PID Motor Controller

Final Project Report
December 9, 2006
E155

Amir Adibi and Andrew Danowitz

Abstract:
The primary objective of this project is to develop a PID motor controller that maintains a set angular velocity. In order to demonstrate a working PID motor controller, the target application is a moving cart which is able to maintain a constant speed. Implementation is demonstrated on a cart which traverses an incline and decline of equal distances in roughly equal times. In order simplify the mechanical design, a LEGO Mindstorm and TECHNIC kit are utilized for the powertrain system, and the cart body is designed with cardboard and sheet metal. Power supply and breadboard (containing all digital devices, i.e. Harris Board 2.0) are carried alongside the cart, due to weight and torque issues. All PID processing is performed on the PIC18F452 and motor feedback is acquired using an absolute motor shaft encoder. A tuning method based off of the Ziegler-Nichols Algorithm is used to experimentally determine the P, I, and D values.

Introduction

In an attempt to combine mechanical, electrical and control disciplines, the primary objective of this project is to design a cart with PID speed control. The original intention was to design a cart that could ease the movement of heavy loads between dorm rooms of Harvey Mudd Students; however, due to the high cost of a motor required to drive such a heavy load, the implementation was simplified to a LEGO Cart.

An overall system block diagram in Figure 1 demonstrates the objective of the control system. The angular velocity outputted by the control system corresponds directly to the velocity of the cart.

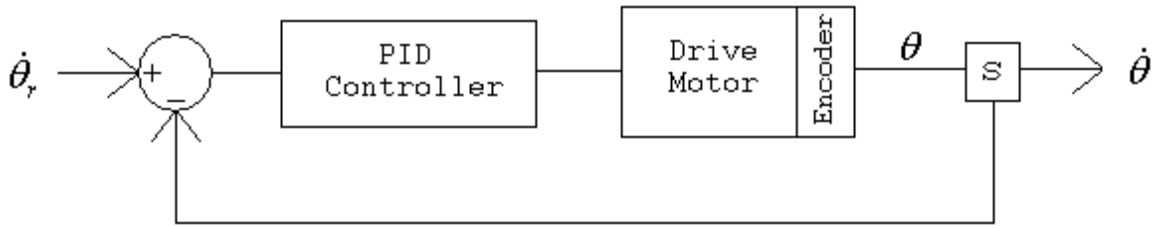


Figure 1: System Block Diagram

The simplest technique to acquire feedback from the motor is with an absolute shaft encoder connected to the shaft of the drive motor. The control system varies the voltage supplied to the motor depending on how the angular velocity deviates from the set voltage. If the angular velocity is too great, the voltage to the motor decreases, slowing down rotation. If the angular velocity is low, the control system compensates by supplying more voltage to the drive motor.

Mechanical System

LEGO Cart Design

The body of the LEGO Cart was constructed using a strip of cardboard. Strips of sheet metal were attached to the sides for extra support. Two LEGO structures were attached using rubber bands to the cardboard body. One structure supported the front axle and wheels. The back LEGO structure supported the back axle and wheels, motor, and encoder. Originally the power supply and hardware were placed above the cardboard body; however, the weight placed too much stress on the plastic axles and limited the movement of the cart, so these components are carried aside the cart during operation.

Motor

The motor of choice was selected from the motors available in the Microprocessors Lab. The motor offers plenty power to drive a LEGO cart. Because of the added complexity of finding gears that could be integrated into our system, gears available in the LEGO Mindstorms kit were used instead. The kit has only two gear sizes that could be easily coupled, so the low gear ratio is not a choice.

The motor is mounted on the end of the cart with a LEGO mount. An encoder is attached to it using a sheet metal insert that is affixed with two screws into the end of the motor, securing the shaft encoder. The motor is connected to the gear through a TECHNIC beam which is super glued to the end of the shaft. Additional LEGO pieces are super glued along the beam to add support. Originally the motor was attached without LEGO supports on the beam, but the beam ended up cracking under the torsional load. The motor was powered with the 12V power supply.

Wheels

Wheels were acquired from the LEGO Mindstorms and TECHNIC kits, and are simple to integrate into the cart.

Drivetrain

The drivetrain was designed using 2 gears: one gear connected directly to the motor shaft using a TECHNIC beam component, superglue and spare LEGOs for added support, and the other gear was connected to the rear axle.

Electrical System

Motor Feedback

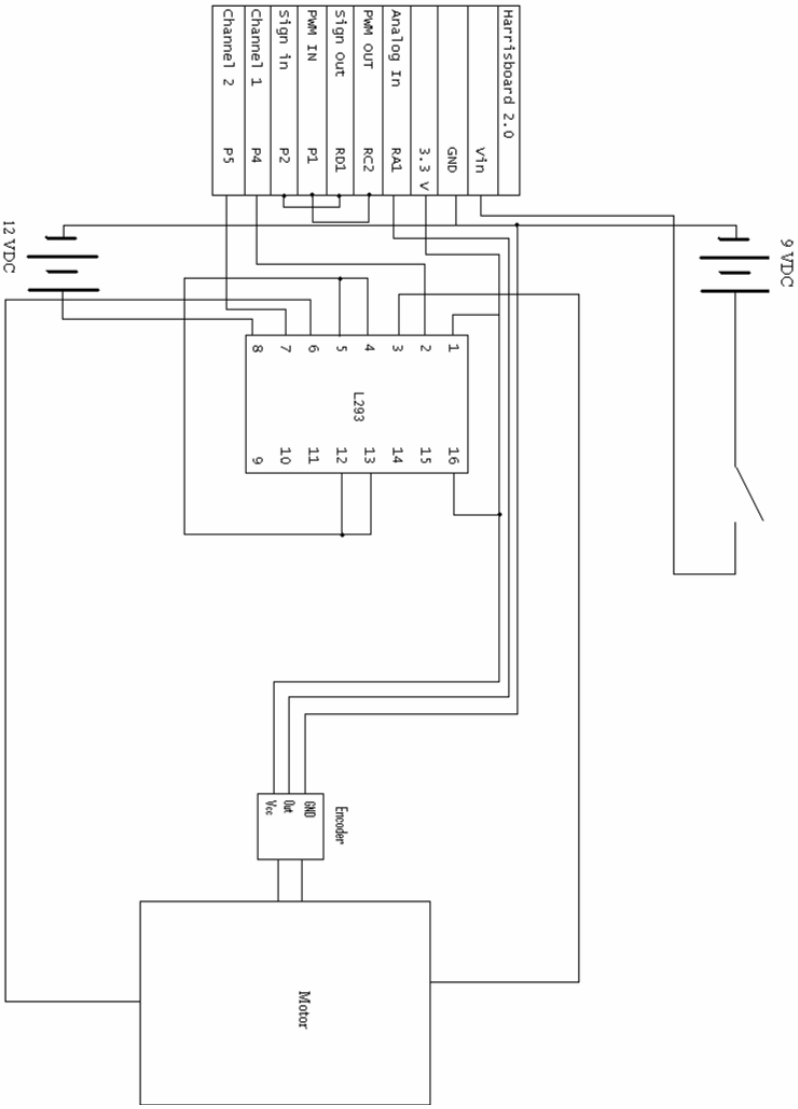
Feedback from the motor is acquired through an absolute encoder used to read the position of the motor shaft. Determining velocity from angular rotation of the shaft was determined to be simplest approach. Angular position of the drive shaft is translated into an analog signal between 0V and the input voltage (maximum of 5V). By comparing previous angular position to current angular position, velocity can be obtained.

The encoder is attached to the end shaft of the motor. Coupling the shaft of the encoder to the end shaft of the motor was problematic because the encoder shaft required a considerable amount of torque to rotate. This caused the rubber tube that coupled the motor to the encoder shaft to slip. Also, since the motor used in the cart was designed for low torque and high rpm, the torque required to spin the encoder drastically decreased the ability of the cart to traverse hills. A new encoder with ball bearings resolved the issue, making coupling simple and vastly improving the maximum slope of cart operation.

H-Bridge

In order to use the low current, low voltage output of the Harrisboard to control the motors, the team uses an H-bridge circuit. An L293 H-Bridge is used since they are made freely available to the team and are known to be compatible with the motors used. In order to verify that these H-Bridges would in fact work for the application, the team connected the H-bridge to the PWM module constructed in lab 7. Once this circuit was constructed, the team was able to verify that the H-Bridge, motor and battery were in fact compatible, and that the team was able to successfully vary the speed of the motor by varying the magnitude of an analog input voltage.

Breadboard Schematics



Control System
Digital Control System

Since the purpose of this project is to create a cart that travels at constant speed, the team decided to construct a PID controller to control a cart based off of angular velocity of the motor shaft. A schematic of a PID block diagram is shown in Figure 2.

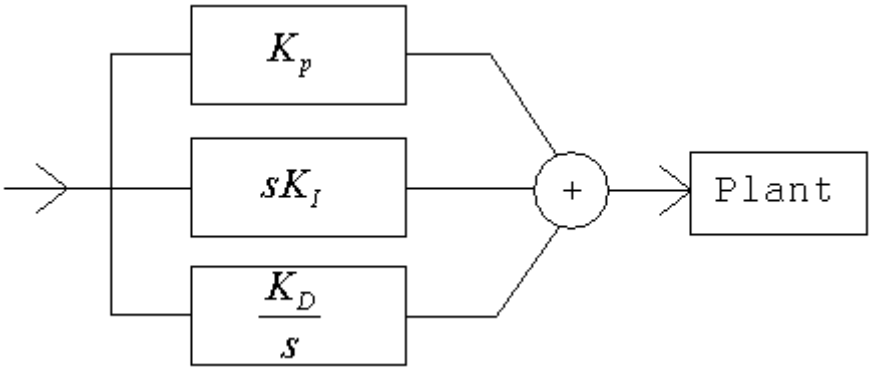


Figure 2: PID Block Diagram

P, proportional, is a straight gain that affects the response time. I, integral, accounts for previous error and increases the steady state error response (while causing a greater overshoot). D, derivative, predicts future values and decreases overshoot (while slowing the transient response). The purpose of the PID block is to provide a corrected voltage to the motors.

Microcontroller

The PID controller was implemented on the PIC 18F452 microcontroller on the Harrisboard 2.0. The controller was written in C code using the C18 PIC C compiler provided in lab. The team ultimately chose a C based implementation since while assembly does offer easier bit manipulation tools and generally utilizes fewer instructions than compiled C code, assembly also offers no easy way to multiply two large signed numbers and only supports 8 bits natively. The team decided that both of these drawbacks would have made the controller implementation much more time-consuming.

In order to calculate the angular velocity of the motor shaft, timer0 is set to overflow once roughly every 10 ms (~100 samples/second). Upon timer overflow, the control algorithm reads the shaft position from the absolute encoder through the A/D converter. The algorithm then checks for shaft direction and to see whether the shaft has completed a revolution since the last sampling cycle. Based off of this information and the previous position value, the algorithm calculates a signed value for the change in shaft position. This change in position is subtracted from a fixed, desired change in position in order to calculate the error signal. The controller then calculates the PID signal by multiplying the error by the proportional gain, multiplying the sum of all past errors by the integral gain, and multiplying the change in error between the current and previous sample by the

derivative gain. Once these terms are found, they are summed to get a signed, sixteen bit PID value.

It is important to note that due to the nature of the integral term of the PID controller, if an error in velocity goes uncorrected for too long, such as in the event that the cart comes to a slope that is too steep and the motor cannot provide the needed torque to control the speed, the integral error term can grow to be very large (possibly to the point of causing either the I or PID values to overflow) and thus make stable control and recovery in such situations exceedingly difficult. Since any uncorrectable error should always be accompanied by the motor running at maximum power in one direction, the team was able to correct for this potential problem by only updating the integral term of the PID controller if the motor was not at maximum power.

In order to power the motor, the sign and magnitude of the PID value must be determined. To do this, the controller first checks to see if the most significant bit of PID is high, which would indicate a negative number. If PID is negative, its two's-complement is taken, yielding a positive PID magnitude. In such cases, the "sign" output bit is set high as well.

The PIC PWM module is used to control the motor. The PWM module operates off of a 10 bit pulse-width value, whereas the PID controller outputs up to 16 bits of correction data. Since the cart was designed to move relatively slowly and thus should only ever need more than 10 bits of PWM data under extreme terrain or loading conditions, we decided that under normal operating conditions, the lowest ten bits of the PID value would be fed directly into the PWM data registers. In the event that the PID value exceeded ten bits worth of data (greater than or equal to 0x3FF), the pulse-width was set to maximum.

PID Controller Coefficient Selection

Since PID controllers are used to control external systems, proportional constants for P, I, and D must be carefully selected. In order to choose optimal proportional constants, it is often necessary to determine the transfer function of both the control system and system being controlled and mathematically solved. Since systems exist that can not be easily characterized, however, a number of experimental techniques, such as the Ziegler-Nichols method for "tuning" PID controller coefficients have been developed. With each of these methods, the system is first run with only a proportional gain. The proportional gain constant is varied until the system is just at the point of instability and the output oscillates periodically. The gain at which this point occurs is then scaled back by a factor of 2 and is used as the proportional gain constant. In order to find this optimum gain coefficient in our system, we programmed the PIC to output the sign bit of the controller to an LED. We then slowly increased the magnitude of the gain from one until we observed that the sign LED was flashing in regular periods. Since our digital implementation of the PID controller did not readily allow for the use of fractional constants, the team was forced to settle for the P value that came closest to causing a purely periodic oscillation of the system.

In many coefficient "tuning" methods, the integral and derivative gains are determined by multiplying the period of system oscillation from the proportional only controller by a predetermined constant. Given the speed of our controller, however, we found that the periods of proportional oscillation lasted roughly several milliseconds. Thus, according to the widely used tuning equations, the I and D gains would need to be fractional values, which is not really feasible in a PIC microcontroller. Upon consulting with Professor Bright of the HMC engineering department, we were informed that most tuning methods were designed for large scale industrial processes where the time of oscillation is much larger. He suggested that we find the P constant in the manner described by these tuning methods, and then reintroduce the I constant and scale it up until the transient errors were mitigated. After finding the P and I constants, Professor Bright suggested that we add in no more than a small derivative constant. With this in mind, the team experimented with the I and D constants until the system appeared to be smoothly controlling the motor.

FPGA

Both the sign bit and the PWM waveform generated from the PIC are directly sent to the Xilinx Spartan 3 FPGA. Based off of the value of the sign bit, the FPGA will output the PWM input to one of two channels connected to the power inputs of the H-bridge. The other channel connected to the H-bridge is set to ground. By changing which channel is set to PWM versus which channel is set to ground, the FPGA is able to run the DC motor either forward or in reverse.

Results

Verification of the controller was performed by setting checkpoints on the ends of a slope. The cart was timed as it traversed from one checkpoint to the other, in both directions (incline and decline). The cart traversed both in nearly the same time implying that a constant average velocity was maintained.

Upon careful observation of the cart's movement, it is apparent that the cart's uphill motion does not maintain a purely constant velocity, but sporadically increases and decreases speed rather than smoothly maintaining a constant speed. This is most likely due to suboptimal P, I, and D values in the controller.

The final cart successfully demonstrated a PID motor controller. If more resources and time were available, the original idea of designing a more powerful cart could be implemented using similar techniques.

Despite the success of the project, the team ultimately had to make some alterations to the original design plan. First, the team had originally intended to use two of the same motors to directly power the back wheels of a larger, wooden cart. In this design, the angular velocity of one of the motors would be measured and the same output would be applied to both motors. This idea had to be scrapped for a number of reasons. First, the motors did not apply enough torque to directly power the cart without gearing. Second, the use of one encoder to control two uncoupled motors assumes that the two motors, given the same input, produce roughly the same output, which is generally not the case. Thus the team was forced to redesign the cart using the LEGO solid axis drive-train construction and single motor drive found in the final cart.

The second change from the initial design was removing the battery and Harrisboard from the cart. Originally, both of these systems were to be mounted on the cart itself to form an entirely self-contained system. Due to limited torque output that the system is capable of, the cart could not handle the added weight of these components. Since our primary goal was to demonstrate a working motor controller proof-of-concept, the team decided to remove the power supply and Harrisboard from the cart.

Parts List

The motor and H-Bridge were acquired from the Microprocessors Lab.

Part	Source	Vendor Part #	Price
Magnetic Absolute Shaft Encoder (Analog Version, B-Option – ball bearing, 8-Option – 1/8" shaft)	US Digital	MA2-A-B8	$35.00
Cable Assembly (3-Wires, 26 AWG, 4-Pin Micro Connector to No Connector, 1FT length)	US Digital	CA-7941-1FT	$5.00
12V Battery	All Electronics Corp		$15.00
Lego Mindstorms Kit	VLSI Lab		

References

Bowling, Stephen. PIC18CXXX/PIC16CXXX DC Servomotor Application. Microchip Technology INC. 2002. 1-49. Nov.-Dec. 2006 <http://ww1.microchip.com/downloads/en/AppNotes/00696a.pdf>.

Bright, Tony, PHD. Personal Communication. 1 Dec. 2006

Strohm, Alan, and Kevin Chu. Inverted Pendulum. Harvey Mudd College. 2004. 1-15. Nov.-Dec. 2006 <http://www3.hmc.edu/~harris/class/e155/projects04/invertedpendulum.pdf>.

Appendix A: Final C code

```
/* PID_Controller.c:  Implements the PID controller for the
 *                    Microp's final project.
 * Authors:  Andrew Danowitz, Amir Adibi
 * Date:  November 2006
*/

#include <p18f452.h>
#include <timers.h>

unsigned char count = 0; //used to determine appropriate
                         //ref value
short dAD = 0;           //delta AD
short ADLast[2] = {0,0}; //previous AD values
short ADPres;            //latest AD value
unsigned char Saturated = 0;       //checks to see if
                                   //response is sat
unsigned char sign = 0;  //checks sign of pid
short Er_dAD = 0;        //dAD minus reference position
short Er_dADLast[2] = {0,0};
short kp;                //proportional constant
short ki;                //integration constant
short kd;                //derivative constant
short P=0;               //proportional result
short I=0;               //integral result
short D=0;               //derivative result
short PID=0;             //sum of P, I and D terms
short Integral=0;
short temp;
unsigned char PWM;       //output to lower 8 PWM bits

/* Function Prototypes */
void main(void);
void isr(void);

#pragma code low_vector = 0x18
void low_interrupt(void)
{
     _asm
          GOTO isr
     _endasm
}

#pragma code
void main(void)
{
```

```
	TRISA = 0xFF;
	TRISD = 0x00;
	TRISC = 0x00;
	PORTD = 0x00;

	//Set up timer 1
	INTCON = 0xF0; //enable interrups
	TMR0H = 0x00;  //set timer initial values
	TMR0L = 0x3D;
	T0CON = 0xC7;  //start the timer

	//Set up timer 2 for use with PWM
	T2CON = 0x07;

	kp = -35;
	ki = -6;
	kd = -1;
	//Initialize the PWM module
	CCP1CON = 0x0C;
	PR2 = 0xFF;

	CCPR1L =0xFF;
	CCP1CONbits.DC1B1 = 0;
	CCP1CONbits.DC1B0 = 0;

	//wait for timer interrupt
	while(1)
	{

	}

}

#pragma interruptlow isr
void isr(void)
{
	//reset the timer
	INTCON = 0xF0; //enable interrups
	TMR0H = 0x00;  //set timer initial values
	TMR0L = 0x3D;
	T0CON = 0xC7;  //start the timer

	//Configure the A/D converter to read motor
	ADCON1 = 0x00;
	ADCON0 = 0x8D;

	//Wait for valid A/D data
```

```
while(ADCON0bits.GO==1)
{
}

//Read A/D value
ADPres = ADRESH;

//if the voltage has only decreased this last //round,
//then we assume that the shaft has just completed a
//rotation and is still moving
//forward.
if (ADPres>=0x00F0 && ADLast[1]<=0x0010)
{
     //If the shaft did overflow, then the difference
     //between the new and old angle valuse is 0x100
     //plus the new eight bit angle value minus the
     //old angle value
     dAD = 0x0100+ADLast[1] - ADPres;
}

//If the shaft hasn't completed a revolution, the
//change in angle is just new minus old
else
{
     dAD = ADLast[1]-ADPres;
}

//Push the latest AD value into the stack
ADLast[0] = ADLast[1];
ADLast[1] = ADPres;

//Every two revolutions, the difference in AD values
//should be 15.

Er_dAD = dAD - 7; //calculate the error

P = kp * Er_dAD;    //P is just P times error

D = kd * (Er_dAD - Er_dADLast[1]); //D is just D times
                                   //change in error

if(Saturated != 1)
{
     Integral += Er_dAD; //The integral term is just
                         //the sum of the previous
     I = ki*Integral;    //errors
}
```

```
PID = P + I + D;

//Shifts in new error value
Er_dADLast[0] = Er_dADLast[1];
Er_dADLast[1] = Er_dAD;

//If the PID result is negative,
if ((PID & 0x8000) == 0x8000)
{
     //Set the sign bit and take the two's complement
     //of PID
     PORTDbits.RD1 = 1;
     PID = ~PID + 1;
}
else
     PORTDbits.RD1=0;

//Set up timer 2 for use with PWM
T2CON = 0x07;


//Initialize the PWM module
CCP1CON = 0x0C;
PR2 = 0xFF;

//Check to see if the PID is saturated.  For our
//purposes, this is saturated if the PID value leads
//to the PWM at full
if (PID >= 0x03FF)
{
     Saturated = 1;
     CCPR1L = 0xFF;
     CCP1CONbits.DC1B1=1;
     CCP1CONbits.DC1B0=1;

}

else
{
     Saturated = 0;
     //Put the lower 8 values of PID into PWM
     PWM = (PID>>2)&0xFF;

     //Put these values into the PWM module
     CCPR1L = PWM;
```

```
            //Put in the other two bits into PWM
            if(PID&0x0003 == 0x0003)
            {
                 CCP1CONbits.DC1B1 = 1;
                 CCP1CONbits.DC1B0 = 1;
            }

            else if(PID&0x0002 == 0x0002)
            {
                 CCP1CONbits.DC1B1 = 1;
                 CCP1CONbits.DC1B0 = 0;
            }

            else if(PID&0x0001 == 0x0001)
            {
                 CCP1CONbits.DC1B0 = 1;
                 CCP1CONbits.DC1B1 = 0;
            }

            else
            {
                 CCP1CONbits.DC1B1 = 0;
                 CCP1CONbits.DC1B0 = 0;
            }
       }

       //Display when it is saturated (not necessary, mainly
       //for testing and demo purposes
       PORTDbits.RD0 = Saturated;
  }
```

Appendix B: Completed Verilog for FPGA

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////
// Engineer: Andrew Danowitz, Amir Adibi
//
// Create Date:    23:43:43 11/19/2006
// Description: This code takes in the magnitude and sign
//              bit from the PID controller and splits it
//              into a two channel output to control the
//              motor's directionality.
//////////////////////////////////////////////////////////////

module channel_changer(sign, magnitude, motor_in,
motor_out);

     input sign;
     input magnitude;
     output reg motor_in;
     output reg motor_out;

     always @ ( * )
          if (sign)
          begin
               motor_in  <= 0;
               motor_out <= magnitude;
          end

          else
          begin
               motor_in <= magnitude;
               motor_out <= 0;
          end

endmodule
```

Appendix C: Synthesized Verilog Top Module