Lab 3 Experiment Guide:
Motor Control

Objectives

- Our goal is to utilize ROBOTIS' Dynamixel to receive encoder data and implement the feedback position control. In other words, We aim to implement classic control theory related to motor control using C++ and examine it through experimentation.

- we will check whether the theoretically learned motor control contents and actual experimental results are consistent.

Problem Statement

Week 1)

Problem 3A. Development Environment Setup
- Setting up the development environment,
- File input/output for storing data and plotting,
- Receiving motor angle values
- Receiving motor velocities values

Problem 3B. Velocity Measurement (Numerical Differentiation with Low pass filter)

Problem 3C. Practicing position, velocity, and current control

Week 2)

Problem 3D. PD Position Controller (reference position: step input)

Problem 3E. PID Position Controller (reference position: step input)

Problem 3F. PID Position Controller (reference position: sinusoidal input)

Problem 3G. Understanding Frequency Response (With Chirp Signal)

Backgrounds

I added only the necessary background information for motor control implementation, excluding the motor control-related materials learned in the lecture.

The brief Introduction to Eigen Library for linear algebra computation

What is the Eigen Library?
The Eigen Library is an open-source C++ library for linear algebra operations, which includes matrix and vector operations, linear solvers, eigenvalue calculations, and more. But, we need a only matrix and vector operation. Therefore, We will briefly focus on the essential functions we need. If you want more information, please refer to the Eigen library website[1]

Why is the Eigen Library required?
Actually, the one motor control is not needed for the Eigen library, But in the Lab5 Robot Manipulator, we will use and control the multiple motors at the same time. In this case, the matrix operation is highly required.

Add the header like this.

#include <Eigen/Dense>

// How to declare the matrix A and vector b of linear algebra.

Eigen::MatrixXd A(3,2);

// 3: the number of the rows,
// 2: the number of the columns.

Eigen::VectorXd b(2);
// 2: the dimension of the vector b.

How to initialize the matrix or vector to zero after declaration.

A.setZero();
b.setZero();

How to assign values to elements of a matrix or vector.

The way 1)

```
Eigen::MatrixXd A(3,2);
Eigen::VectorXd b(2);
A << 2, 1,
       1, 3,
       0, 1;
b << 2, 1;
```

The way 2)

```
Eigen::MatrixXd A(3,2);
Eigen::VectorXd b(2);
A(0,0) = 2; A(0,1) = 1;
A(1,0) = 1; A(1,1) = 3;
A(2,0) = 0; A(2,1) = 1;

b(0) = 2;
b(1) = 1;
```

Important: **(The index of Eigen library starts from 0.)**

how to print the Eigen::MatrixXd or Eigen::VectorXd

std::cout << “A” << std::endl;
 	
std::cout << A << std::endl;
	result:
	A
	2 1
	1 3
	0 1

When you want to get the value of an element of the matrix A,

```
double a = A(0,0);
std::cout << “a: “ << a << std::endl;
```

result:
```
a: 2
```

When you want to get the value of an element of the vector b,

```
double a = b(1);
std::cout << “a: “ << a << std::endl;
```

result:
```
a: 1
```

Matrix Multiplication

std::cout << “A*b” << std::endl;
 	
std::cout << A*b << std::endl;
	result:
	
	A*b
	
	5
	
	5
	
	1

How to get the matrix size or vector's dimension

Matrix Case)

A.rows() : the number of the rows == 3

A.cols() : the number of the columns == 2

A.size(): the number of total elements == A.rows() * A.cols() == 6

Vector Case)

b.size(): the number of total elements == the dimension of the vector b == b.size() == 2

Eigen Library is a huge library for linear algebra operations. But I only covered the tiny part of the Eigen library for our lab procedure. If you want to find more information, please refer to [1].

The brief Introduction to the STL vector

STL Vector is also used in the code for the lab, but we will only cover the tiny parts where it is used. STL vector is a dynamic array that can be resized during runtime. It is a container class provided by the C++ Standard Template Library (STL) that allows you to store and manipulate a collection of elements in a contiguous memory block. It provides a wide range of member functions for adding, removing, and accessing elements, as well as iterators for iterating through the elements of the vector.

Example)

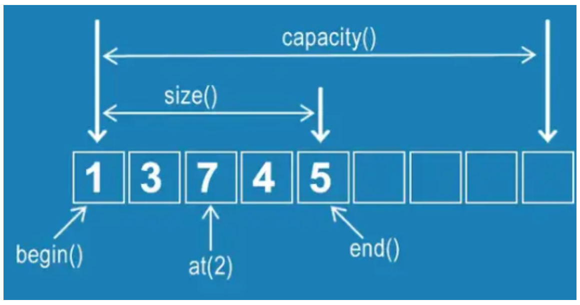
```
#include <iostream>
#include <vector>

int main() {
    std::vector<int> vec;
    vec.emplace_back(1);
    vec.emplace_back(3);
    vec.emplace_back(7);
    vec.emplace_back(4);
    vec.emplace_back(5);

    std::cout << "Vector contents: ";
    for (int i = 0; i < vec.size(); ++i) {
        std::cout << vec.at(i) << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

```
Vector contents: 1 3 7 4 5
```

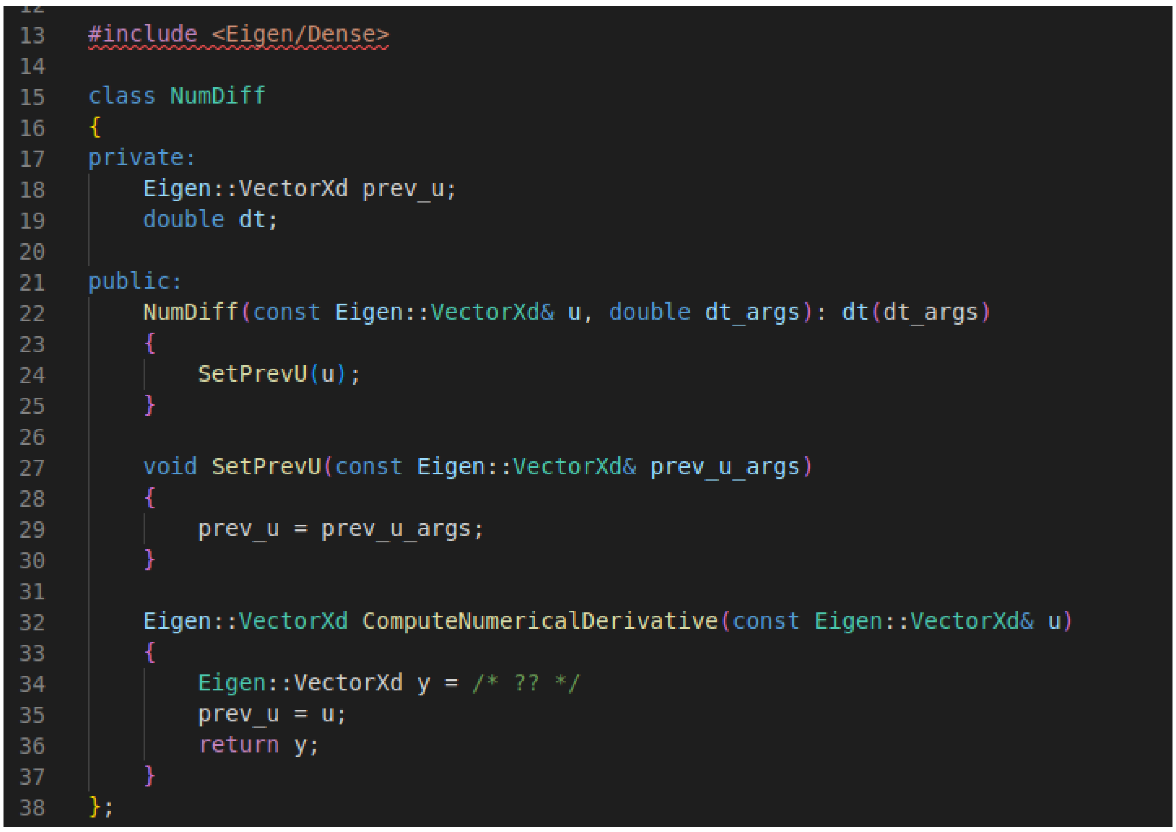


vec.size() returns 5
vec.begin() returns 1
vec.at(1) returns 3
vec.end() returns 5

The STL vector is implemented as a template class, so it can be used not only for int types but also for double, Eigen::VectorXd, any other class.

The brief Introduction to the Class of C++

Please refer to the following examples.



```
13  #include <Eigen/Dense>
14
15  class NumDiff
16  {
17  private:
18      Eigen::VectorXd prev_u;
19      double dt;
20
21  public:
22      NumDiff(const Eigen::VectorXd& u, double dt_args): dt(dt_args)
23      {
24          SetPrevU(u);
25      }
26
27      void SetPrevU(const Eigen::VectorXd& prev_u_args)
28      {
29          prev_u = prev_u_args;
30      }
31
32      Eigen::VectorXd ComputeNumericalDerivative(const Eigen::VectorXd& u)
33      {
34          Eigen::VectorXd y = /* ?? */
35          prev_u = u;
36          return y;
37      }
38  };
```

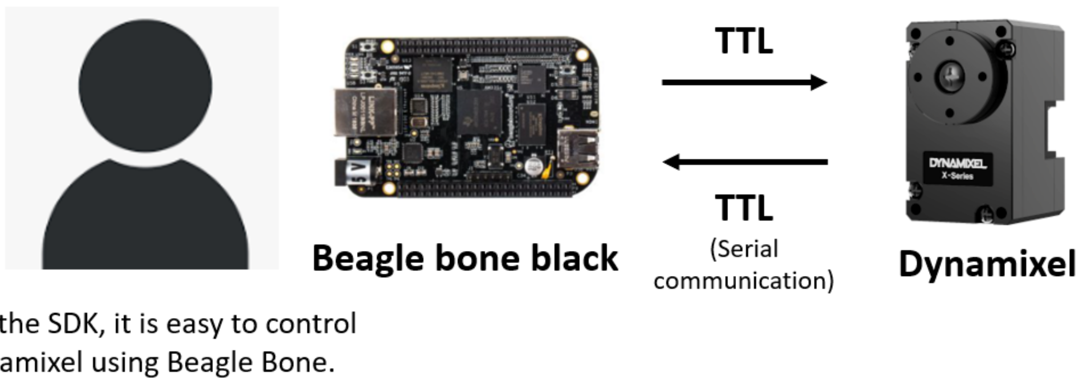
To implement numerical differentiation, we need to store previous values. Using a class to bundle data and member functions that operate on that data makes the code more readable and organized compared to using global variables. Therefore, we used classes to construct the skeleton code for numerical differentiation, low-pass filtering, and articulated systems, and in this lab.

You don't need to know everything about classes for this lab session. However, **you have to implement the core formula parts of the member functions** for numerical differentiation and low-pass filtering like line 34. So, having a basic understanding of classes will help you smoothly complete the lab procedure. If you are unfamiliar with classes, Please refer to this video for more information [2].

Dynamixel Introduction

Dynamixel is a brand of high-performance smart servos developed by ROBOTIS. It has the advantage of being compact as it combines a reduction gear, encoder, motor, and driver together. Internally, the Dynamixel provides position, velocity, and current control modes, and in this lab session, we will work with these control modes to control the motor and manipulator.
We used the XC330-T288-T model, and you can check its specifications in [3]. For more information on the Dynamixel XC330-T288-T, please refer to [3].

Dynamixel SDK


Beagle bone black
TTL
TTL
(Serial communication)
Dynamixel
With the SDK, it is easy to control Dynamixel using Beagle Bone.

To communicate with Dynamixel, a protocol that serves as a kind of agreement is necessary. To make it easier for Dynamixel users to communicate through the protocol, ROBOTIS provides the Dynamixel Software Development Kit, which allows users to use C++ functions to perform serial communication in accordance with the protocol. If you are interested, please refer to this GitHub site [4]

The introduction of the Articulated system class

The Dynamixel SDK is a program designed to control one motor. However, to control robot manipulators with multiple motors simultaneously, we need a new framework such as Articulate system class by expanding Dyanmixel SDK. Due to the limited time available during the lab sessions, we have already implemented the necessary functions for you to use. Therefore, we will introduce the essential functions required for controlling motors or manipulators.

Constructor: You can set up the degree of freedom, control mode, serial communication port name, various motor setup

```
ArticulatedSystem articulated_system(dof,
ArticulatedSystem::Mode::POSITION, "/dev/ttyUSB0", dynamixel_id_set,
position_resolution, motor_torque_constants);
```

The control mode includes POSITION, VELOCITY, and CURRENT modes.

```
articulated_system.GetJointAngle();
```
The GetJointAngle() function returns the **current motor angle** as Eigen::VectorXd data type, with a size of DOF that was set in the class constructor. The unit of position is **rad**, and it can be used **regardless of the control mode.**

```
articulated_system.GetJointVelocity();
```
The GetJointVelocity() function returns the **current motor velocity** as Eigen::VectorXd data type, with a size of DOF that was set in the class constructor. The unit of velocity is **rad/s**, and it can be used **regardless of the control mode.**

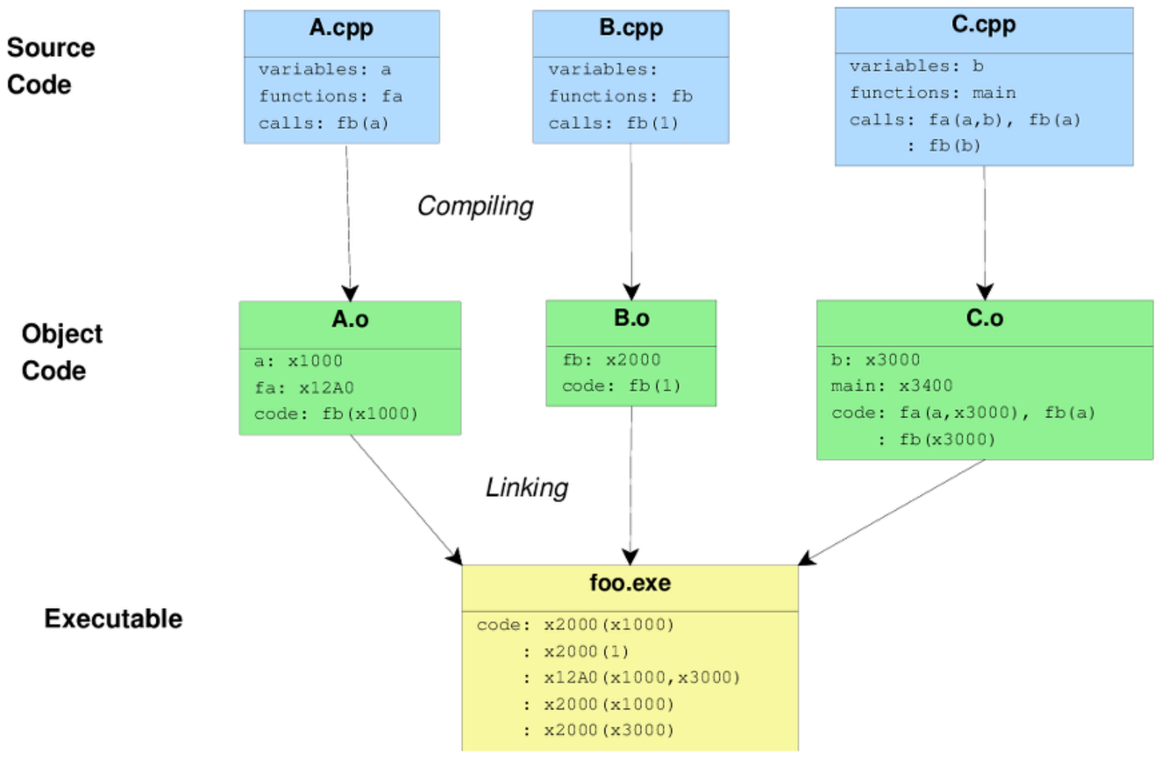
```
articulated_system.SetGoalPosition(targetQ);
```
The motor moves in accordance with the targetQ(=**target motor position**), which represents the desired motor position and is of type Eigen::VectorXd. The position is measured in **rad**. The size of the VectorXd must match the DOF that was set in the constructor. This function can **only** be used when the **POSITION mode** has been set in the constructor. The **built-in PID position controller** in the Dynamixel SDK is utilized.

```
articulated_system.SetGoalVelocity(targetQdot);
```
The motor moves in accordance with the targetQdot(=**target motor velocity**), which represents the desired motor velocity and is of type Eigen::VectorXd. The velocity is measured in **rad/s**. The size of the VectorXd must match the DOF that was set in the constructor. This function can **only** be used when the **VELOCITY mode** has been set in the constructor. The **built-in PI velocity controller** in the Dynamixel SDK is utilized.

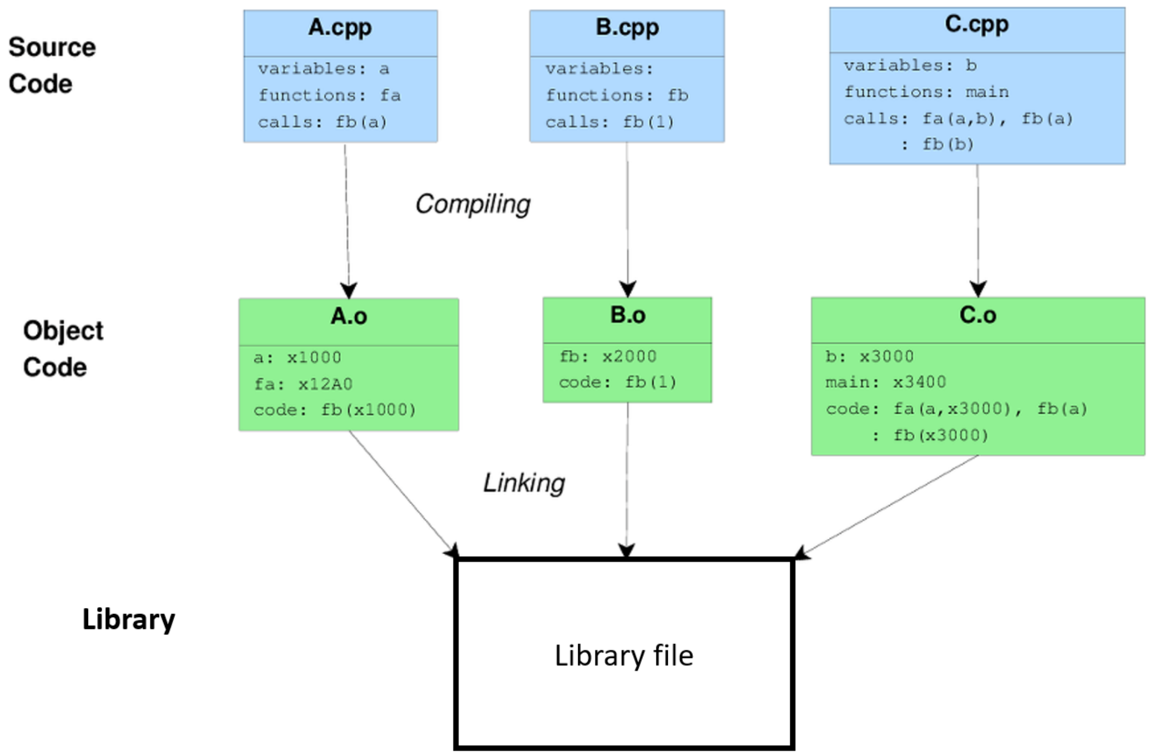
```
articulated_system.SetGoalTorque(targetTorque);
```
The SetGoalTorque() function **sets the torque** the motor produces. The motor moves according to the set targetTorque(=**target motor torque**). The input's type is Eigen::VectorXd. The torque is measured in **Nm**. The size of VectorXd must be the same as the previously set dof. and it is a function that can be operated **only** when the **CURRENT mode** is set in the constructor.

Why was it named CURRENT mode?
```
double target_current = target_torque(i)/motor_torque_constants[i];
```
If you open line 288 of the include/Controller/ArticulatedSystem.cpp, You will simply be able to confirm that the torque constant has been divided.
we cannot provide torque feedback because we don't have a torque sensor. Instead, we can only measure the current, which means that we're actually using current feedback control. Therefore, actually we use the current control not torque control.

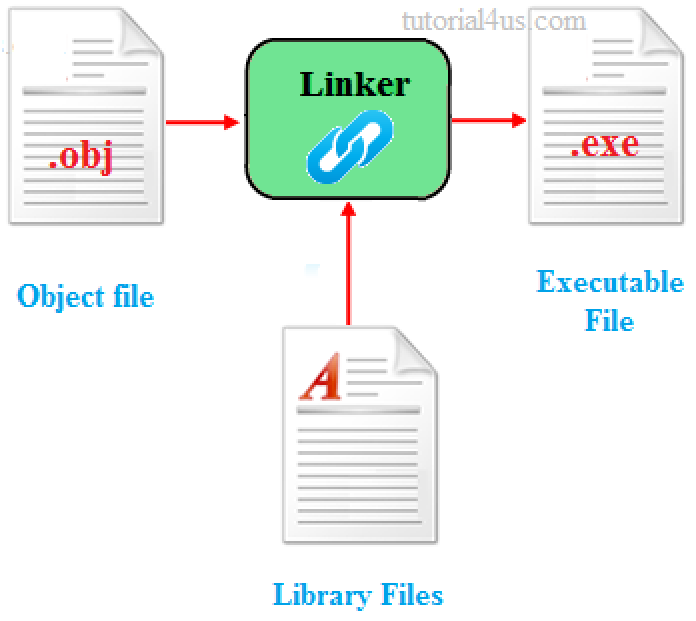
The brief introduction to C++ Language Compiling Process about Linker, Shared Library, Static Library



Until Lab2, we compiled multiple source codes into multiple object files and performed compilation through the linking process to create a single executable file.



From now on, it is also possible to create a library file instead of an executable file.

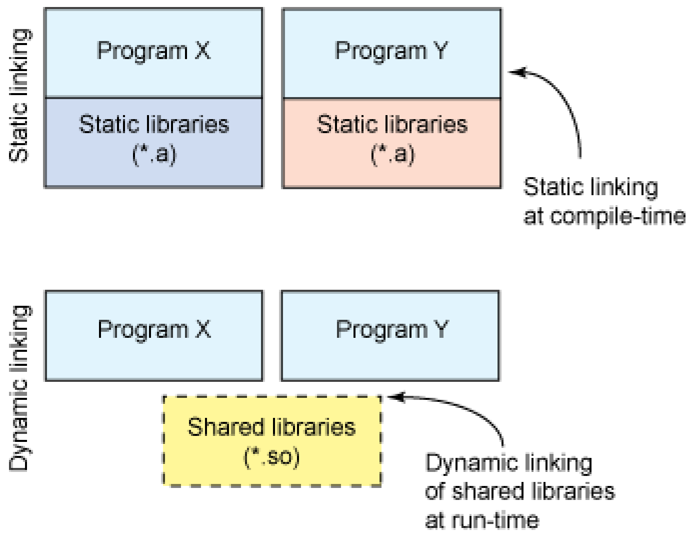


And then the linker can compile the object files and library files to create an executable file by linking them together.

The reasons for the emergence of libraries are as follows:

- Code reuse
- Realization of modularization of code
- Preventing technology leakage by not providing source code
- Reducing development time for users

Static Library vs Shared Library



A static library is a type of library that is compiled and linked directly into an executable file during the build process. All functions and code within the static library are included in the final executable, making it larger in size. Static libraries are typically used when the code will not change frequently.

On the other hand, a shared library is compiled separately and linked at runtime by the executable file. This means that multiple programs can use the same shared library, reducing the amount of memory required for the programs to run. Shared libraries are typically used when the code is expected to change frequently, as they can be updated independently without having to recompile the entire program.

Based on this, the important things to be covered in the lab session are as follows:
Dynamixel SDK can be compiled the shared library file. So, we will provide you with shared library files that are already compatible with the ARM-based architecture of the Beagle Bone. The important thing to note is that, due to the nature of shared libraries, they need to be transferred from the development PC to the BeagleBone. Additionally, during the lab session, we will also look into how to create static library files using CMake and our skeleton code.

Preparation

List of components

- IBM PC with Linux
- Beaglebone set: Beaglebone embedded board, USB cable, 1 ethernet cable, 4(or More) GB microSD card (with debian Linux image in it)
- Motor Control skeleton code.
- Dynamixel Starter kit
- motor control lecture note part 1 & 2

Lab Procedures

- See the documentation "EE405 Lab3 Procedure."

Final Report

Each student must write a final report for each experiment. Especially, if the content of the discussion is similar to that of other students, it will result in a deduction or a score of 0. The final report must include the following items:

- Experiment Results
- Discussion
- References (if you cited)

The essential discussion questions for Lab3 experiment are as follows, If you would like to conduct further experiments, feel free to do so and write the discussion on your report.

[Note!] When discussing, point to the resulting graph as much as possible

1. **Topic: Measurement of motor angular velocity using numerical differentiation with low pass filter**
    a. Plot the three following motor angular velocity results
        - Result of obtaining joint angular velocity through numerical differentiation
        - the result of applying the low pass filter
        - motor angular velocity obtained by using the GetJointVelocity() function in Articulated System Class.

    b. Plot the tendency of the results as the cutoff frequency changes. (For example, plot the filtering noise result when f = 1Hz, f = 10 Hz, f = 50 Hz, f = 100 Hz, f = 1000 Hz, etc)

    c. Discuss why we required the low pass filter when we use the numerical differentiation and Discuss the characteristics of the cutoff frequency, including following elements. (Related lab procedure: 26, 27)
        - The reason for amplified noise when performing numerical differentiation
        - Why we use the low pass filter.
        - The meaning of the cutoff frequency
        - the results of the plot the tendency of cutoff frequency
        - trade-off problem of cutoff frequency

[Note!] The second question is asking about an experiment where the input was a step desired position, not a sinusoidal desired position. Hence when you attach the figure to your report, please attach the figure regarding the step desired position.

2. **Topic: Characteristics of each PID gain of the position controller**
    a. Discuss the characteristics of P gain and D gain of the position PD controller including the following elements. **(Related lab procedure: 60 ~ 62, 65)**
        - The description about physical meaning of the P and D gain of the position controller from control thoery
        - Explanation of the effect of P gain on the steady-state error using the Final Value Theorem
        - Explanation of the effect of D gain on the system transient response
        - The tendency of actual experiment results when we P gain increase
        - the tendency of the experiment results for the system transient response as the D gain increases
        - (Optional) Characteristics of each P and D gain which you learned during actual tuning

    b. Discuss about the characteristics of i gain of the position PID Controller including **(Related lab procedure: 63, 64, 65)**
        - Explanation of the effect of I gain on the steady-state error using the Final Value Theorem
        - Experimental proof that The I gain makes the steady-state error goes to zero

3. **Topic: Frequency response**
    a. Using the meaning of example from the bode plot result on page 11 of the lecture_control_part2.pdf, Discuss the experimental result including how the amplitude and phase delay phenomena change as the frequency increases. **(Related lab procedure: 69, 70)**

Reference

[1] Eigen Library, https://eigen.tuxfamily.org/index.php?title=Main_Page
[2] C++ Tutorial for Beginners 20 - C++ Classes and Objects https://youtu.be/h1NIXUaDpmk
[3] Dynamixel XC330-T288-T https://emanual.robotis.com/docs/en/dxl/x/xc330-t288/
[4] Dynamixel SDK Github https://github.com/ROBOTIS-GIT/DynamixelSDK