Almond Board PID

Dante Eleuterio dos Santos

1 Introduction

This report will talk about the steps that I took to control the actuonix PQ12-30-12-P with the Almond Board and to tune its PID.

2 Relevant Links

2.1 Must Watch

- 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)

2.2 Important

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

2.3 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)

3 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.
- The tuning can be done in the pid_controller.h file in the FingerLib library.

4 Code Explained

4.1 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);
}
```

4.2 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;
}
```

4.3 pid_controller.h

The hardest part of the process was to find the K_p, K_i, K_d values as I had to use brute force to find them. In the end I settled for K_p=3,K_i=0.7 and K_d=0.3 (require more tests).

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

5 PID Tuning

The PID Tuning could also be done using some known methods as the Ziegler-Nichols as the T_u (period) can be roughly calculated using the rqt plot plugin. (require more tests)