# Lab Procedures:

## Motor control

## preparation

- 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

## Problem

Week 1)

Problem 3A. Development Environment Setup

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)

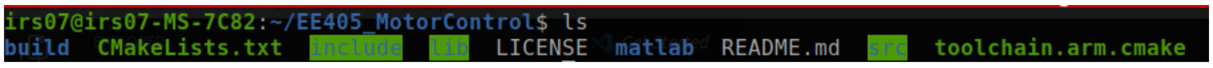
Problem 3F.  PID Position Controller (reference position: sinusoidal input)

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

## Problem 3A. Development Environment Setup

1. Turn on the development PC.

2. Please download the skeleton code from KLMS and either copy or move it to your home directory on the development PC, as shown in the following image:



```
irs07@irs07-MS-7C82:~/EE405_MotorControl$ ls
build  CMakeLists.txt  include  lib  LICENSE  matlab  README.md  src  toolchain.arm.cmake
```

3. Turn on the Beagle Bone Black Using SD Card.

4. Set up the NFS.

---

### 1. Configure NFS on the development PC

Remember IPs for PC and Bone, for example,

PC: either 192.168.7.1 or 192.168.6.1
Beaglebone: either 192.168.7.2 or 192.168.6.2

Edit /etc/exports to INCLUDE hosts allowed to connect on development PC (i.e., IP of Beaglebone).

```
$ sudo nano /etc/exports
```

```
# /etc/exports: the access control list for filesystems which may be exported
#        to NFS clients.        See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes      hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
# Example for NFSv4:
# /srv/nfs4 gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes gss/krb5i(rw,sync,no_subtree_check)
#

# nfs for Bone Ubuntu - Robot Manipulator
/home/<your computer's name>/EE405_MotorControl/build
192.168.7.2(rw,sync,no_root_squash,no_subtree_check)
```

Whenever we modify /etc/exports, we must run 'exportfs' to make the changes effective afterwards.

```
$ sudo exportfs -a
```

### 2. Start PC NFS server

To start the NFS server, you can run the following command at a terminal prompt:

```
$ sudo /etc/init.d/nfs-kernel-server start
```

### 3. Make the mount point on Beaglebone

```
$ mkdir ~/nfs_client
```

Note. There should be no files or subdirectories in the ~/nfs_client directory.

### 4. Start Beaglebone nfs client

command: sudo mount {pc_ip}:{pc_directory} {beaglebone_directory}

```
$ cd ~
$ sudo mount 192.168.7.1:/home/<your computer's name>/EE405_MotorControl/build ~/nfs_client
```

Check your PC IP!

---

5. To install the Dynamixel SDK Shared Library on Beagle Bone, follow these steps:

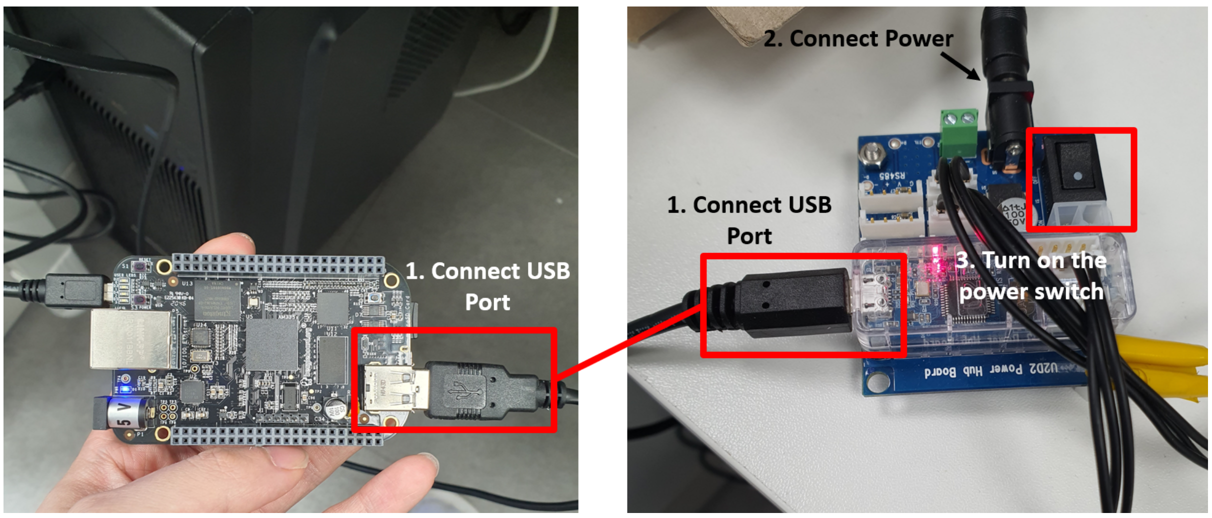
   **[On the Development PC]**

   ```
   $ cd ~/EE405_MotorControl/lib/dynamixel_sdk

   $ sudo scp libdxl_sbc_cpp.so debian@192.168.7.2:/home/debian
   ```

   **[On Beagle Bone Black]**

   ```
   debian@beaglebone$ sudo cp ~/libdxl_sbc_cpp.so /usr/lib
   ```

6. Connect between beagle bone and dynamixel U2D2, And Supply power.



7. Open the vscode on development PC.

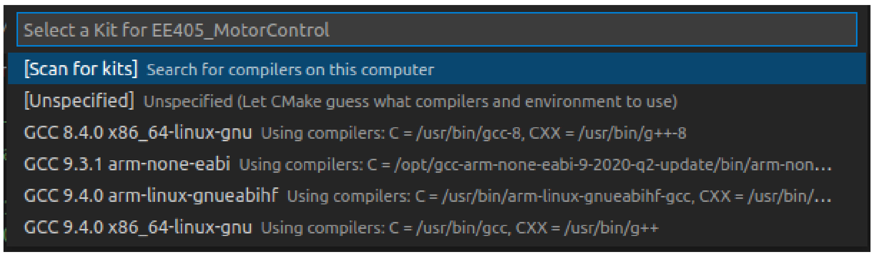
   ```
   $ cd ~/EE405_MotorControl
   $ code .
   ```
8. Open the CMakeLists.txt file and Please read the comments to understand their meaning and uncomment such as the following code.

```cmake
cmake_minimum_required(VERSION 3.10)
project(EE405_MotorControl LANGUAGES CXX)

add_compile_options(-std=c++17)

### find package: pre-installed library The CMake can automatically find
following installed package.
find_package(Eigen3 REQUIRED)
find_package(Threads REQUIRED)

### include_directories: We can set include directories, The CMake can look for
the header file now in this path.
include_directories (${EIGEN3_INCLUDE_DIRS}
                    include
                    include/dynamixel_sdk
                    src)

### link_directories: We can set link directories, The CMake can look for the
library file now in this path in order to link library and executable file.
link_directories(
    lib
    lib/dynamixel_sdk
)

### add_library: How to make a static library file using the following code. If
you want to create a shared library, change 'STATIC' to 'SHARED'. However, for
cross-compiling, it is not recommended to change to 'SHARED'.
add_library(EE405_robot_manipulator STATIC
include/Controller/ArticulatedSystem.cpp
include/FileIO/MatrixFileIO.cpp
include/Common/LowPassFilter.cpp
include/Common/NumDiff.cpp
)

### add_executable: Make executable file using following motor_control.cpp code.
add_executable(motor_control src/motor_control.cpp )

### target_link_libraries: link between exe file and library file.
## EE405_robot_manipulator: The static library which we make. please refer to
add_library().
## dxl_sbc_cpp: The dxl_x64_cpp shared library is from the Dynamixel SDK and it
allows motor rotation in the program.
target_link_libraries(motor_control EE405_robot_manipulator dxl_sbc_cpp  )
```

```cmake
cmake_minimum_required(VERSION 3.10)
```

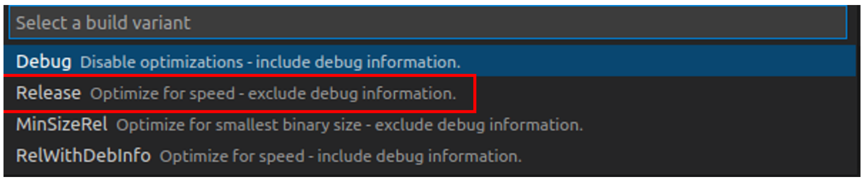
**Please change the cmake_minimum_required from Version 3.13 to 3.10.**
Because The Cmake version on the computers of N5 2355 is 3.10.

9. Build the project using vs code button.



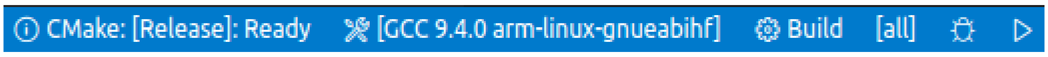
```
Select a Kit for EE405_MotorControl
[Scan for kits] Search for compilers on this computer
[Unspecified] Unspecified (Let CMake guess what compilers and environment to use)
GCC 8.4.0 x86_64-linux-gnu Using compilers: C = /usr/bin/gcc-8, CXX = /usr/bin/g++-8
GCC 9.3.1 arm-none-eabi Using compilers: C = /opt/gcc-arm-none-eabi-9-2020-q2-update/bin/arm-non...
GCC 9.4.0 arm-linux-gnueabihf Using compilers: C = /usr/bin/arm-linux-gnueabihf-gcc, CXX = /usr/bin/...
GCC 9.4.0 x86_64-linux-gnu Using compilers: C = /usr/bin/gcc, CXX = /usr/bin/g++
```

Set the compiler as GCC **arm-linux-gnueabihf.**



```
Select a build variant
Debug Disable optimizations - include debug information.
Release Optimize for speed - exclude debug information.
MinSizeRel Optimize for smallest binary size - exclude debug information.
RelWithDebInfo Optimize for speed - include debug information.
```

Set the compile mode as **Release** mode. (Do not set Debug mode.)



```
CMake: [Release]: Ready   [GCC 9.4.0 arm-linux-gnueabihf]   Build   [all]
```

Now, you can build the project by either clicking on the build button or pressing the F7 shortcut key.

10. Check the build directory on the PC,

$ **ls ~/EE405_MotorControl/build**

build.ninja     cmake_install.cmake          **motor_control**
CMakeCache.txt  compile_commands.json
CMakeFiles      **libEE405_robot_manipulator.a**  rules.ninja

You can check the static library(=libEE405_robot_manipulator.a) and executable file(=motor control).
And you can also check the object files in the following directory.

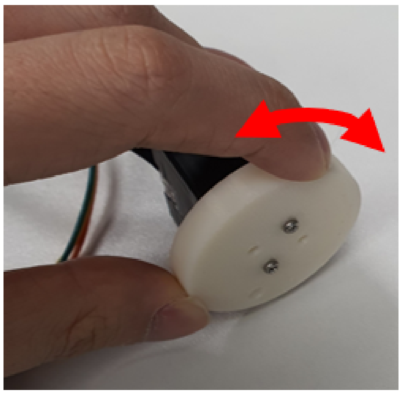
~/EE405_MotorControl/build/CMakeFiles/EE405_robot_manipulator.dir
~/EE405_MotorControl/build/CMakeFiles/motor_control.dir

11. Run the motor control on the Beagle Bone. (This is a process of rotating the motor by hand without any control input applied, measuring the motor angle and velocity, and checking whether the plot is successfully displayed.)

debian@beaglebone:~$ **cd ~/nfs_client**
debian@beaglebone:~/nfs_client$ **./motor_control**

**The result of a successful execution of the program)**
Succeeded to open the port!
Succeeded to change the baudrate!
Succeeded operating mode to current mode.
Succeeded enabling DYNAMIXEL Torque.

12. Rotate the motor by hand like in the following picture.



13. Turn off the code by pressing Ctrl + C on the terminal window. (From now on, when you're done experimenting, press Ctrl + C directly once even if there are no instructions.)

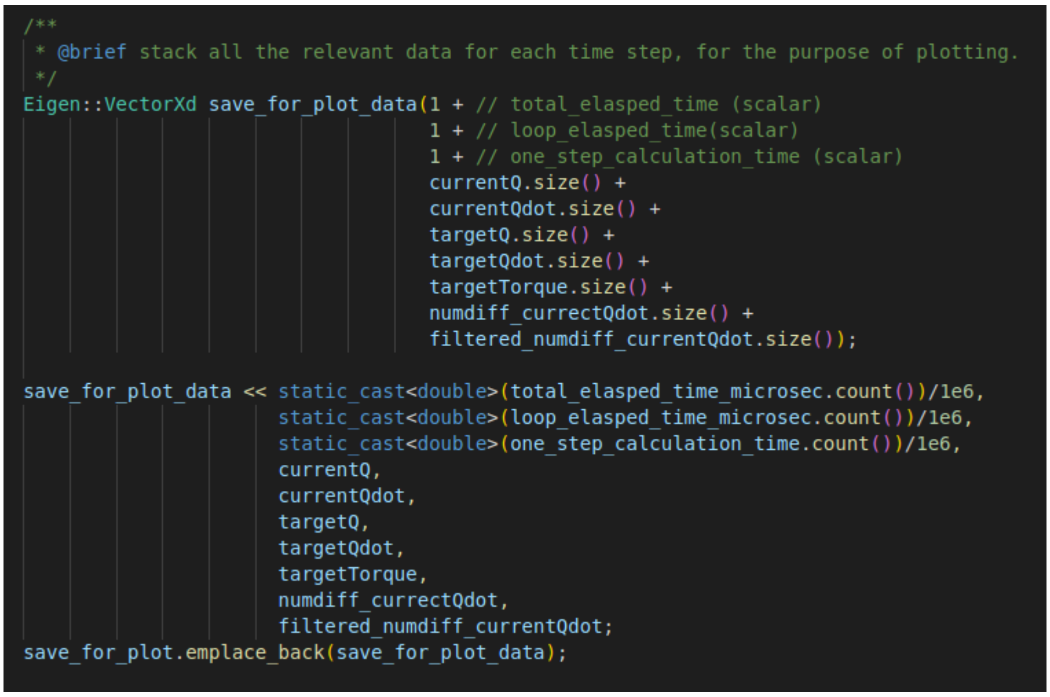
**If you want to terminate the process, Press Ctrl + C once.**

14. Check the build directory on the PC again

$ **ls ~/EE405_MotorControl/build**

```
build.ninja     cmake_install.cmake          motor_control
CMakeCache.txt  compile_commands.json        recorded_data.csv
CMakeFiles      libEE405_robot_manipulator.a  rules.ninja
```

**Now, The motor angle, velocity, and control loop period time were recorded in the 'recorded_data.csv' file.** If you want to know which data was saved, please refer to line 210 of the 'motor_control.cpp' file.



```
/**
 * @brief stack all the relevant data for each time step, for the purpose of plotting.
 */
Eigen::VectorXd save_for_plot_data(1 + // total_elasped_time (scalar)
                                   1 + // loop_elasped_time(scalar)
                                   1 + // one_step_calculation_time (scalar)
                                   currentQ.size() +
                                   currentQdot.size() +
                                   targetQ.size() +
                                   targetQdot.size() +
                                   targetTorque.size() +
                                   numdiff_currectQdot.size() +
                                   filtered_numdiff_currentQdot.size());

save_for_plot_data << static_cast<double>(total_elasped_time_microsec.count())/1e6,
                      static_cast<double>(loop_elasped_time_microsec.count())/1e6,
                      static_cast<double>(one_step_calculation_time.count())/1e6,
                      currentQ,
                      currentQdot,
                      targetQ,
                      targetQdot,
                      targetTorque,
                      numdiff_currectQdot,
                      filtered_numdiff_currentQdot;
save_for_plot.emplace_back(save_for_plot_data);
```

15. Rename the 'recorded_data.csv' file to a name of your choice in order to avoid overwriting by our code.

    For example, from “recorded_data.csv” to “recorded_data_test.csv”

16. Open the matlab

    ```
    $ cd ~/EE405_MotorControl/matlab
    $ matlab
    ```

17. Open the plot_data.m on the matlab

18. Enter the name of the CSV file containing the data you want to check here.

    For example,
    Before: `csv_title = "recorded_data";`
    After: `csv_title = "recorded_data_test";`

19. Now, Run the matlab “plot_data.m” code.
20. See figure number 1 on the matlab

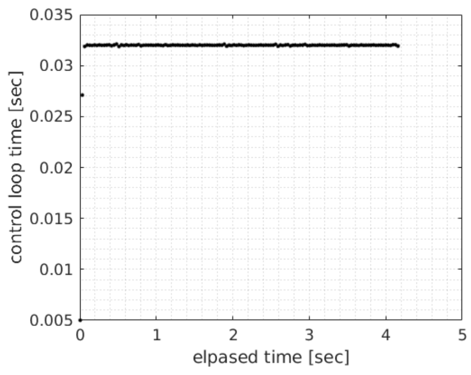


**Figure 1**

As you can see in Figure 1, Each step takes 0.032 seconds. in other words, the control frequency is not 200Hz. I will introduce an important command as follows.

```
debian@beaglebone:~$ cat /sys/bus/usb-serial/devices/ttyUSB0/latency_timer
16
debian@beaglebone:~$ echo 1 | sudo tee
/sys/bus/usb-serial/devices/ttyUSB0/latency_timer
1
```

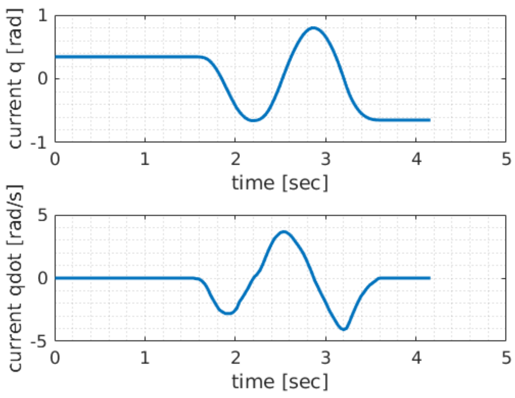
**After executing this command, is the 200Hz control loop frequency properly maintained?**

When sending and receiving commands via serial communication, it took 32ms, which is a multiple of latency time(=16ms). Therefore, it needs to be set to 1ms to satisfy the 200 Hz control loop frequency.

## [Note!] IMPORTANT Things

**You need to enter this setting latency command every time you connect the serial communication cable between the Dynamixel U2D2 and BeagleBone.**

21. See figure 2



you can also check the recorded motor angle and velocity which you rotated.

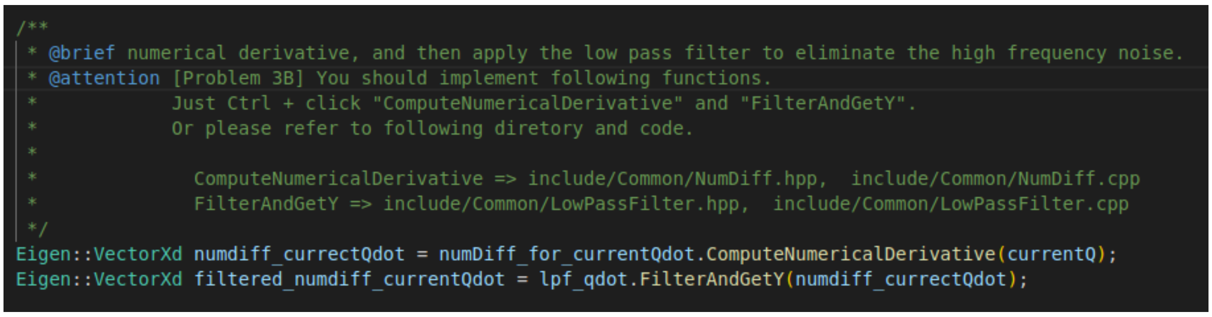
## [Note!] IMPORTANT Things

**Take all experiment results(=csv files) with you for future report writing. When writing a report, it is very important to attach the result graph**

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

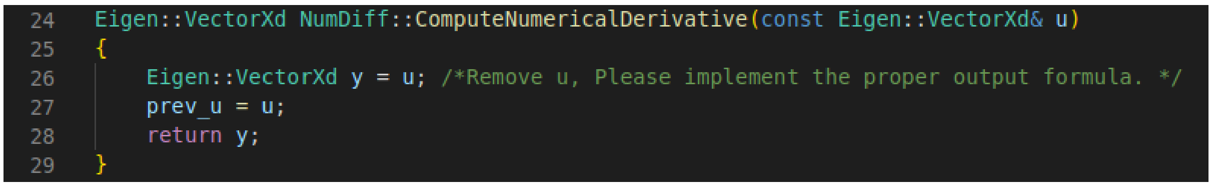
22. Open the src/motor_control.cpp using Visual Studio Code on the development PC.

23. Now, you can check Problem 3B at line 140 of motor_control.cpp, please note the “@attention”.



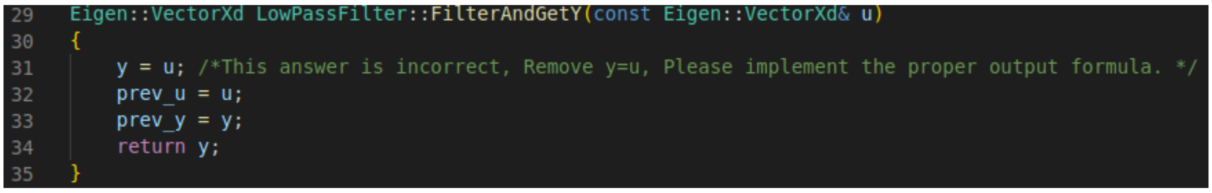
```
/**
 * @brief numerical derivative, and then apply the low pass filter to eliminate the high frequency noise.
 * @attention [Problem 3B] You should implement following functions.
 *            Just Ctrl + click "ComputeNumericalDerivative" and "FilterAndGetY".
 *            Or please refer to following diretory and code.
 *
 *              ComputeNumericalDerivative => include/Common/NumDiff.hpp,  include/Common/NumDiff.cpp
 *              FilterAndGetY => include/Common/LowPassFilter.hpp,  include/Common/LowPassFilter.cpp
 */
Eigen::VectorXd numdiff_currectQdot = numDiff_for_currentQdot.ComputeNumericalDerivative(currentQ);
Eigen::VectorXd filtered_numdiff_currentQdot = lpf_qdot.FilterAndGetY(numdiff_currectQdot);
```

24. Remove the y=u, Implement the proper output of numerical differentiation using slide number 7 of lecture_control_part1.pdf.
    **member variables)**
    **dt :=** $\Delta t$**,   prev_u :=** $u_{k-1}$ **,   u :=** $u_k$



```
Eigen::VectorXd NumDiff::ComputeNumericalDerivative(const Eigen::VectorXd& u)
{
    Eigen::VectorXd y = u; /*Remove u, Please implement the proper output formula. */
    prev_u = u;
    return y;
}
```

25. Remove the y=u, Implement the proper formula of low pass filter using slide number 10 of lecture_control_part1.pdf.
    **member variables)**
    **T :=** $\Delta t$ **(sampling period), L := cutoff frequency (rad/s)**
    **prev_u :=** $u_{k-1}$ **, u :=** $u_k$
    **prev_y :=** $y_{k-1}$**, y :=** $y_k$**,**



```
Eigen::VectorXd LowPassFilter::FilterAndGetY(const Eigen::VectorXd& u)
{
    y = u; /*This answer is incorrect, Remove y=u, Please implement the proper output formula. */
    prev_u = u;
    prev_y = y;
    return y;
}
```

26. Now, Build using Visual Studio Code (press F7) and run the program on the BeagleBone. (I will refer to this instruction as "**Build and run**" from now on.)

    and then rotate the motor by hand like in procedure 12.

    And, Discuss the results of the numerical differentiation and describe some of the characteristics Including the effect of noise amplification in numerical differentiation.

**27. Plot the three following motor angular velocity results to compare**

- 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.

```
    LowPassFilter lpf_qdot(currentQdot, 5, dt);
```

At the 91st line of the motor_control.cpp, The 5 rad/s means the cutoff frequency. You can change the cutoff frequency.

**Plot the tendency of cutoff frequency changes.** (For example, plot the filtering noise result when f = 1 rad/s, f = 10 rad/s, f = 50 rad/s, f = 100 rad/s, f = 1000 rad/s, etc)

find the appropriate cut-off frequency value, and discuss the characteristics of the cut-off frequency.

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

**28. Test Position Control Mode**

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

You can modify the control modes, such as position control mode, velocity mode, and current mode, by changing the code at line 78 of the motor_control.cpp file.

Change the from `ArticulatedSystem::Mode::CURRENT` to `ArticulatedSystem::Mode::POSITION`

29. Uncomment and Set the target motor angle to 1 rad and the target motor velocity to 0 rad/s.

```
    ///////////// Constant target /////////////
    targetQ << 1;// M_PI;
    targetQdot << 0;
    ///////////// Constant target END /////////////
```

30. Uncomment and Set the Goal position to “targetQ”.

```
    ///////////////// POSITION CONTROL //////////////////////
    articulated_system.SetGoalPosition(targetQ);
    ///////////////// POSITION CONTROL END //////////////////////
```

31. **Build and run (Be careful as the motor moves)**

32. Rotate the Cylinder attached to the motor by hand carefully, being mindful of safety. Does the motor rotate easily when a torque is applied by hand? Additionally, if you want, Try moving the motor to angles other than 1 rad by adjusting the target motor angle(=targetQ).

33. Please comment out the SetGoalPosition line.

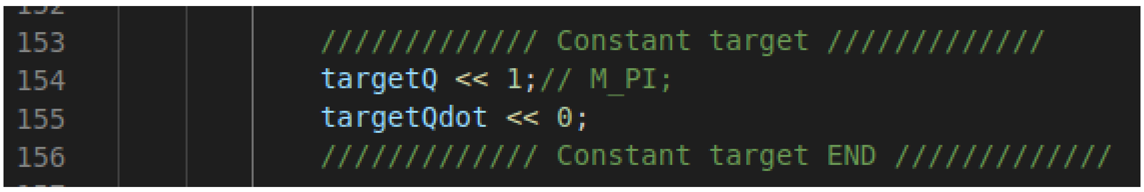
```
///////////////// POSITION CONTROL //////////////////////
// articulated_system.SetGoalPosition(targetQ);
///////////////// POSITION CONTROL END //////////////////////
```

**34. Test Velocity Control Mode**

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

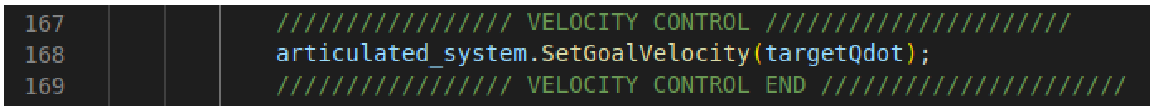
Change the from `ArticulatedSystem::Mode::POSITION` to `ArticulatedSystem::Mode::VELOCITY`

35. Double-check that the target motor velocity is set to 0 rad/s.



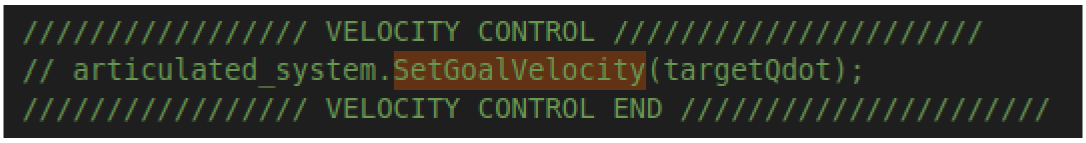
```
            ///////////// Constant target /////////////
            targetQ << 1;// M_PI;
            targetQdot << 0;
            ///////////// Constant target END /////////////
```

36. Uncomment this SetGoalVelocity line.



```
            //////////////// VELOCITY CONTROL /////////////////////
            articulated_system.SetGoalVelocity(targetQdot);
            //////////////// VELOCITY CONTROL END /////////////////////
```

37. **Build and run**
38. Rotate the cylinder by hand carefully, being mindful of safety. When using velocity control, can the motor rotate easily even when a torque is applied by hand?
39. comment out this SetGoalVelocity line.



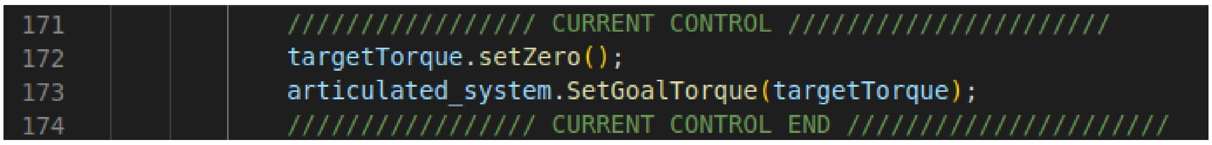
```
//////////////// VELOCITY CONTROL /////////////////////
// articulated_system.SetGoalVelocity(targetQdot);
//////////////// VELOCITY CONTROL END /////////////////////
```

**40. Test Current Control Mode**

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

Change the from `ArticulatedSystem::Mode::VELOCITY` to `ArticulatedSystem::Mode::CURRENT`

41. Uncomment the set zero torque line.

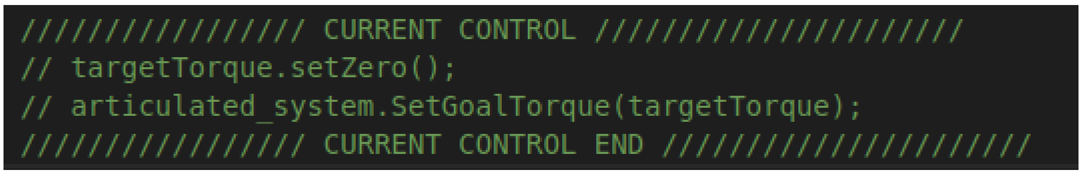


```
            //////////////// CURRENT CONTROL /////////////////////
            targetTorque.setZero();
            articulated_system.SetGoalTorque(targetTorque);
            //////////////// CURRENT CONTROL END /////////////////////
```

**42. Build and run**

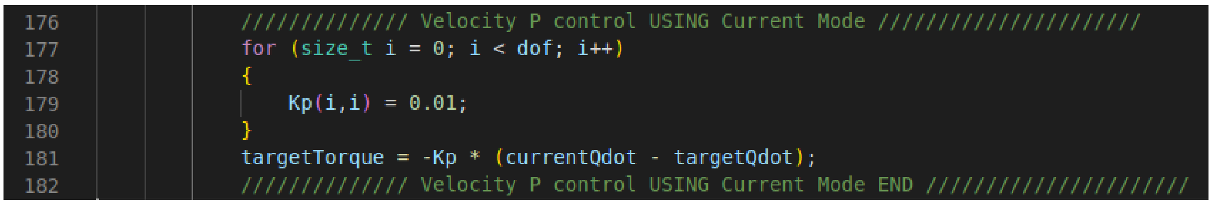
**43.** Rotate the cylinder by hand carefully, being mindful of safety. What are some characteristics of zero current control? Is there a difference from when the motor is off? (If you want to compare, turn off the power switch of Lab3 procedure 6th step, and rotate the motor, Check the motor LED and feel the additional resisting torque. What do the blinking of the motor LED and additional torque mean?)

**44.** comment out this “Set Torque Zero” line.



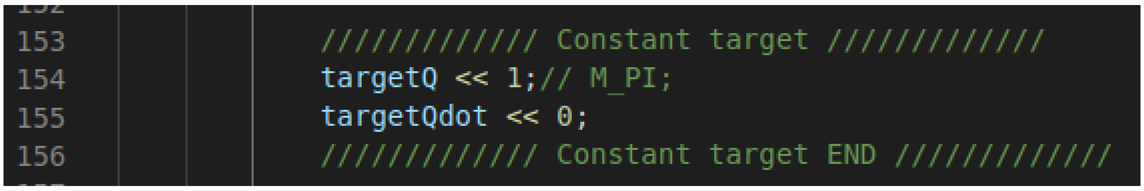
```
//////////////// CURRENT CONTROL /////////////////////
// targetTorque.setZero();
// articulated_system.SetGoalTorque(targetTorque);
//////////////// CURRENT CONTROL END /////////////////////
```

45. Uncommented the motor velocity P Control line.



```
///////////// Velocity P control USING Current Mode /////////////////////
for (size_t i = 0; i < dof; i++)
{
    Kp(i,i) = 0.01;
}
targetTorque = -Kp * (currentQdot - targetQdot);
///////////// Velocity P control USING Current Mode END /////////////////////
```

46. Check again the target motor velocity is 0 rad/s.



```
///////////// Constant target /////////////
targetQ << 1;// M_PI;
targetQdot << 0;
///////////// Constant target END /////////////
```

**47. Build and run**

**48.** Rotate the cylinder by hand carefully, being mindful of safety. Does the motor move easily when a disturbance is given when we use the velocity P control?

49. What happens to the motor as the velocity's P gain increases? And what is the physical meaning of the P gain in this case? Increase the P gain from 0.01 to your desired value and feel the damping torque increase.

**50.** The velocity control on lab procedures from 33 to 38, This controller is a PI controller [1]. Is there a difference between velocity PI control and velocity P control? If so, why did the difference occur?

**51.** Save all your CSV data, and take it for your report writing.

## Motor Control **Week 1 End**

---

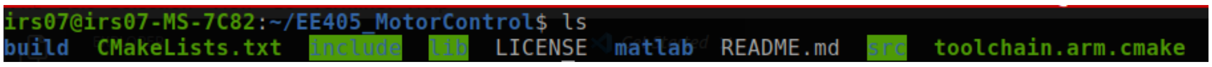
## Motor Control **Week 2 Start**

### [Note!] IMPORTANT Things

**Take all experiment results(=csv files) with you for future report writing.**

52. Turn on the development PC.

53. Move the skeleton code for motor control to your home directory on the development PC.

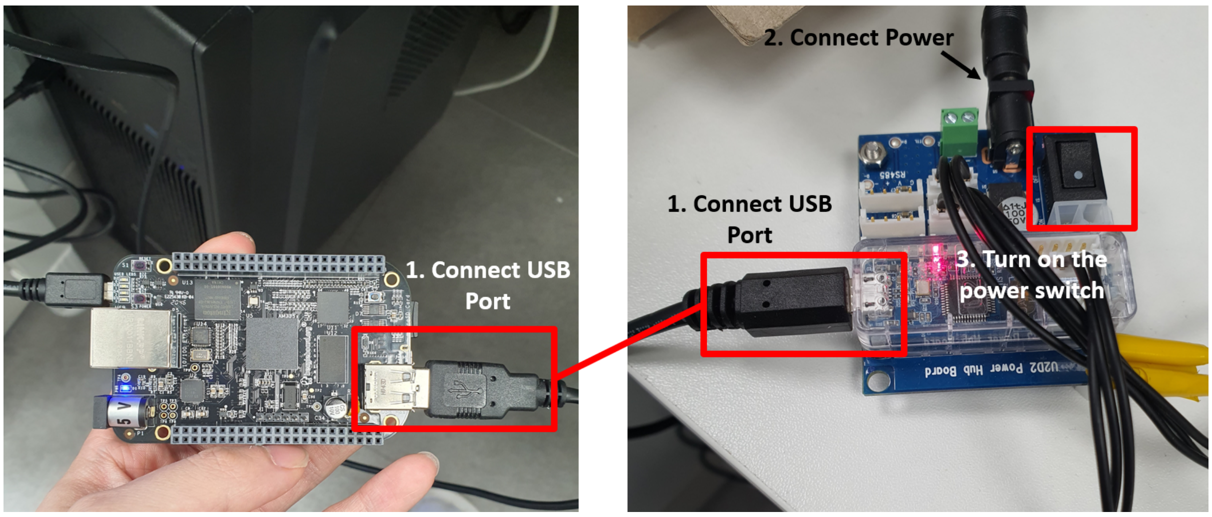


```
irs07@irs07-MS-7C82:~/EE405_MotorControl$ ls
build  CMakeLists.txt  include  lib  LICENSE  matlab  README.md  src  toolchain.arm.cmake
```

54. Turn on the Beagle Bone Black Using your SD Card.

55. Set up the NFS.

56. Connect between beagle bone and dynamixel U2D2.



57. Change the latency timer from 16 ms to 1 ms on beagle bone black.

debian@beaglebone:~$ **cat /sys/bus/usb-serial/devices/ttyUSB0/latency_timer**
16
debian@beaglebone:~$ **echo 1 | sudo tee /sys/bus/usb-serial/devices/ttyUSB0/latency_timer**
1

## [Note!] IMPORTANT Things

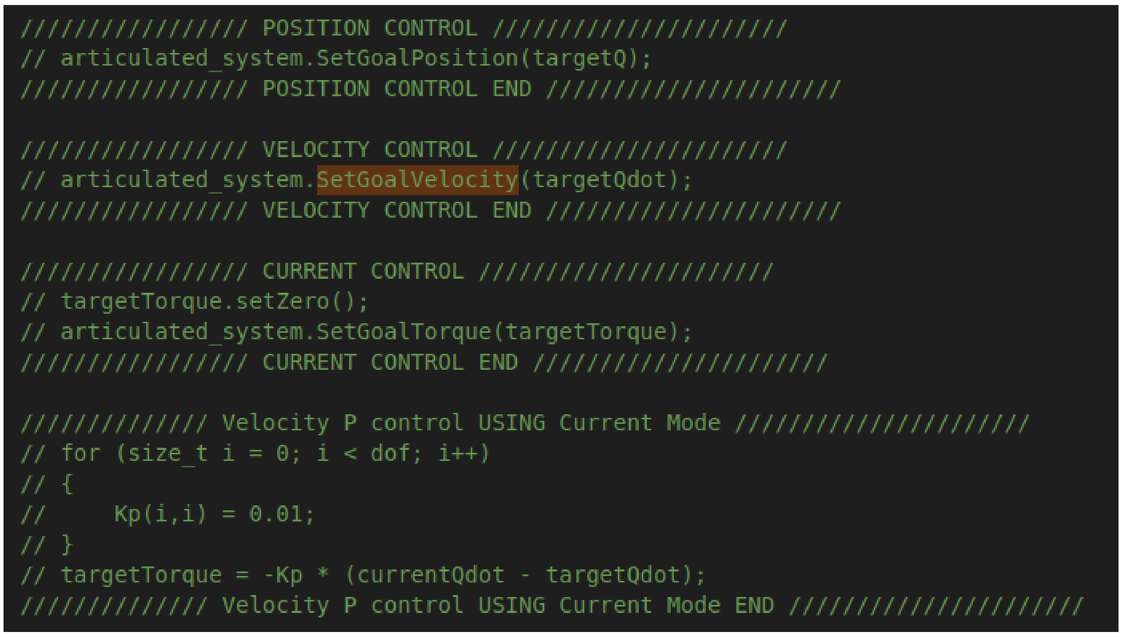
**You need to enter this command every time you connect the serial communication cable between the Dynamixel U2D2 and BeagleBone.**

58. Change the **Current Control Mode**

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

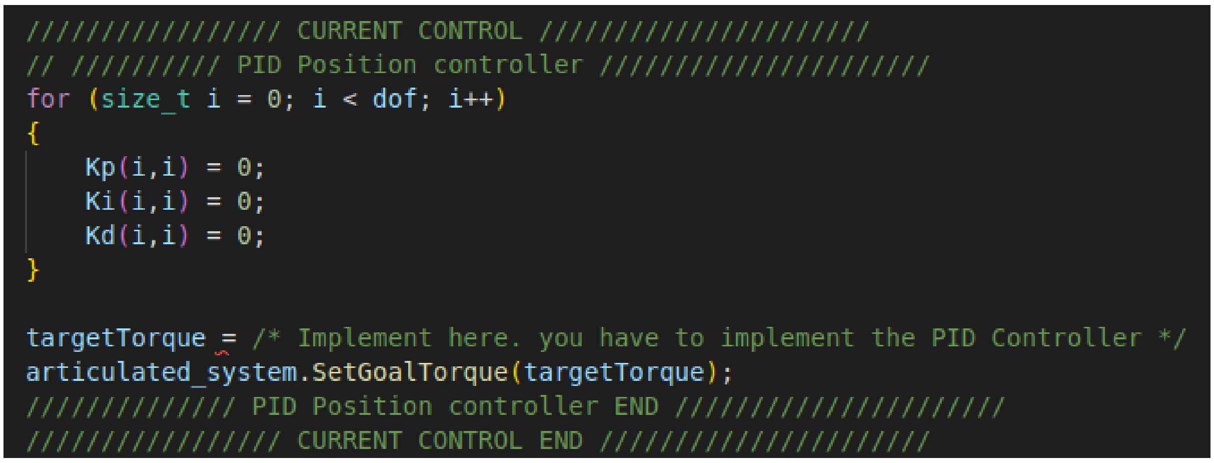
Change the to `ArticulatedSystem::Mode::CURRENT`

59. Check that all controller is commented on.



```
///////////////// POSITION CONTROL //////////////////////
// articulated_system.SetGoalPosition(targetQ);
///////////////// POSITION CONTROL END /////////////////////

///////////////// VELOCITY CONTROL //////////////////////
// articulated_system.SetGoalVelocity(targetQdot);
///////////////// VELOCITY CONTROL END /////////////////////

///////////////// CURRENT CONTROL //////////////////////
// targetTorque.setZero();
// articulated_system.SetGoalTorque(targetTorque);
///////////////// CURRENT CONTROL END /////////////////////

////////////// Velocity P control USING Current Mode /////////////////////
// for (size_t i = 0; i < dof; i++)
// {
//     Kp(i,i) = 0.01;
// }
// targetTorque = -Kp * (currentQdot - targetQdot);
////////////// Velocity P control USING Current Mode END /////////////////////
```

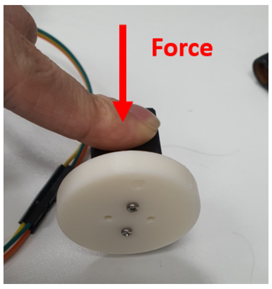
60. Uncomment the following code and **implement the PID Controller** using the target torque. **You can use the ROBOTIS's motor velocity using the GetJointVelocity() function.**



```
///////////////// CURRENT CONTROL //////////////////////
// ////////// PID Position controller /////////////////////
for (size_t i = 0; i < dof; i++)
{
    Kp(i,i) = 0;
    Ki(i,i) = 0;
    Kd(i,i) = 0;
}

targetTorque = /* Implement here. you have to implement the PID Controller */
articulated_system.SetGoalTorque(targetTorque);
////////////// PID Position controller END /////////////////////
///////////////// CURRENT CONTROL END /////////////////////
```

61. Let's implement a PD controller with I gain set to 0 in the implemented controller. Set the target joint angle to an arbitrary constant value (e.g. 1 rad, 2 rad, ... etc)

62. Build and run, Press firmly with your hand to keep the motor from moving.



Please refer to page 8 of lecture_control_part2.pdf.

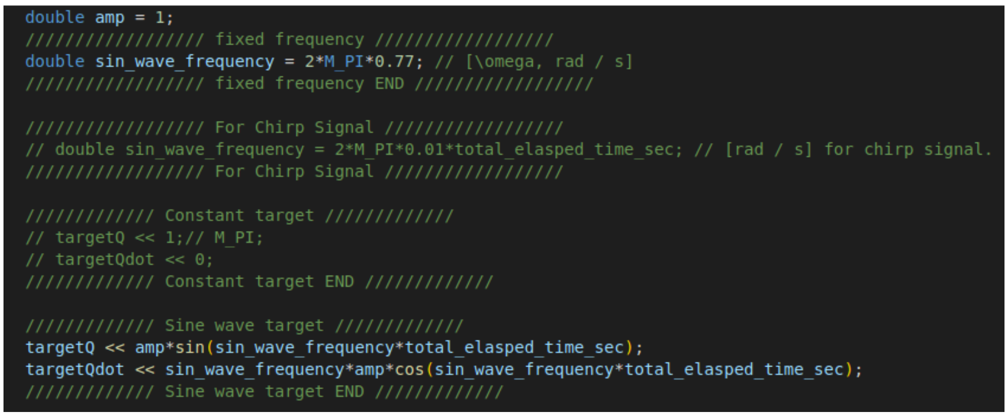
- Tune the P and D gain on a real motor system, taking into consideration the safety precautions.
- Plot the tendency of actual experiment results when we P gain increase to explain the effect of P gain on the steady-state error.
- Plot the tendency of the experiment results for the system transient response as the D gain.
- Discuss the characteristics of P gain and D gain based on the theoretical background and the experimental result.

63. Let's check the result of the Position PD Controller once again. Is there a steady-state error?
64. Let's add the I gain now. When the I gain is a little added, does the steady-state error disappear due to the final value theorem?

    Plot the experimental proof that the I gain makes the steady-state error goes to zero.

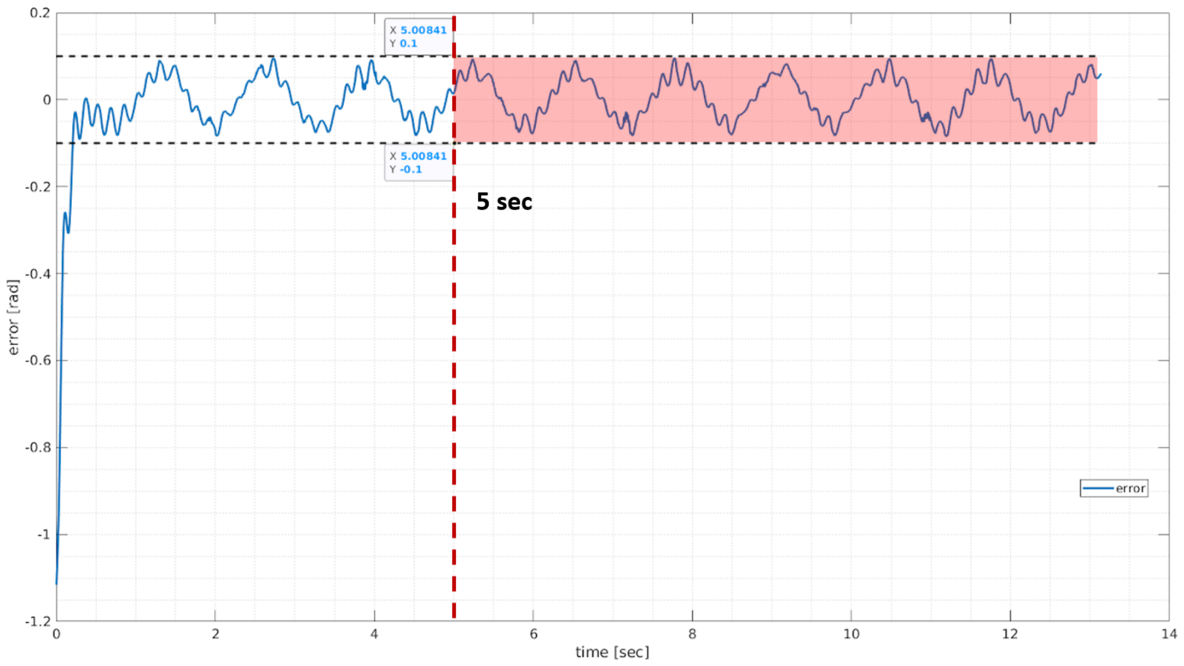
65. If you have learned any additional information about P, I, D tuning through experiments, describe it in the report.

66. Comment out the Constant target, and uncomment the fixed frequency part and sinusoidal target motor angle(=targetQ), velocity(=targetQdot)
    sine wave amp : 1 rad
    sine wave frequency : 0.77 Hz

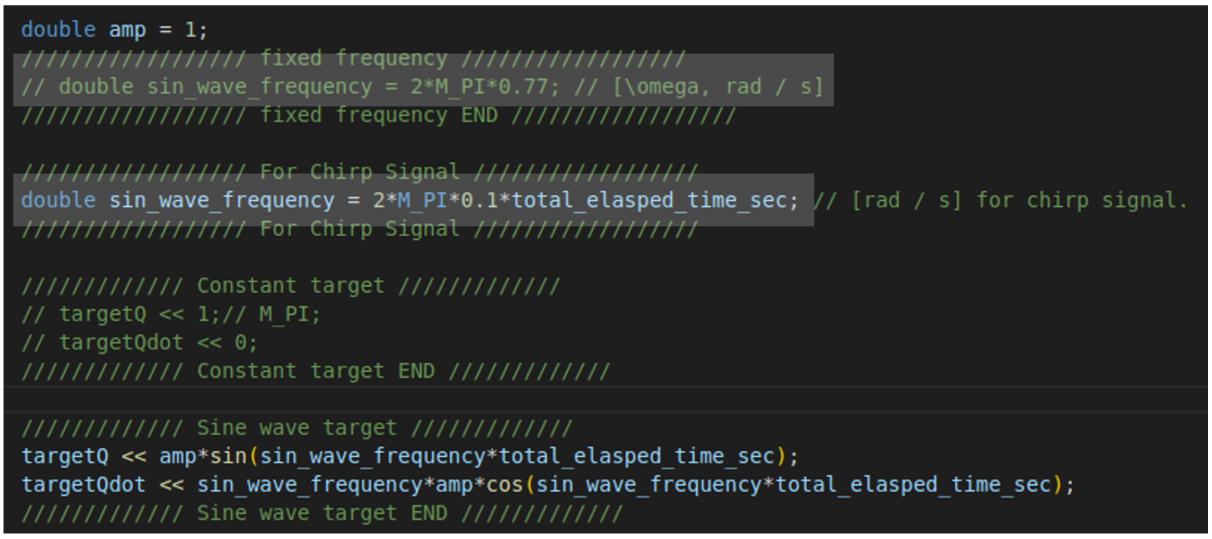


```
double amp = 1;
////////////////// fixed frequency //////////////////
double sin_wave_frequency = 2*M_PI*0.77; // [\omega, rad / s]
////////////////// fixed frequency END //////////////////

////////////////// For Chirp Signal //////////////////
// double sin_wave_frequency = 2*M_PI*0.01*total_elasped_time_sec; // [rad / s] for chirp signal.
////////////////// For Chirp Signal //////////////////

///////////// Constant target /////////////
// targetQ << 1;// M_PI;
// targetQdot << 0;
///////////// Constant target END /////////////

///////////// Sine wave target /////////////
targetQ << amp*sin(sin_wave_frequency*total_elasped_time_sec);
targetQdot << sin_wave_frequency*amp*cos(sin_wave_frequency*total_elasped_time_sec);
///////////// Sine wave target END /////////////
```

67. Tune the PID controller gain to ensure that the error value after 5 seconds is between -0.1 and 0.1 rad as shown in the following graph.

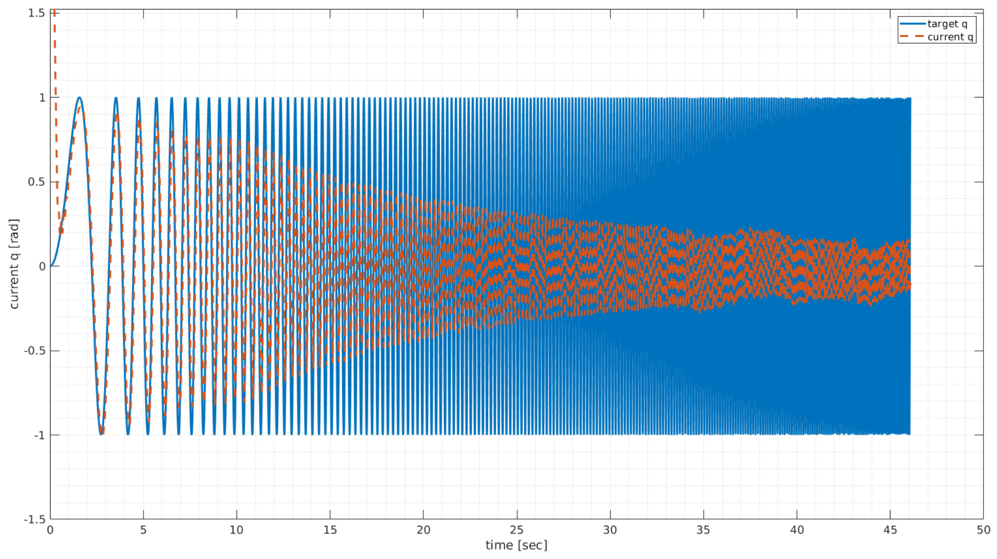


68. Use the same gain. Make a chirp signal. Comment out the fixed frequency, Uncomment the chirp signal's frequency.



```
double amp = 1;
///////////////// fixed frequency /////////////////
// double sin_wave_frequency = 2*M_PI*0.77; // [\omega, rad / s]
////////////////// fixed frequency END /////////////////

///////////////// For Chirp Signal /////////////////
double sin_wave_frequency = 2*M_PI*0.1*total_elasped_time_sec; // [rad / s] for chirp signal.
////////////////// For Chirp Signal /////////////////

///////////// Constant target /////////////
// targetQ << 1;// M_PI;
// targetQdot << 0;
///////////// Constant target END /////////////

///////////// Sine wave target /////////////
targetQ << amp*sin(sin_wave_frequency*total_elasped_time_sec);
targetQdot << sin_wave_frequency*amp*cos(sin_wave_frequency*total_elasped_time_sec);
///////////// Sine wave target END /////////////
```

(When calculating the target motor angular velocity from the chirp signal, the chain rule was omitted and the equation is incorrect. To correct it, apply the chain rule to derive the appropriate target motor angular velocity.)



69. Plot the graph using the target motor angle and the actual motor angle.
70. Using the meaning of the example from the bode plot result on page 11 of the lecture_control_part2.pdf, Discuss the result including how the amplitude and phase delay phenomena change as the frequency increases.
71. Save your all CSV data, and take it for your report writing.

## Motor Control **Week 2 End**

## Reference

[1] Dynamixel Velocity PI controller https://emanual.robotis.com/docs/en/dxl/x/xc330-t288/