Velocity Control of Car Wheels

Garrett Wilson and Nathan Zimmerly

ENGR 352

June 8, 2016

Contents

1 Purpose **3**

2 Components **3**

3 Wiring **3**

4 PID Controller **3**

5 Results **4**

6 Future Work **4**

7 References **4**

1 Purpose

The purpose of this project was to control the velocity of a car by using a PID controller. This would be useful for having the car move at a constant speed at a variety of user-settable speeds, instead of just the maximum velocity. To determine the speed of the car, we used an optical encoder on each motor. Our test was setting the two wheels to go a certain speed, placing the car on the ground, and seeing if it traveled straight.

2 Components

- Robotic car kit
- L298N Dual H-Bridge Motor Controller
- TI LaunchPad MSP430

3 Wiring

- ENA (PWM left) – P2.4
- ENB (PWM right) – P2.5
- IN1 – P8.1
- IN2 – P8.2
- IN3 – P2.3
- IN4 – P3.7
- Left sensor – P1.2
- Right sensor – P1.3

4 PID Controller

A PID controller uses the a proportional, integral, and derivative term to control a system. In PID controllers, increasing the proportional term decreases the rise time, the integral term reduces steady state error, and the derivative term improves settling time and the stability of the system. We used the PID controller because it is simple and can often work well. By adapting Professor Frohne's code to work without Bluetooth and for our own H-Bridge, we were able to get a PID controller to run based on Control Tutorials.

To tune the PID controller, we started with only the proportional term, and increased it until there was a slight oscillation in the motor. Once we found the proportional term, we cut it in half and increased the integral term until the motors started oscillating again. After that we added a derivative term and adjusted it until we had good results. We found that our integral sum terms would increase to such a high magnitude that it would take a long time to undo. Because of this, we set a limiting magnitude for the term. The tuning was difficult because using the *printf* function in Code Composer Studio (CCS) would cause the microcontroller to lag which affected our controller negatively. Because of this we had to tune the PID controller simply by looking at the wheels spin instead of looking at graphs. To help with debugging, we set an LED to toggle on interrupts from each optical encoder.

We tried changing how the PID controller worked several times. One time we tried adding a set point into the control signal, and then removed it after talking with Frohne and realizing that the error won't actually go to zero when it's going the desired speed. Also, we allow negative control signals which turn the wheels in the opposite direction instead of cutting the signal off at zero. This

allowed for the motors to spin backwards, but that did not improve our results. We also had the option to use the H-Bridge to brake our motors, but we chose not to do this because when we simply turned off the motors, the car slowed down fast enough due to friction. Frohne's code had a few issues including resetting the integral sum terms to zero every interrupt, so we made the terms static so that they would not reset. Also, we limited the control output to be between 0 and 100 since that is the range for our PWM output. However, between 0 and 30 the PWM output was not strong enough to make the motor spin.

5 Results

We were able to tune the wheels individually to be a fairly constant speed, but once we ran them together they would oscillate on and off randomly. Also, when the sensor took a reading inbetween the open and closed points on the code track, the microcontroller would read values that the motors were spinning much faster than is possible. At a certain point the car would zig-zag across the floor in a fairly straight line, but that was the best result we had. Another group had similar results as us and ended up using one wheel to set the speed and a PID controller on the other wheel to match it. This method eliminated the problem of both sensors not being able to run properly off the microcontroller. We are not sure why both sensors do not work at the same time. A possibility is that the microcontroller cannot keep up with both at a fast enough rate to make the sensors input accurate. We used an oscilloscope to look at the sensor output and it looked as expected, so we believe the issue is with the microcontroller.

6 Future Work

- We encountered issues debugging since we tried having printf statements while it was running, which changed how the controller acted since sending those over serial was too slow. Thus, Frohne recommended we save the values into an array and then after the controller runs for a bit we then print out to the console.
- Something was wrong with our measurements. Michael and Chris also ran into this, and they thought it had to do with both wheels operating at the same time. They didn't have the issues when only controlling one wheel. This might be the case here too, but in either case, we'd need to fix these measurement issues if we are going to get the controller to work well.

7 References

- Robot information
- Controlling motor directions with PWM
- How to wire up motor driver
- TI LaunchPad MSP430 F5529 Development Kit Datasheet
- Datasheet for the photo interrupter