BALANCING A BALL AND BEAM WITH PID

Dr. Pranav Bhounsule
pranav.bhounsule@utsa.edu
Mentor
The University of Texas at San Antonio
Department of Mechanical Engineering
RAM Lab

Geoffrey Chiou
Programming
The University of Texas at San Antonio
Department of Mechanical Engineering
RAM Lab
geoffrey.chiou@gmail.com

Andy Plascencia
Hardware and Circuit Design
The University of Texas at San Antonio
Department of Mechanical Engineering
RAM Lab
andy.plascencia@thewheeldeal.org

Tyler Rowe
Systems Modelling
The University of Texas at San Antonio
Department of Mechanical Engineering
RAM Lab
tyler.rowe@thewheeldeal.org

ABSTRACT

This project centers around the application of PID theory and how this theory can be applied to model and predict the movement of a ball on a beam. The PID system provides feedback and response to the system in order to allow the control system to automatically adjust itself with the goal being to balance a ball on a beam and always have the ball return to the center. This is accomplished by gathering the distance from an ultrasonic sensor and using PID theory to predict the appropriate K, Kp, and Kd values for the system to correctly adjust itself. After attempting to predict the values using PID theory, the best results were instead obtained using trial and error. Through trial and error, the team was able to successfully identify the appropriate values and build a PID control system that balanced a ball on a beam.

INTRODUCTION

The goal of this project was to explore the application of using a proportional-integral-derivative (PID) controller to balance a ball at the center of a beam. The purpose of the project was to build a robotic platform that will allow a user to balance a ball on a beam by altering the tuning parameter (Kp, Ki, and Kd) values of a PID controller based off of the visual performance of the ball. Using an Arduino Mega along with a stepper motor and various electrical components, the robot was constructed out of 3-D printed parts printed using an Ultimaker. The robot has the ability to provide variable tuning parameters which are controlled using three different linear potentiometers. To successfully complete construction of the robot, a theoretical model of the system was modeled using equations of motion and the transfer function was analyzed using MATLAB to find the theoretical tuning parameters. Next, the robot was modeled using SolidWorks to create CAD files for 3-D printing. After the robot was assembled, software was written in C++ to control the robot. The end result of the project was a fully functional experiment where a user could practice tuning PID controllers visually.

THEORY

The purpose of this study was to demonstrate how feedback influences the response of a control system by constructing a PID controller. A control system is a device that manages and regulates the behavior of a particular system. The system in our case is the ball balance beam. In this system, a beam must be able to balance a ball and return the ball to the center of the beam if moved.

There are two different types of systems, open loop and closed loop. Open loop systems have no feedback and are simply based on the input while closed loop systems use feedback to improve the control of the system. The principle being tested by this experiment is how feedback can improve the system. This is achieved by the construction of a PID controller. The PID controller is a type of feedback controller where P stands for proportional, I for integral, and D for derivative. The “P” term is responsible for producing an output value that is proportional to the current error that the system is calculating. The “I” term is essentially the sum of all the instantaneous error and is responsible for eliminating the steady state error that the P controller would produce. The “D” term is responsible for predicting the future system behavior and using the predicted behavior to improve the time and stability of the system.

When all three parts are put together in one controller, the system should be able to achieve a steady state of oscillation. The

Copyright © 2016 by RAM Lab

output of the system is reliant of each part of the PID controller and each term has a corresponding gain that accompanies it. The "P" produces a proportional gain and this value has a direct result on response speed and percent overshoot. The "I" produces and integral gain, which when combined with the proportional gain can produce an increase in rise time and settling of the system. The derivative gain that is produce by the "D" term will help to decrease the overshoot value and settling time.

HARDWARE AND CIRCUIT DESIGN

The ball balancing mechanism was designed using SolidWorks modeling software. The figure below shows the assembled model with the actuator and feedback sensor. The mechanism consists of an 18 inch beam connected to a base and stand which allows the beam to rotate in the X Y plane. The stepper motor controls the angle displacement of the beam by rotating the 3 inch lever connected to the link, which is used to control the position of the ball.

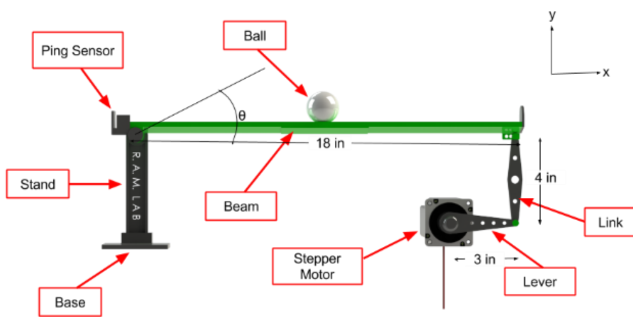

Figure 1. Ball Balancing Robot Schematic

The hardware used for this project is shown below in the wiring diagram.

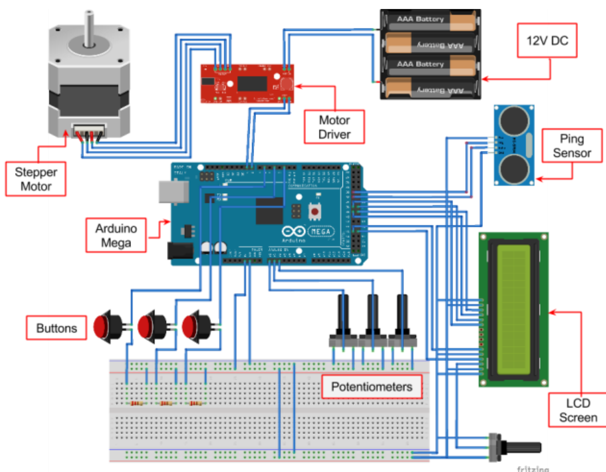

Figure 2. Wiring Diagram

A stepper motor was used to change the angle displacement of the beam. A motor driver was used to provide the actuator with 12 volts using an external power supply. An ultrasonic ping sensor provided feedback on the position of the ball. The potentiometers controlled the PID K values which affect the transient response of the system. The Kp, Ki and Kd values are displayed on the LCD screen. The buttons are used to adjust the angle of the beam by controlling the motor manually and also engage the control program. The Arduino Mega is the microcontroller used to run the program.

SOFTWARE DESIGN

The software for the ball balancing robot, found in the Appendix, was written in C++ and compiled using avr-gcc for an Arduino microcontroller. The software allows a user to set the zero points for the stepper motor, change tuning parameters, and watch as the robot tries to balance the ball on the beam with those parameters. If the user is not satisfied with the results, a reset button could be used and the robot would stop and let the user change their tuning parameters. All of this was performed while a LCD screen displayed the tuning parameters and the current mode of the robot. These features for the robot were accomplished through the use of several third party libraries which included the Arduino PID library, the Liquid Crystal library, and the Stepper library.

The software started with several define statements for the values of some of the pins used by the electronic components. There was also a structure declaration which stored several key values used throughout the program. Some of the functions include in the program included a function to display the current tuning parameters and mode, a function to receive a distance reading from the ultrasonic sensor, a function to read the tuning parameters from the linear potentiometers, functions to control the buttons, functions to control the stepper motor, a function to control the modes, and a PID mode function for when the controller is started.

The main function of the program consisted of several calls to pinMode and digitalWrite to set the state of some of the pins. This section also contained some code to initialize the LCD screen. A declaration for the structure mentioned earlier was made, and this structure was passed by reference to an infinite for loop running the control function.

SYSTEM MODELING

The system being modeled consists of a flat 3D printed beam attached to a servomotor through a series of 3D printed gears forming a track upon which a ball is free to roll. One end of the beams is coupled to the servomotor through a lever arm and gears and the other end is fixed. A free body diagram representing the system can be seen below. These equations were derived using sources provided by the University of Michigan and the Milwaukee School of Engineering, located in the references.

Copyright © 2016 by RAM Lab

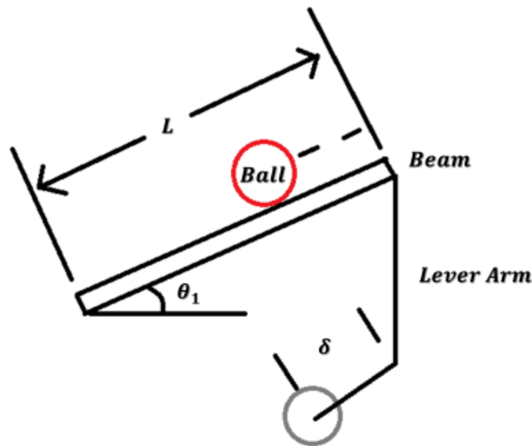

Figure 3. Free Body Diagram of System

In this free body diagram the system parameters are as follows:

$$m = mass\ of\ ball$$
$$R = radius\ of\ ball$$
$$\delta = offset$$
$$L = length\ of\ beam$$
$$x = ball\ position$$
$$\theta_1 = beam\ angle$$
$$\theta_2 = gear\ angle$$

In this system, the equations of motion can be used to solve for the relationship between the angle of the beam and the ball movement. The equations of motion can be developed into the following form.

$$M\frac{\delta^2 x(t)}{\delta t^2} + C\frac{\delta x(t)}{\delta t} + kx(t) = F(t)$$
$$M\frac{\delta^2 X(t)}{\delta t^2} + C\frac{\delta X(t)}{\delta t} + kX(t) = F(t)$$
$$M\ddot{X} + C\dot{X} + kX = 0$$

From this equation, it is possible to solve for M and for theta. The reason for doing this is to develop a relationship between the movement of the ball and the angle of the beam. Once this relationship is developed, it is possible to code, model, and predict the position of the ball based on the angle of the beam. The next step in the system modeling is to solve for M. This figure is shown below.

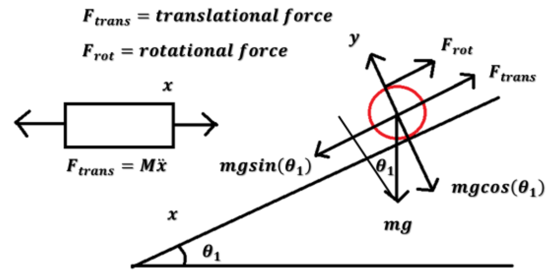

Figure 4. Reaction Forces on System

$$F_{rot} = \frac{Tr}{R} = \frac{J}{R}\frac{\delta w_b}{\delta t} = \frac{J}{R}\frac{\delta\left(\frac{V_b}{R}\right)}{\delta t}$$
$$F_{rot} = \frac{\frac{J}{R}\delta^2\left(\frac{x}{R}\right)}{\delta t^2} = \frac{J}{R^2}\ddot{X}$$
$$F_x = F_{trans} + F_{rot} = m\ddot{x} + \frac{J}{R^2}\ddot{x} = \left(m + \frac{J}{R^2}\right)\ddot{x}$$
$$M = m + \frac{J}{R^2}$$

After solving for M, it is necessary to solve the angle of the beam. This can be done by equating the value for M to the force of the ball and gravity. This is shown below.

$$\left(m + \frac{J}{R^2}\right)\ddot{x} = -mgsin(\Theta_1)$$
$$\left(m + \frac{J}{R^2}\right)\ddot{x} + mgsin(\Theta_1) = 0$$
$$mgsin(\Theta_1) = mg\Theta_1$$
$$Equation\ of\ Motion:$$
$$\boxed{\left(m + \frac{J}{R^2}\right)\ddot{x} + mg\Theta_1 = 0}$$

$$J_{sphere} = \frac{2}{5}mR^2$$
$$m + \frac{J}{R^2} = m + \frac{\frac{2}{5}mR^2}{R^2} = m + \frac{2}{5}m = \frac{7}{5}m$$
$$\left(\frac{7}{5}m\right)\ddot{x} = mg\Theta_1$$
$$\frac{7}{5}\ddot{x} = g\Theta_1$$
$$\ddot{x} = \frac{5}{7}g\Theta_1$$
$$Since\ \Theta_1 = \frac{d}{L}\Theta_2$$
$$\left(m + \frac{J}{R^2}\right)\ddot{x} + mg\frac{d}{L}\Theta_2 = 0$$
$$\boxed{\left(m + \frac{J}{R^2}\right)\ddot{x} = -mg\frac{d}{L}\Theta_2}$$

Copyright © 2016 by RAM Lab

Taking the Laplace transform and rearranging the equation gives:

$$\left(m + \frac{J}{R^2}\right) R(s)s^2 = -\frac{mgd}{L}\Theta(s)$$

$$\boxed{P(s) = \frac{R(s)}{\Theta(s)} = -\frac{mgd}{L\left(m + \frac{J}{R^2}\right)} \frac{1}{s^2} \left[\frac{m}{rad}\right]}$$

PID MODELING

Using the transfer function of the system, the tuning parameters of the PID were estimated using MATLAB. First, the following parameters were used:

$$mass\ of\ ball = 0.0027\ kg$$
$$radius\ of\ ball = 0.02\ m$$
$$lever\ arm\ offset = 0.0254\ m$$
$$gravitational\ acceleration = 9.71\frac{m}{s^2}$$
$$length\ of\ beam = 0.4318\ m$$
$$Ball's\ Moment\ of\ Inertia = 0.207 \times 10^{-7}\ kgm^2$$

Setting "s" as the variable for a transfer function in MATLAB, the transfer function determined earlier was entered as an equation.

```
m = 0.0027; % kg
R = 0.02; % m
d = 0.0508; % m
g = -9.81; % m/s^2
L = 0.4318; % m
J = 6.207e-7; % kg*m^2


s = tf('s');
P_ball = -m*g*d/L/(J/R^2+m)/s^2;
```

The previous code was run to load the variables into memory. Using the PID tuning application built into MATLAB, the transfer function "P_ball" was selected and imported as a Plant Model. Next, the design mode was set to the time domain and the type of control was switched to PID. Since a low settling time was desired, the response time ticker was set to 0.24 seconds and the transient behavior was set to a more robust value of 0.81 to reduce overshoot. The step response plot can be found in the figure below.

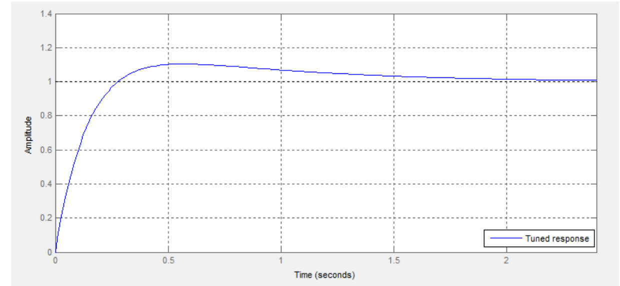

Figure 5. Theoretical Step Response Plot

Using these settings, the software suggested using the following tuning parameters:

$$K_P = 13.4675$$
$$K_I = 1.6649$$
$$K_D = 10.7311$$

Using these tuning parameters would theoretically result in the following performance values:

$$Rise\ Time = 0.205\ s$$
$$Settling\ Time = 1.91\ s$$
$$Overshoot = 10.4\ \%$$

The root locus of the system was calculated in MATLAB with the command:

```
rlocus(P_ball)
```

The output is shown below.

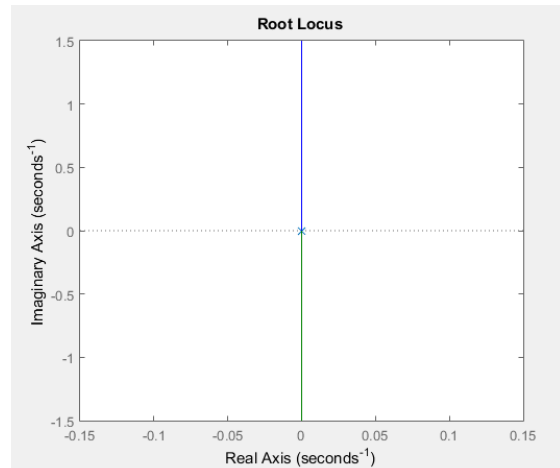

Figure 6. Initial Root Locus Plot

Using the settling time of 0.205 seconds and a percent overshoot of 10.4 percent calculated earlier, the root locus was replotted. The damping ratio and natural frequency are calculated below using those parameters.

Copyright © 2016 by RAM Lab

$$\%OS = 100e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}}$$

$$\zeta = \sqrt{\frac{\left(\ln\left(\frac{\%OS}{100}\right)^2\right)}{\pi^2 + \left(\ln\left(\%\frac{\%OS}{100}\right)^2\right)}}$$

$$\boxed{\zeta = 0.5847}$$

$$T_s = \frac{4}{\zeta\omega_n}$$

$$\omega_n = \frac{4}{\zeta T_s} = \frac{4}{(0.5847)(0.205)}$$

$$\boxed{\omega_n = 3.5815\ Hz}$$

Using the damping ratio and natural frequency, calculated above, the commands below generated a new root locus plot with a constant damping ratio and natural frequency. A figure of the plot is shown below.

```
zeta = ((log(POS/100)^2) / (3.14^2 +
log(POS/100)^2))^(1/2);
Wn = 4/(zeta*Ts);
sgrid(zeta, Wn);
axis([-5 5 -2 2]);
```

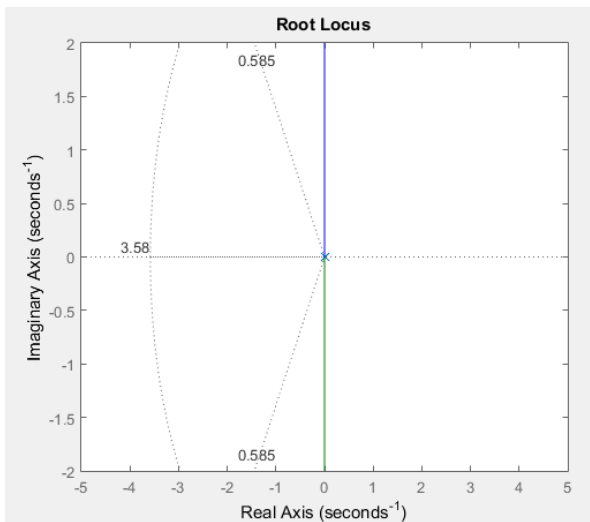

Figure 7. Root Locus Plot with Bounds

Since the root locus did not fit within the bounds of the design specifications calculated earlier, a lead compensating controller was added to shift the root locus to the left. A zero frequency of 0.01 and a pole frequency of 5 was used in the code shown below.

```
z0 = .01;
p0 = 5;

C = tf([1 z0], [1 p0]);

rlocus(C*P_ball)
sgrid(0.70, 1.9)
```

This generates a root locus graph that is within the bounds of the design specifications. The plot of the root locus is shown below.

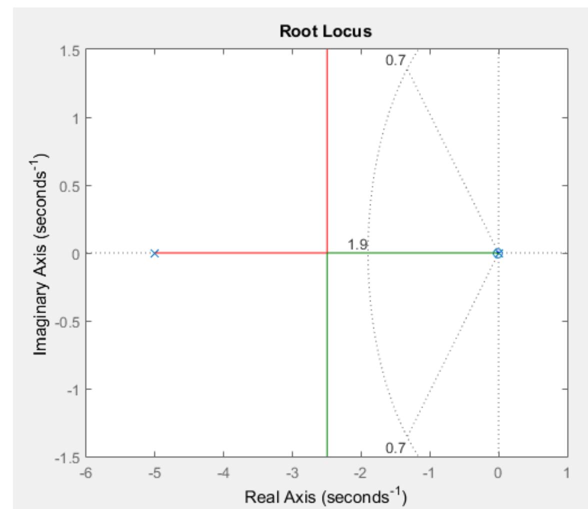

Figure 8. Designed Root Locus Plot

Using this new root locus graph, a gain was calculated using the following code.

```
[k, poles] = rlocfind(C*P_ball)
sys_cl = feedback(k*C*P_ball,1);
t = 0:0.01:5;
step(0.25*sys_cl,t)
```

The gain (k = 8.5616) was used to plot a theoretical step response curve shown below.

Copyright © 2016 by RAM Lab

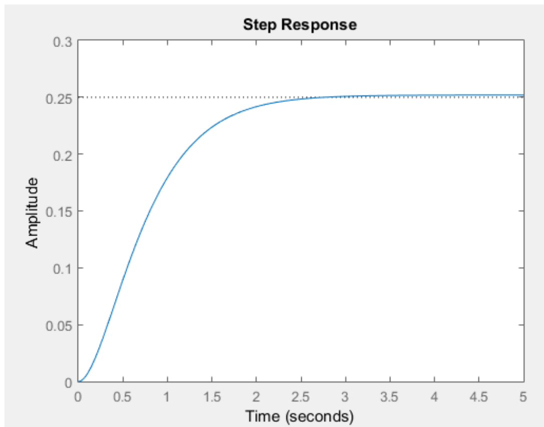

Figure 9. Step Response Based Off of Root Locus

A block diagram of the plant was constructed in Simulink to model the control system based off of the Lagrangian equation of motion for the ball. These block diagrams were derived from a University of Michigan tutorial located in the references. The block diagram, built using Simulink and modeled directly from the equation of motion, can be found in the figure below.

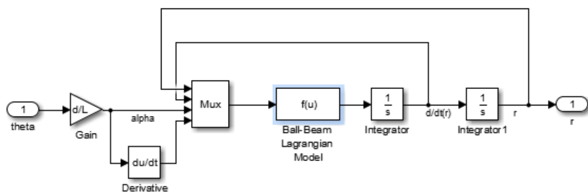

Figure 10. Block Diagram of Plant

The lead compensator controller for the model was added using the transfer function $\frac{s+0.01}{s+5}$ and gain of 8.5616. The block diagram with the control and plant is shown in the figure below.

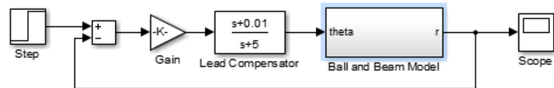

Figure 11. Block Diagram of Lead Compensating Controller and Plant

When the simulation is ran in Simulink with a step input of 0.25, the following plot shown below is generated. This plot matches the step response found earlier.

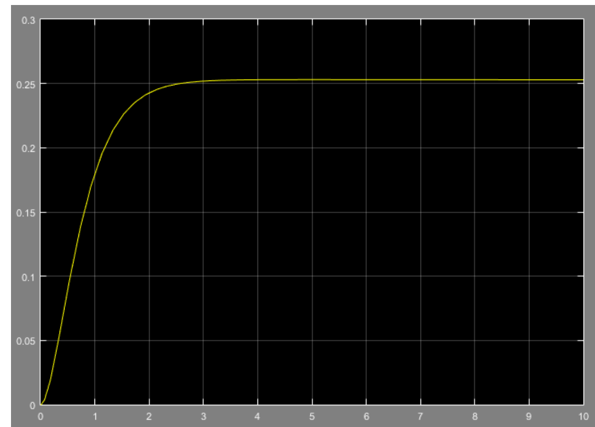

Figure 12. Response of Block Diagram Simulation

RESULTS

The robot was started using the tuning parameters simulated with MATLAB. The results of the run did not follow the theoretical predictions. The ball was not close to balancing, instead, it oscillated back and forth. Since the theoretical model did not work, an attempt was made to tune the robot using visual feedback of the ball and trial and error. After several trials, the best tuning parameters appeared to be the following values:

$$K_p = 5$$
$$K_i = 235$$
$$K_d = 88$$

Using these tuning parameters, the ball was able to stay to be balanced in the middle. Sometimes the ball would stop balancing since the ultrasonic sensor had occasional noisy readings.

CONCLUSION

A robot which is able to balance a ball on a beam using PID control with the ability to rapidly change tuning parameters was designed and constructed. The results of the physical experiment did not follow the theoretical predictions made using MATLAB. Instead, the tuning parameters were determined manually using visual feedback from the person tuning. Future work for this project can include adding a filter to the ultrasonic sensor to reduce noise. Another suggestion would be to use a Raspberry Pi as the microcontroller instead of an Arduino Mega. An attempt was made to publish the input data to the serial monitor for live plotting, but the serial buffer filled up to fast and the robot slowed to a crawl. A Raspberry Pi has more processing power and would be able to live plot since the controller and OS do not need to be linked through a serial port.

Copyright © 2016 by RAM Lab

REFERENCES

[1] "PID Tuning." - MATLAB. Web. 12 May 2016.

[2] "Arduino - LiquidCrystal." Arduino - LiquidCrystal. Web. 12 May 2016.

[3] "Arduino - Stepper." Arduino - Stepper. Web. 12 May 2016.

[4] "PID Library." Arduino Playground. Web. 12 May 2016.

[5] "Control Tutorials for MATLAB and Simulink -." Control Tutorials for MATLAB and Simulink -. Web. 12 May 2016.

[6] "Control Tutorials for MATLAB and Simulink -." Control Tutorials for MATLAB and Simulink -. Web. 12 May 2016.

[7] "BALL AND BEAM DESIGN PROJECT." EE-371 CONTROL SYSTEMS LABORATORY. Dr. Hadi Saadat. Web. 12 May 2016.

[8] "PID Controller." Wikipedia. Wikimedia Foundation. Web. 12 May 2016.

Copyright © 2016 by RAM Lab

APPENDIX A

PID ROBOT SOFTWARE

```
#include <LiquidCrystal.h>
#include <PID_v1.h>
#include <Stepper.h>

#define STEPPER_DIR     8
#define STEPPER_STEP    9
#define RIGHTBUTTON     6
#define LEFTBUTTON      4
#define STARTBUTTON     2
#define ECHOPIN         30
#define TRIGPIN         32
#define DELAY           500
#define POTKP           A0
#define POTKI           A1
#define POTKD           A2
#define LCD_RS          47
#define LCD_ENABLE      46
#define LCD_D4          36
#define LCD_D5          37
#define LCD_D6          38
#define LCD_D7          39

LiquidCrystal lcd(LCD_RS, LCD_ENABLE, LCD_D4, LCD_D5, LCD_D6, LCD_D7);

struct Status
{
    int state;
    int position;
    int setpoint;
    int Kp;
    int Ki;
    int Kd;
};
```


Copyright © 2016 by RAM Lab

```
void display(struct Status *status)
{
    lcd.clear();
    lcd.setCursor(0, 0),
    lcd.print("Kp: ");
    lcd.setCursor(4, 0);
    lcd.print(status->Kp);
    lcd.setCursor(8, 0);
    lcd.print("Ki: ");
    lcd.setCursor(12, 0);
    lcd.print(status->Ki);
    lcd.setCursor(0, 1);
    lcd.print("Kd: ");
    lcd.setCursor(4, 1);
    lcd.print(status->Kd);
    lcd.setCursor(8, 1);
    lcd.print("M: ");
    lcd.setCursor(11, 1);

    if (status->state == 2)
    {
        lcd.print("PID");
    }

    else if (status->state == 1)
    {
        lcd.print("TUNE");
    }

    else
    {
        lcd.print("ZERO");
    }

    delay(10);
}
double ping(void)
```


Copyright © 2016 by RAM Lab

```
{
    double distance;
    digitalWrite(TRIGPIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIGPIN, HIGH);
    delayMicroseconds(2);
    digitalWrite(TRIGPIN, LOW);
    distance = pulseIn(ECHOPIN, HIGH, 6000) / 58.2;
    if (distance == 0 || distance > 45)
    {
        ping();
    }
    else
    {
        return distance;
    }
}

void readKValues(struct Status *status)
{
    if (status->state == 0 || status->state == 1)
    {
        status->Kp = analogRead(POTKP);
        status->Ki = analogRead(POTKI);
        status->Kd = analogRead(POTKD);
        status->Kp = map(status->Kp, 0, 1023, 0, 300);
        status->Ki = map(status->Ki, 0, 1023, 0, 300);
        status->Kd = map(status->Kd, 0, 1023, 0, 300);
        display(status);
    }
    else
    {
        display(status);
    }
}
void rightButton(struct Status *status)
{
```


Copyright © 2016 by RAM Lab

```
    if (status->state >= 1)
    {
        if (status->position <= 150)
        {
            turnRight();
            status->position++;
        }
        else {
            stop();
        }
    }
    else
    {
        turnRight();
    }
}
void leftButton(struct Status *status)
{
    if (status->state >= 1)
    {
        if (status->position >= -150)
        {
            turnLeft();
            status->position--;
        }
        else {
            stop();
        }
    }
    else
    {
        turnLeft();
    }
}
void PIDMode(struct Status *status)
{
    double input;
```


Copyright © 2016 by RAM Lab

```
    double output;
    double setpoint;
    double Kp;
    double Ki;
    double Kd;
    status->position = 0;
    Kp = (double) status->Kp;
    Ki = (double) status->Ki;
    Kd = (double) status->Kd;
    setpoint = (double) status->setpoint;
    PID myPID(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);
    myPID.SetMode(AUTOMATIC);
    for(;;)
    {
        if (digitalRead(STARTBUTTON) == HIGH)
        {
            status->state = 0;
            break;
        }
        else
        {
            input = ping();
            myPID.Compute();

            if (output >= 127)
            {
                if (status->position <= 100)
                {
                    turnRight();
                    status->position++;
                }
            }
            else if (output < 127)
            {
                if (status->position >= -100)
                {
                    turnLeft();
```


Copyright © 2016 by RAM Lab

```
                    status->position--;
                }
            }
        }
    }
}

void control(struct Status *status)
{
    readKValues(status);
    if (status->state == 2)
    {
        PIDMode(status);
    }
    if (digitalRead(STARTBUTTON) == HIGH)
    {
        if (status->state == 0)
        {
            status->state = 1;
            status->position = 0;
            delay(500);
        }
        else if (status->state == 1)
        {
            status->state = 2;
            status->position = 0;
            delay(500);
        }
    }

    if (digitalRead(RIGHTBUTTON) == HIGH)
    {
        rightButton(status);
    }

    if (digitalRead(LEFTBUTTON) == HIGH)
    {
```


Copyright © 2016 by RAM Lab

```
        leftButton(status);
    }

    else
    {
        stop();
    }
}

void turnRight(void)
{
    digitalWrite(STEPPER_DIR, HIGH);
    digitalWrite(STEPPER_STEP, HIGH);
    delayMicroseconds(DELAY);
    digitalWrite(STEPPER_STEP, LOW);
    delayMicroseconds(DELAY);
}

void turnLeft(void)
{
    digitalWrite(STEPPER_DIR, LOW);
    digitalWrite(STEPPER_STEP, HIGH);
    delayMicroseconds(DELAY);
    digitalWrite(STEPPER_STEP, LOW);
    delayMicroseconds(DELAY);
}



void stop(void)
{
    digitalWrite(STEPPER_DIR, LOW);
    delayMicroseconds(500);
    digitalWrite(STEPPER_STEP, LOW);
    delayMicroseconds(500);
}

int main(void)
```


Copyright © 2016 by RAM Lab

```
{
    init();
    pinMode(STEPPER_DIR, OUTPUT);
    pinMode(STEPPER_STEP, OUTPUT);
    digitalWrite(STEPPER_DIR, LOW);
    digitalWrite(STEPPER_STEP, LOW);
    pinMode(RIGHTBUTTON, INPUT);
    pinMode(LEFTBUTTON, INPUT);
    pinMode(STARTBUTTON, INPUT);
    pinMode(TRIGPIN, OUTPUT);
    pinMode(ECHOPIN, INPUT);
    lcd.begin(16, 2);
    lcd.clear();
    lcd.noCursor();
    lcd.noBlink();
    lcd.noAutoscroll();
    // state 0 = free adjust, state 1 = locked adjust, state 2 = running.
    struct Status status;
    status.state = 0;
    status.position = 0;
    status.setpoint = 21;
    for(;;) {
        control(&status);
    }
    return 0;
}
```


Copyright © 2016 by RAM Lab