README

Dante Eleuterio dos Santos

This repository contains the directories for the almond board actuonix motor, the basic dynamixel controllers and the OMX container.

1 almond_ws

The almond_ws directory was used for controlling the PQ12-30-12-P Actuonix motor with the Almond Board motherboard.

1.1 Relevant Links

1.2 Must Watch

- https://youtube.com/playlist?list=PLP0ejH1sBWj939Fdy46L3Zgez9lYR23_S&si=xN7-ry6o2KzJqT1g (Northwestern Robotics PID Playlist)
- https://youtube.com/playlist?list=PLn8PRpmsu08pQBgjxYFXSs0DEF3Jqmm-y&si=d9sC7KBgRjUSRfvz (MatLab PID Playlist - Watch at least the first two videos)

1.3 Important

- https://github.com/vncprado/lab_dynamixel_contol (Project Repository)

1.4 Complementary

- https://github.com/Open-Bionics/FingerLib/tree/master (FingerLib library)
- https://www.robotsforroboticists.com/pid-control/ (Mainly for the pseudocode and the manual tuning table)
- https://github.com/JeelChatrola/Control-of-motor-with-ROS (Example of ROS control that I based myself on)
- https://www.arduino.cc/en/software (Arduino IDE)

1.5 Setup Summary

- Download Arduino IDE: https://www.arduino.cc/en/software
- Clone the project Repository: https://github.com/vncprado/lab_motors_control/tree/master
- Clone the modified version of the FingerLib folder from the repository into the Arduino libraries folder (Make sure to check if my folder locations match yours) :

```
cp -r lab_dynamixel_contol/almond_ws/src/FingerLib ~/Arduino/libraries/
```

- Add yourself into the dialout group

```
sudo usermod -a -G dialout $USER
```

- Go to Boards Manager on the left side of the Arduino IDE, search for Arduino AVR Boards by Arduino and download it.
- To connect IDE into the board and the port click on the "Select Board box" on the top side of the interface and search for the Mega 2560 board. The port that I used was the /dev/ttyUSB0.
- Go to File and then to Open... on the top left side of the IDE and open the file lab_dynamixel_contol/almond_ws/src/almond/src/test.ino to see my example code to create a ROS enviroment to control the position of the actuator.
- Open rqt and use the plugins Topics/Message Publisher, Topics/Topic Monitor and Visualization/Plot to publish on the position topic to change the position of the actuator and to visualize the position plot.
- Running the ROS topics:

```
roscore

*OPEN NEW TERMINAL*

rostopic pub goPosition2 std_msgs/Int16 {SetPoint} --once
```

- The PID tuning can be done in the pid_controller.h file in the FingerLib library.

1.6 FingerLib Briefly Explained

This will be a brief explanation of how the FingerLib works:

- The FingerLib.cpp and .h files are responsible for the Finger class and its functions to control the finger.
- The pid files are responsible for the PID controller and its details can be found in the next section.
- The timers files are responsible for the timers used for internal operation of the library.
- Roughly every 9 ms the timer function will call the motor function, recalculate the PID and move the motor.

1.7 Code Changes Explained

1.7.1 test.ino

This is the arduino code that I used for tests.

First I include the headers

```
#include <ros.h>
#include <std_msgs/Int16.h>
#include <FingerLib.h>
```

ROS NodeHandle is our node, the variable str_msg and the position Publisher will be used to publish the real position of the actuator on a topic and finger belongs to the Finger class from the FingerLibrary.

```
ros::NodeHandle nh;
std_msgs::Int16 str_msg;
ros::Publisher position("position", &str_msg);
Finger finger;
```

The writePosition function and the goPosition Subscriber are used to receive the value of the target position.

```
void writePosition(const std_msgs::Int16& msg) {
    int aux = msg.data;
    finger.writePos(aux);
}

ros::Subscriber<std_msgs::Int16> goPosition("goPosition2", &writePosition);
```

In the setup function I started the Serial for debugs and then used the FingerLibrary function to start the pins 5 and 2 as OUTPUT and the A0 pin as INPUT. Then I start the node and subscribe and advertise the topics.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(38400);
  finger.attach(5, 2, A0);
  nh.initNode();
  nh.subscribe(goPosition);
  nh.advertise(position);
}
```

In the main loop I just keep publishing the actual position of the actuator on the position topic.

```
void loop() {
    str_msg.data = finger.readPos();
    position.publish(&str_msg);
    nh.spinOnce();
    delay(10);
}
```

1.7.2 pid_controller.cpp

The PID controller is done at the pid_controller.cpp file in the pid folder at the FingerLib repository.
The PID code is very similar to the basic PID code structure that you must have seen on the videoclasses. As I had to make some changes at the run function please read the code comments:

```
double PID_CONTROLLER::run(double targ, double curr){
    if (!_en){  //if not enable don't run
        return 0;
    }
    double output = 0;
    double error = targ - curr;
    double sampleTime = (_sampleTimer.stop() / 1000);  // sampleTime in ms
    if (sampleTime>0){ //Not really sure if it's necessary
        //Anti-windup for the integral factor
        if(error>targ*0.33 || error < -targ * 0.33){
            _integral=0;
        }else{
            _integral +=  error * (sampleTime/1000);    //time in seconds
            _integral = constrain(_integral, _min, _max);
        }
        // store the change in input to remove the derivative spike
        double _dInput = ((error - _prev) / (sampleTime/1000)); //time in seconds
        // calculate and constrain output
        output = (_Kp * error) + (_Ki * _integral) + (_Kd * _dInput);
        output = constrain(output, _min, _max);
        // if a rate limit is set
        if (_rampLim != 0){
            double change = (output - _prevOutput);

            // if the change in outputs exceeds the rate limit
            if (change > _rampLim)
            {
                output = _prevOutput + _rampLim;  // just change by the ramp rate
            }
            else if (change < -_rampLim)
            {
                output = _prevOutput - _rampLim;  // just change by the ramp rate
            }
        }
        // store history
        _prev = error;   // previous pos/vel
        _prevOutput = output; // store the previous output
    }
    // restart the sample timer
    _sampleTimer.start();
    return output;
}
```

1.7.3 pid_controller.h

The K_p, K_i, K_d values were found though the use of brute force . In the end I settled for K_p=3,K_i=0.7 and K_d=0.3.

```
    #define DEFAULT_GAIN_P  3
    #define DEFAULT_GAIN_I  0.7
    #define DEFAULT_GAIN_D  0.3
```

2 dynamixel_ws

2.1 Introduction

The dynamixel workbench package can be used to control dynamixel motors and it is a metapackage that contains 3 other packages: dynamixel_workbench_controllers, dynamixel_workbench_operators, dynamixel_workbench_toolbox.

2.2 Links

- https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_workbench/ (dynamixel_workbench tutorial)

- https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_wizard2/ (dynamixel wizard 2.0)
- https://github.com/vncprado/lab_motors_control

2.3 Dynamixel Info

All the following information can be found in the Dynamixel Wizard 2.0 (the link is in the links section) :

- IDs = 11 and 12
- protocol = 1.0
- usb_port= usually /dev/ttyUSB0
- baud_rate = 1000000
- model = 12

2.4 Failure to use topics

6 The Dynamixel_workspace tutorial(that can be found at the Links section) offers the possibility to use the /dynamixel_workbench/joint_states topic however it requires a trajectory that can be calculated using the MoveIt ROS software but it was decided not to use it because of bad previous experiences.

2.5 Summary

First install the dynamixel workbench library for ROS through apt-get(change noetic for your ROS version) :

```
sudo apt-get install ros-noetic-dynamixel-workbench
```

It is recommended at this point to follow the tutorials at the dynamixel_worbench website link that can be found at the links section. Keep in mind that you might have to change the ID and the baud rate at the launch file.
Create you own ros workspace and copy the launch/dynamixel_controllers.launch and config/basic.yaml files into it which can be found by running roscd dynamixel_worbench_controllers (you can also check my example at the repository link for the files). Keep in mind that you will have to change " find dynamixel_workbench_controllers " to " find your_package_name". The baud rate can be changed at the launch file and the IDs of the dynamixels can be changed at the config file. You can change the following line at the launch file to suport any other config files you want to use:

```
<param name="dynamixel_info" value="$(find
    dynamixel_workbench_controllers)/config/basic.yaml"/>
```

3 omxchanges PID Tuning

The omxchanges directory is a fork of the OMX-Moveit-devcontainer repository which can be found at the following link: https://github.com/vncprado/OMX-Moveit-devcontainer. It can be used to control the Openmanipulator-X robot.

4 Relevant Links

- https://youtube.com/playlist?list=PLPOejH1sBWj939Fdy46L3Zgez9lYR23_S&si=xN7-ry6o2KzJqT1g (Northwestern Robotics PID Playlist)
- https://youtube.com/playlist?list=PLn8PRpmsu08pQBgjxYFXSsODEF3Jqmm-y&si=d9sC7KBgRjUSRfvz (MatLab PID Playlist - Watch at least the first two videos)
- https://www.sciencedirect.com/topics/computer-science/ziegler-nichols-method (Ziegler-Nichols)

4.1 Files

The launch file joint_position_controller.launch for the dynamixels can be found in the directory /home/bioinlab/lab_dynamixel_contol/omxchanges/OMX-Moveit-devcontainer/src/open_manipulator_controls/open_manipulator_controllers/launch.

The PID config file joint_position_controller.yaml can be found in the directory home/bioinlab/lab_dynamixel_contol/omxchanges/OMX-Moveit-devcontainer/src/open_manipulator_controls/open_manipulator_controllers/config

4.2 PID Tuning

To tune the PIDs of Dynamixel joints the Ziegler-Nichols method was used in the following steps:

4.3 RQT

- Set up RQT using the Topics/Message Publisher ,Visualization/Plot and Configuration/Dynamic Reconfigure plugins.
- In the Message Publisher the /jointX_position_controller/command (being X the number of the joint) topics were used to publish the joint positions.
- The /jointX_position_controller/command/data was plotted to keep track of the setpoint, and joint_states/position[I] (being I the index of the joint) was used to know the real position of the joint.

- The dynamic reconfigure was used to dynamically change the value of the PID variables.

4.4 Ziegler–Nichols

- The Ziegler–Nichols method which consists on setting I and D as 0 and then keep increasing the P value until constant oscillation is reached (it must be used with care with real life motors) and save this P value as the K_p
- After this the period(P_0) of the wave must me measured which can be done by extracting the data from the plotting and using analyzing it,by eye-measuring how long it takes for one oscillation to happen or by founding the frequency(F_0) by counting how many oscillations happen in one second and $P_0 = \frac{1}{F_0}$
- The base PID values will be defined as P= 0.6 $\cdot$K$_p$, I= $\frac{P_0}{2}$,D=$\frac{P_0}{8}$
- Usually the found values will not be enough to tune the controller but can be used as a basis for the real tuning which can be found through try and error testing and it is recommended to watch the videos on the relevant links section.

5 Viral_pc

The Viral_pc directory is composed by some code that was given to me by the researcher Viral so that I could work on the alignment of the robot IMU sensors. Not mush success was achieved in this part of the research and the only relevant change that was made was the rotation of the frame of the right sensor in the Y axis which can be found at the following file inte lines 105 to 110:
lab_dynamixel_contol/viral_pc/viral/src/bioin_tacto/python_scripts/publish_sensor_transforms.py