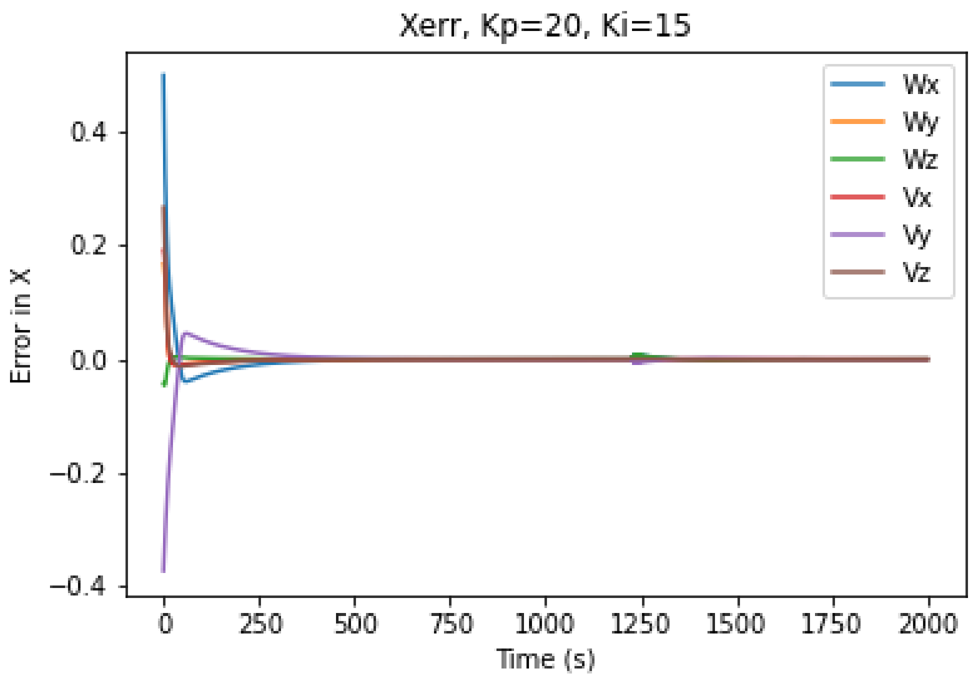
Initial configuration of the cube in world frame:

T_scube_initial = [1, 0, 0, 1]
[0, 1, 0, 0]
[0, 0, 1, 0.025]
[0, 0, 0, 1]

Final configuration of the cube in the world frame:

T_scube_final = [0, 1, 0, 0]
[-1, 0, 0, -1]
[0, 0, 1, 0.025]
[0, 0, 0, 1]

Error plot:



We observe that for Kp = 20 and Ki= 15, the error plot converged to zero very quickly and also it did not fluctuate as time progressed. This means that the Feedforward plus PI controller in this case was able to minimize and maintain zero error quickly and hence the robot performed the task successfully without any issues.

Video Link:
https://drive.google.com/file/d/1-3hn44V4JeLAt-WRyBF_iFQ0fbShSFpg/view?usp=sharing

2. Overshoot:
Type of Controller: Feedforward plus PI
Gain values: Kp = 2; Ki = 15

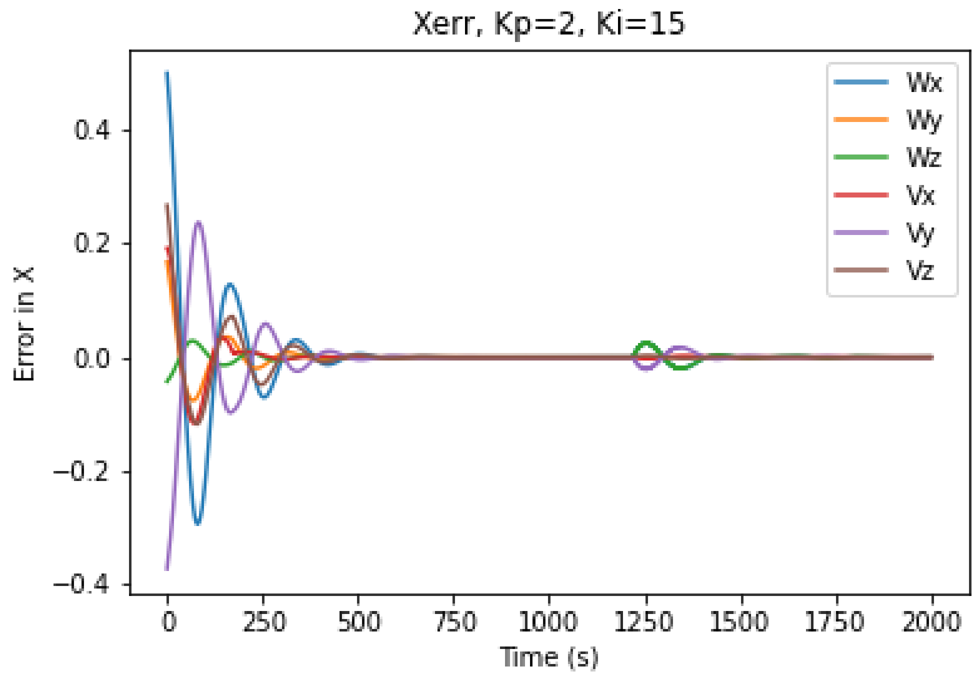
Initial configuration of the cube in world frame:

```
T_scube_initial = [1, 0, 0, 1 ]
                  [0, 1, 0, 0 ]
                  [0, 0, 1, 0.025 ]
                  [0, 0, 0, 1 ]
```

Final configuration of the cube in the world frame:

```
T_scube_final = [0, 1, 0, 0 ]
                [-1, 0, 0, -1 ]
                [0, 0, 1, 0.025 ]
                [0, 0, 0, 1 ]
```

Error plot:



For a feedforward plus PI controller, with a high integral gain, the system displays inconsistent behavior as the system overcompensates for the error caused and hence, leads to a negative error in the next step. Hence, we can see in the video that the robot overruns the initial location of the cube and then compensates with a negative error to bring the end effector back in the right place.

Video Link:
https://drive.google.com/file/d/1-0lDZ-19BGJf0QNze6IsiLOfqPRfXbo3/view?usp=sharing

3. New task:
Type of Controller: Feedforward plus PI
Gain values: Kp = 20; Ki = 15

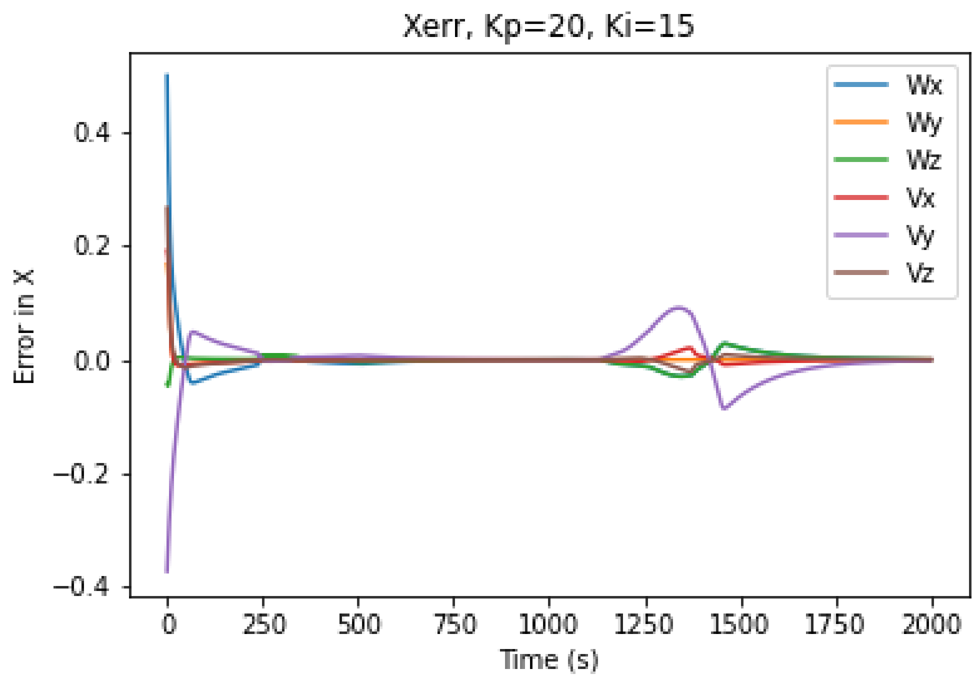
Initial configuration of the cube in world frame:

```
T_scube_initial = [1, 0, 0, 1 ]
                  [0, 1, 0, -1 ]
                  [0, 0, 1, 0.025 ]
                  [0, 0, 0, 1 ]
```

Final configuration of the cube in the world frame:

```
T_scube_final = [1, 0, 0, 1 ]
                [0, 1, 0, 0.5 ]
                [0, 0, 1, 0.025 ]
                [0, 0, 0, 1 ]
```

Error plot:



With the optimum proportional and integral gain, the new task of lifting the cube from the new position and putting it to a new final position is ensured smoothly although we see a little spike in the error values, the system compensates and the task is performed successfully as seen in the video.

Video Link:
https://drive.google.com/file/d/11BMadv1EHN3TvHBOOWdNOSR0uA6QXe7k/view?usp=sharing