Grafton Cook

Joe Tobbe

Project 2: Ready, Get Set, Go!

## Introduction

The purpose of the second project is to create a robot which drives 10 feet in a straight line. The Parallax BOE Bot chassis fitted with servo motor wheels, optical encoders, and an Arduino was used to meet the requirements. To meet the distance and straightness criteria, the hardware required careful calibration, proper system testing, and a robust control system.

|  |
| --- |
|  |
| **Figure 1.** BOE Bot chassis and Arduino used in project two. |

First, the physical system was modeled and analyzed. Next, a proportional-plus-integral-plus-derivative (PID) controller was designed to control the speed of the right servo motor with respect to the left servo motor speed. Finally, the control was implemented and fine-tuned to ensure no tracking error along the path of travel.

## Physical System and Model

### Servo motor and optical encoder

The chassis was primarily driven by two Parallax #900-00008 continuous-rotation servo motors. The motor speed was monitored by Parallax #28107 Rev. B optical encoders.

### System analysis

The movement of the system is dependent on the properties of the of the servo motor. The servo motor is a first-order system: represented in block diagram format in Figure 2.

|  |
| --- |
| C:\Users\212454356\AppData\Local\Microsoft\Windows\INetCache\Content.Word\block diagram.png |
| **Figure 2.** Servo motor system in block-diagram format. |

Thus the undisturbed transfer function can be represented by the equation shown in Equation 1.

|  |  |
| --- | --- |
|  | [1] |

The first step in modeling the system was to find the value of the coefficient. First, the time for one window to pass the optical encoder was determined by stepping through inputs to the writeMicroseconds() Arduino function. As illustrated in Figure 2, this quickly showed the speed of the encoder wheel relative to writeMicroseconds() input.

|  |
| --- |
|  |
| **Figure 2.** Window-passing times as a function of writeMicroseconds() input |

The coefficient is the slope of the plot of speed with respect to writeMicroseconds() input and therefore the inverse of the slope of time as a function of writeMicroseconds() input. Therefore, finding the average curve fit of the linear regions around the center value was found.

|  |
| --- |
|  |
| **Figure 3.** Linear Region of window-passing-time plot with linear curve fit. |

Both linear regions were plotted, fitted, and averaged. Because speed and time are inversely proportional, the coefficient was determined by Equation 2:

|  |  |
| --- | --- |
|  | [2] |

To identify τ, a program was written to cumulate the time the system took to reach steady state, and fitted to an equation modeling exponential decay. (Equation 3.)

|  |  |
| --- | --- |
|  | [3] |

Thus, the right-hand servo motor transfer function is shown in Equation 4.

|  |  |
| --- | --- |
|  | [4] |

## PID Controller

### Original System

To create an effective control scheme, the step response of the original system was analyzed in SISOtool, a feature of the Control System Toolbox found in Mathworks’ MATLAB. The step response of system is shown in Figure 4. While the response time of is adequate, its peak amplitude of 0.195­ shows an error greater than 80%.

|  |
| --- |
|  |
| **Figure 4.** r to y plot of untuned system . |

### PID coefficient selection

To reduce iteration and simplify the control scheme, MATLAB’s PID Auto Tune tool was used to create PD, then PI controllers. These controllers were not able to correct serious steady-state errors manifesting as a long leftward arc of travel which was especially pronounced between eight- and 10 feet in the robot’s path. To correct for long-term tracking error, a PD system was automatically created, then augmented with a derivative term. This control scheme resulted in a steady-state error of less than three inches of lateral travel over 10 feet of distance. The final controller equation is shown in Equation 5.

|  |  |
| --- | --- |
|  | [5] |

### Final system

The tuned response is shown in Figure 5. The 2% settling time is just over 57 milliseconds with no steady-state error.

|  |
| --- |
|  |
| **Figure 5.** Tuned transient and steady-state response of the system. |

Additionally, the disturbance plot reaches a maximum amplitude of 0.8 before trending to zero, as illustrated in Figure 6.

|  |
| --- |
|  |
| **Figure 6.** Tuned du-to-y response of the system. |

## Conclusion

The objective of the project was to create a robot that traveled 10 feet in a straight line. After modeling and analysis of the reactive servo motor, a proportional-plus-integral-plus-derivative (PID) controller was designed and tuned to control the servo motor. This controller drastically reduced the steady-state error of the system, allowing the robot to meet performance requirements with a high degree of accuracy.

## Appendix A: Production Code

#include <Servo.h>

#include <PID\_v1.h>

// create servo object to control a servo

Servo right\_servo;

Servo left\_servo;

// pin definition

int right\_encoder\_pin = 10;

int left\_encoder\_pin = 9;

int right\_servo\_pin = 11;

int left\_servo\_pin = 12;

// servo center values & speeds

int right\_center\_value = 1469;

int left\_center\_value = 1491;

// encoder counter and desired travel distance

volatile int cc\_left;

// = desired distance ft \* (12 in/ft \* 64 encoder\_changes/rotation / 8 in/rotation)

int distance = 10\*(12\*64/8); // = # of 0.125" w fine encoder wheels in 10 feet

// PID variables & initialization

double dt; // time difference between encoders

double left\_spd = 150; // speeds determined by turn\_around\_micro.ino

double right\_spd = 132;

double desired\_dt = 0; // desired time difference between encoders

PID myPID(&dt, &right\_spd, &desired\_dt, 0.084457214, 7.712170356, 0.014842, DIRECT);

void setup() {

Serial.begin(9600); // initialize USB communication

myPID.SetMode(AUTOMATIC); //turn on PID

myPID.SetOutputLimits( right\_spd - 30, right\_spd + 30 );

pinMode(right\_encoder\_pin, INPUT\_PULLUP);

pinMode( left\_encoder\_pin, INPUT\_PULLUP);

attachInterrupt( left\_encoder\_pin, left\_counter, CHANGE);

attach\_servos(1);

orient\_encoders();

delay(1000);

}

void loop() {

cc\_left = 0;

while (cc\_left < distance) {

drive(right\_spd, left\_spd);

dt = read\_encoders();

Serial.print(cc\_left); Serial.print(" ");

Serial.print(dt); Serial.print(" ");

Serial.print(right\_spd); Serial.println(";");

myPID.Compute();

}

drive(0, 0);

delay(1000); // orient encoders, then delay one second

attach\_servos(0);

}

long read\_encoders() {

unsigned long t\_right\_pass, t\_left\_pass;

long delta\_t\_pass;

int right\_value = digitalRead(right\_encoder\_pin);

while (digitalRead(right\_encoder\_pin) == right\_value) { }

t\_right\_pass = millis();

while (digitalRead(right\_encoder\_pin) != right\_value) { }

t\_right\_pass = millis() - t\_right\_pass;

int left\_value = digitalRead(left\_encoder\_pin);

while (digitalRead(left\_encoder\_pin) == left\_value) { }

t\_left\_pass = millis();

while (digitalRead(left\_encoder\_pin) != left\_value) { }

t\_left\_pass = millis() - t\_left\_pass;

return delta\_t\_pass = t\_left\_pass - t\_right\_pass;

}

void orient\_encoders() {

int right\_value = digitalRead(right\_encoder\_pin);

drive(10 - 20\*right\_value, 0);

while (digitalRead(right\_encoder\_pin) == right\_value) { }

drive(0, 0);

int left\_value = digitalRead(left\_encoder\_pin);

drive(0, 20\*left\_value - 10);

while (digitalRead(left\_encoder\_pin) == left\_value) { }

drive(0, 0);

}

void attach\_servos(int ats) {

if ( ats == 0) {

right\_servo.detach(); left\_servo.detach();

}

else {

right\_servo.attach(right\_servo\_pin); left\_servo.attach( left\_servo\_pin);

}

}

void drive(double right\_speed, double left\_speed) {

right\_servo.writeMicroseconds(right\_center\_value - right\_speed);

left\_servo.writeMicroseconds( left\_center\_value + left\_speed);

}

void left\_counter() { cc\_left++; }