A low cost and flexible open source inverted pendulum for feedback control laboratory courses

Dr. Matthew Turner, Purdue University (Statewide Technology)

Prof. Timothy R. Cooley, Purdue University, New Albany

Tim Cooley is an Associate Professor in the Purdue Mechanical Engineering Technology Department at the New Albany, Indiana location.

A low cost and flexible open source inverted pendulum for feedback control laboratory courses

Abstract

This paper presents the complete design and build of a low-cost, open source inverted pendulum (IP) platform to support control systems engineering and technology laboratory instruction. In this standard IP system, a linear actuator consisting of a belt drive and stepper motor is used to stabilize the vertical angular position of an inverted pendulum connected to a cart. The novelty of the presented system is the open-source approach, which achieves both low-cost and permits individual customization. The linear actuator and cart were designed in SolidWorks and manufactured using a MakerBot Replicator 2X, with the design files published on Thingiverse.com. Linear motion is achieved via a standard NEMA17 size stepper motor driven by an Arduino Uno microcontroller development platform and an open source stepper motor driver circuit that can either be purchased or built in-house. Angle conversion of the IP is performed using a low cost rotary encoder and the analog to digital converter of the Arduino Uno. Additionally, a digital Proportional Integral Derivative control algorithm is presented that addresses the issues of sample time, derivative kick, on-the-fly tuning and reset windup. The code is documented to explain these phenomenon and to enable tuning using standard practices such as Ziegler-Nichols tuning. Through the use of 3D printing technology and open-source electronics and computer code, the material cost of the system was kept under \$100 per unit, making this an ideal student project for an undergraduate controls curriculum. Additionally, through open access to the design files, control systems educators and students have the flexibility to customize the project to their individual needs. Student feedback is also presented supporting the efficacy of the system as an active learning tool.

1. Introduction

The inverted pendulum control experiment, in which a pendulum with a center of mass above the pivot point is mounted to a linear actuator and the actuator is moved to attain a balanced condition (Figure 1), is a common example used in introductory feedback control systems courses¹, particularly in the design of the Proportional Integral Derivative (PID) control algorithm². Since the inverted pendulum is inherently unstable, it provides an interesting laboratory experiment for students to perform and can be easily linked to real-world applications such as rocket and missile guidance system or the Segway self-balancing electric vehicle. In addition, the inverted pendulum is often used as the test case for control strategies³ which may be encountered in practice or in graduate coursework. For these reasons the inverted pendulum is often considered one of the classic problems in control theory, and has been adapted to balance the pendulum ($\theta=90^\circ$), maintain the pendulum at a constant angle ($\theta\neq 90^\circ$), or control a multi-segment pendulum.

Figure 1. Linear actuator (cart) and Inverted pendulum system.

Often when the inverted pendulum is used as a laboratory instructional project, the plant, manipulating element, and measuring transmitter (Figure 2) are pre-assembled for students, who instead study and implement the controller element, often in the form of a PID controller. Although this approach enables increased attention to be paid to the design and implementation of the control algorithm, it deprives students of the learning experience associated with physically constructing the entire feedback control system: conversion of physical parameters into electrical signals, integration and wiring of the manipulating element, etc. However, building such systems is often time intensive and therefore unsuitable for use in a class or laboratory setting.

Figure 2. Closed loop feedback control system.

The inverted pendulum control system kit presented in this paper (Figure 3) addresses this issue by reducing design and build times through the use of open source hardware and software modules that can be customized to specific applications. The system plant consists of an inverted pendulum (1) that has been attached to a slider (2), which is in turn connected to two rails that form a track in the x-direction(3). The slider is moved along the track via the manipulating element: a stepper motor (4) and pulley and belt system (5), which form a simple linear actuator. The measuring transmitter consists only of a potentiometer (6) attached to the bottom of the pendulum rod that is used to convert a change in resistance to a change in voltage, which is compared to a reference voltage signal to calculate the angular position error. The

stepper motor is actuated by an H-bridge motor driver circuit (7), with an Arduino Uno microcontroller development platform (8) used to provide computer control to the H-bridge and to act as the controller.

Figure 3. Inverted pendulum control systems kit.

2. Hardware Design

To facilitate the build of the inverted pendulum kit, several portions of the linear actuator and sliding cart were designed in SolidWorks 3D computer aided design software for fabrication using 3D printing technology. The end-stops of the linear actuator (Figure 4, left and middle) serve as supports for the rails, built from 8mm 1060 carbon steel rods, and for a subminiature snap action limit switch on each end. The stepper end stop (Figure 4, left) is designed to mount a standard NEMA17 stepper motor via a motor mounting plate. The pulley end stop (Figure 4, middle) is designed to hold a custom-designed pulley and mounting plate. The sliding cart (Figure 4, right) accepts two linear 8mm ID linear bearings and a Panasonic P122426TR rotary potentiometer.

Figure 4. Linear actuator assembly.

To actuate the cart, a drive belt was made using a ¼” wide timing belt with tooth pitch of 0.2”. The belt is held in tension by a custom pulley (Figure 5, left), a gear (Figure 5, center), and a belt tensioner (Figure 5, right). Two custom mounting plates attach the pulley and stepper motor / gear to the end-stops of the linear actuator. The belt tensioner is also used to connect the belt drive to the sliding cart using two 6-32 machine screws. All five pieces were designed in SolidWorks, with the pulley, gear, and belt tensioner designed for 3D printing. The mounting plates are designed for cutting on a laser cutter, but could be fabricated with standard shop equipment. The parts for the linear actuator and cart were manufactured with Acrylonitrile butadiene styrene (ABS) using a MakerBot Replicator 2X⁴. By using a soft thermoplastic, the linear actuator parts are easily modified by students for on-the-fly design changes. The CAD files have been published to Thingiverse⁵ in STereoLithography (STL) format, and are free to download, modify, and re-distribute.

Figure 5. Drive belt assembly.

The rotary potentiometer (Figure 6, left) was attached to the sliding cart with a friction fit into a notch and with liquid adhesive. Additionally, the sliding cart has four strain-relief holes that accept 22-24 gauge stranded wires from the pot. The pendulum arm (Figure 6, center) was fabricated from a ½” wood dowel, with the pivot end sanded into a curve and a pivot point drilled with a 16 gauge drill bit. The pendulum arm was then affixed to the cart using an 8-32 machine screw (Figure 6, right) that had been ground to a flat on the end matching the profile of the rotary potentiometer and two 8-32 nut/washers. When filing the machine screw, leaving an 8-32 nut on the screw can help to re-align threads damaged during machining.

Figure 6. Pendulum arm pivot point assembly.

3. Circuit Design

The inverted pendulum kit is controlled by an Arduino Uno microcontroller platform (Figure 7). Angular position of the pendulum arm is determined from the 10-k potentiometer configured in a

voltage divider circuit. The analog voltage output signal from the angle sensor is read by the Arduino Uno's 10-bit analog to digital converter A0, resulting in a mapping of the 0 to 5 volt input voltage to an integer value between 0 and 1023 with a voltage resolution of 4.9mV. End stop switches mounted at the ends of the linear actuator are configured in the Normally Open configuration with an RC pull down network. Depressing either end stop switch results in 5 volts at the Arduino D2 input.

Figure 7. Inverted pendulum kit control circuit.

The control for the stepper motor that drives the linear actuator is provided via a DFRobotics L298P Motor Driver shield for the Arduino. This shield utilizes the STMicroelectronics L298P Dual Full-Bridge Driver, which can operate up to 46V and deliver a DC current of 2A from an appropriate power supply, in this case a 12V 5A DC switching power adapter. Control logic to deliver the appropriate voltage pulse trains comes from an Arduino Uno, as shown in Figure 8, which implements a bi-polar, one-two phase step sequence, or half-stepping. Rotational direction of the stepper motor can be reversed by the Arduino by reversing the DC pulse order.

Figure 8. Bipolar stepper pulse sequence for half-stepping.

4. Motor Control Source Code

To control the motors, the step sequences from Figure 8 were individually coded for speed and direction as Arduino sketch functions, with the addition of a function to turn both motors off. These individual functions were placed into an array of function pointers, and stepping through the array calls the functions in order, resulting in forward rotation. For example, Step 0 from Figure 8 is coded as:

```
// -----ENABLE BOTH MOTORS AND SET MOTOR 1 HIGH AND MOTOR 2 LOW-----
Void Step0 () //declare the function named Step0 that returns no information
{
    digitalWrite(M1, HIGH);    // Enable motor 1
    analogWrite(E1,PWR);      // Set the value of motor 1 to high level
    digitalWrite(M2, HIGH);    // Enable motor 2
    analogWrite(E2, 0);        // Set the value of motor 2 to low level
    delayMicroseconds(DELAY); // A delay is required so not to exceed the maximum slew rate of
                               // the stepper
}
```

The H-Bridge motor shield then translates the bi-polar logic of Figure 8 into the appropriate voltage signals as provided by the external voltage supply to the motor shield. The array of function pointers is then coded:

```
// -----CREATE AN ARRAY OF POINTERS FOR EACH OF THE STEPPER FUCNTIONS-----
int *steps[8] = {(int*)Step0, (int*)Step1, (int*)Step2, (int*)Step3, (int*)Step4, (int*)Step5,
(int*)Step6,(int*)Step7};
```

Reversing the direction through the array will result in reverse rotation. Two additional functions, Forward() and Reverse() handle this. These function both receive an integer input that corresponds to the number of times to step through the array of function pointers:

```
// -----STEP THROUGH THE FUNCTION ARRAY FROM Step0() TO Step7() AND REPEAT-----
Void Forward(int length) // Declare the function name Forward that has an integer
                          // argument length
{
    int i; //Declare the variable i to be used to iterate through the loop.
    void (*s)();
    for (i=0; i<length; i++)
    {
        s=(void (*)( ))steps[i%8];
        (*s)();
    }
}
```

The value of the variable length in the Forward() function can be calibrated to achieve specific distances of travel by the linear actuator. Once declared, the functions Forward() and Reverse() can be called iteratively to cause the stepper motor to rotate clockwise or counter clockwise fixed distances. For example, the following Arduino sketch uses Forward() and Reverse() to move the from end to end of the linear actuator in increments of *length*:

```
// -----MOVE THE CART BACK AND FORTH BETWEEN THE END STOP SWITCHES-----
Void setup()
{
    attachInterrupt(1, SwitchInterrupt, RISING); //Create an externally triggered interrupt on a rising
```

```

}                                     edge signal on pin D3

Volatile boolean interruptflag =true;           // Declare a Boolean variable interrupt flag that can
                                                // be changed inside of an interrupt

Void SwitchInterrut() //declare the interrupt function
{
    interruptflag = !interruptflag;           // Invert the value of interrupt flag each time the
                                                // interrupt is called, i.e. when the momentary switch
                                                // connected to pin D3 is pressed
}

Void loop() //write the main program routine
{
    length = 200;           //Set the distance the linear actuator will travel per call of the
                            // Forward()/Reverse() functions
    if( interruptflag)      // If the interrupt flag is true, move in the reverse direction
    {
        Reverse(length); // Reverse the stepper motor
        MotorsOff();     // Turn off the motors in case the direction changes
    }
    else
    {
        Forward(length); // Move the stepper forward
        MotorsOff()      // Turn off the motors in case the direction changes
    }
}

```

5. PID Controller Source Code

To provide feedback control to the inverted pendulum, proportional, integral, derivative control algorithm is used to adjust the variable *length* and to change the direction of movement according to the error, based on the equation for the three-mode PID controller:

$$Output = K_P * e + K_I \int e dt + K_D \frac{de}{dt} [1]$$

In this equation, error is the difference between the angular position of the pendulum and the desired angle set-point, and K_P , K_I , and K_D are the proportional, integral, and derivative gain coefficients that are used to tune the PID controller. Output can be either the total control effort of the plant, or can be a multiplication factor by which a base control effort is scaled, with this implementation using the later, adjusting the total distance traveled by the cart for each call to the Forward() or Revers() functions.

The Arduino development environment has an included PID library that can easily be implemented by students for rapid results ⁸. However, students can also program their own PID functions. A basic PID function based on Equation 1 and reference 8 is:

```

// -----SET GAIN CONSTANTS AND COMPUTE PID OUTPUT-----
float kp,ki, kd; //declare variables used to adjust gain constants

Void SetTunings(float KP, float KI, float KD) // Create function to set gain constants. Using a function is
                                                // necessary if on-the-fly-tuning is implemented

```



```

{
    kp = KP; //set proportional constant
    ki = KI; //set integral constant
    kd = KD; //set derivative constant
}

//Declare necessary global variables to be passed between calls of ComputPID()
float Setpoint, Input, Output;
unsigned long Told;
float ErrorSum, ErrorOld;

Void ComputePID()
{
    float error = Setpoint - Input; //Calculate error between Setpoint and Input
                                    //from angular sensor
    float Tnew = millis(); //Store current time
    float dT = Tnew - Told; //Calculate the time change since the last
                            //PID calculation by subtracting previous
                            //time from current time

    ErrorSum += (error * dT); //Approximate the integral by summing the
                             //total error
    float dError = (error - ErrorOld) / dT; //Approximate the derivative by dividing
                                           //the change in error by the time change

    Output = kp * error + ki * ErrorSum + kd * dError; //Compute the PID controller output

    //Update variables for use in next iteration of ComputePID()
    ErrorOld = error;
    Told = now;
}

```

These functions can then be called as part of the main program routine:

```

// -----MAIN PROGRAM LOOP-----
Void loop()
{
    SetTuning(1,0,0); //Set the PID gain constants
    Start:
    int oldstate=0; //Create variables to keep track of motor
    direction
    int newstate=0;
    int sensorValue = analogRead(A0); //Get the voltage from the angle sensor
    Input = sensorValue * (5.0/1023.0); //Convert the binary reading into equivalent
                                       //voltage
    ComputePID() //Call the PID function
    length=int(200*Output); //Use the output of the PID function to scale
                           //the distance the cart is moved for each
                           //call of Forward() or Reverse()

    if(length<200) length = 200; //Set minimum distance cart can travel
    if(length>1000) length =1000; //Set maximum distance cart can travel
    if(error <0) //If the error is negative, move the cart the
                //appropriate distance in the reverse

```

	direction and record the direction of movement
{	
newstate=1;	//A 1 indicates reverse travel
if (newstate !=oldstate) MotorsOff()	//If the cart needs to change direction, the motors need to be turned off first
Reverse(length);	
}	
if(error>0)	//If the error is positive, move the cart the appropriate distance in the forward direction and record the direction of movement
{	
newstate=0;	//a 0 indicates forward travel
if (newstate !=oldstate) MotorsOff()	//If the cart needs to change direction, the motors need to be turned off first
Forward(length);	
}	
oldstate=newstate	//Update motor direction state variable for next iteration of loop
if (interruptflag==1) goto start;	//Check to see if the motor has reached an end stop.
MotorsOff();	//Turn of the motors
Finish:	//Enter an infinite loop to halt program
Goto finish;	
}	

6. PID Enhancements

Derivative kick describes the phenomenon of a PID controller creating a large output due to derivative action when the set point or input suddenly changes (Figure 9). The derivative term of the PID controller can be improved by taking the negative derivative of the input signal instead of the derivative of the error signal.

Figure 9. Derivative kick in a PID controller

To implement a PID controller that responds to changes in the input signal as opposed to changes in the setpoint, or derivative on the measurement, the following change can be made to ComputePID():

```
float dInput = (Input - oldInput)/dT; //calculate the derivative of the input

Output = kp * error + ki * ErrorSum - kd * dInput; //Compute the PID controller output

oldInput = Input; //Update input for next iteration of PID calculations
```

In this implementation, a change in the tuning parameter k_i will affect the entire k_i ErrorSum product, resulting in an over-emphasis of the integral term in the PID algorithm that complicates tuning the controller. This can be easily fixed by calculating the integral term iteratively. In ComputePID(), replace:

```
ErrorSum += (error * dT);
```

With:

```
ITerm += (ki * error*dT);
```

And compute the PID output based on ITERM:

```
Output = kp * error + ITerm - kd * dInput;
```

An additional problem with the integral mode is Reset windup⁸ in which the I-term of the integral mode will continue to integrate and change the output of the PID controller even when outside of the operating range of the manipulation element. Therefore, although no further change occurs in the plant, the integral term of the PID calculation continues to grow. If the plant is then to return to within an operable range, a period of time exists during which the controller will still respond with a large integral term, as it takes time for the integral mode to reset down from the wound-up condition. In the PID implementation presented, the total Output of ComputePID() has already been clamped to min and max values. However, the ITerm component of the PID output equation can still grow unchecked and needs to be clamped:

```
if(ITerm> outMax) ITerm= outMax;
else if(ITerm< outMin) ITerm= outMin;
```

7. Student Feedback

To evaluate the pedagogical value of the inverted pendulum experiment, a survey was posted to students. For each category, students were asked to evaluate the impact of the lab experiment on their learning from no-impact (grade of 0) to excellent impact (grade of 10). Although only evaluated by a small student sample ($n=11$), initial results are promising (Table 1). Some open student comments are also presented.

The results of the experiment assessment reveal that the hands on approach had positive effects on student's perception of their own understanding of the PID algorithm, their understanding of the hardware requirements of a closed loop feedback control system, and confidence in creating

PID systems. Previous to this experiment, PID algorithms had only been implemented in MATLAB simulations. The IP system was less successful at educating students in the software aspects of the PID algorithm. Open ended comments revealed that students found the software control of the H-bridge to be too confusing, which is likely a result of the use of pointer arrays in the sub-routine. A recommended change based on this input is to control the linear actuator with a stepper motor driver to simplify source code for both motor direction and speed control. Additionally, low scores were reported for both impact to understanding of derivative kick and reset windup. It is hypothesized that both scores could be improved by having students implement PID algorithms without these PID enhancements for comparison studies.

Table 1 Student Evaluation Results

Question	Mean value
Impact on understanding of PID algorithm design.	8.64
Impact on understanding of PID implementation in software.	6.1
Impact on understanding of PID implementation in hardware.	9.34
Impact on understanding derivative kick errors.	6.0
Impact on understanding reset windup errors.	6.64
Improved confidence in creating PID system for other applications.	7.45
Paraphrased relevant student comments	
Motor control algorithm difficult to understand	
Too much code	
Fun to work with hardware/nice change from simulation	
Needs more practical application	
How to tune?	

8. Conclusion

Although a complex non-linear control problem, the inverted pendulum experiment can serve as a lively educational and demonstration tool for undergraduate controls courses. Unfortunately, the cost of pre-built systems often prohibits them from being “dissected” or modified by students. Conversely, the design and build time required for custom IP systems prohibits them from use as part of a larger controls curriculum. The demonstrated customizable IP kit and associated PID controller solves these problems via an open-source approach that results in a fully customizable kit that can be fabricated and controlled in a relatively short time frame. This allows students to gain practical experience with physical implementation, develop fully original control functions, and well as understand the nuances of automatic feedback control, such as windup, derivative kick, and on-the-fly tuning. By sharing the source code and design files, the inverted pendulum kit can be widely disseminated and further improved upon by the educational community.

9. References

- [1]. Horáček, P. "Laboratory experiments for control theory courses: A survey." *Annual Reviews in Control*, vol. 24, 2000.

- [2].Wang, J. "Simulation studies of inverted pendulum based on PID controllers." *Simulation Modelling Practice and Theory*, vol. 19, no. 1, 2011.
- [3].Cong, S., D. Zhang, and H. Wei. "Comparative study on three control methods of the single inverted-pendulum system [J]." *Systems Engineering and Electronics*, vol. 11, 2001.
- [4].www.makerbot.com
- [5].www.thingiverse.com
- [6].<http://playground.arduino.cc/Code/PIDLibrary>
- [7].Bohn, C. and Atherton, D.P., "An analysis package comparing PID anti-windup strategies," *Control Systems, IEEE* , vol.15, no.2, pp.34,40, Apr 1995