DC Motor Control

By

Tony Hoff

Course Instructor:

Dr. Guiherme DeSouza

ECE 4220 Real Time Embedded Computing

Department of Electrical and Computer Engineering

University of Missouri – Columbia

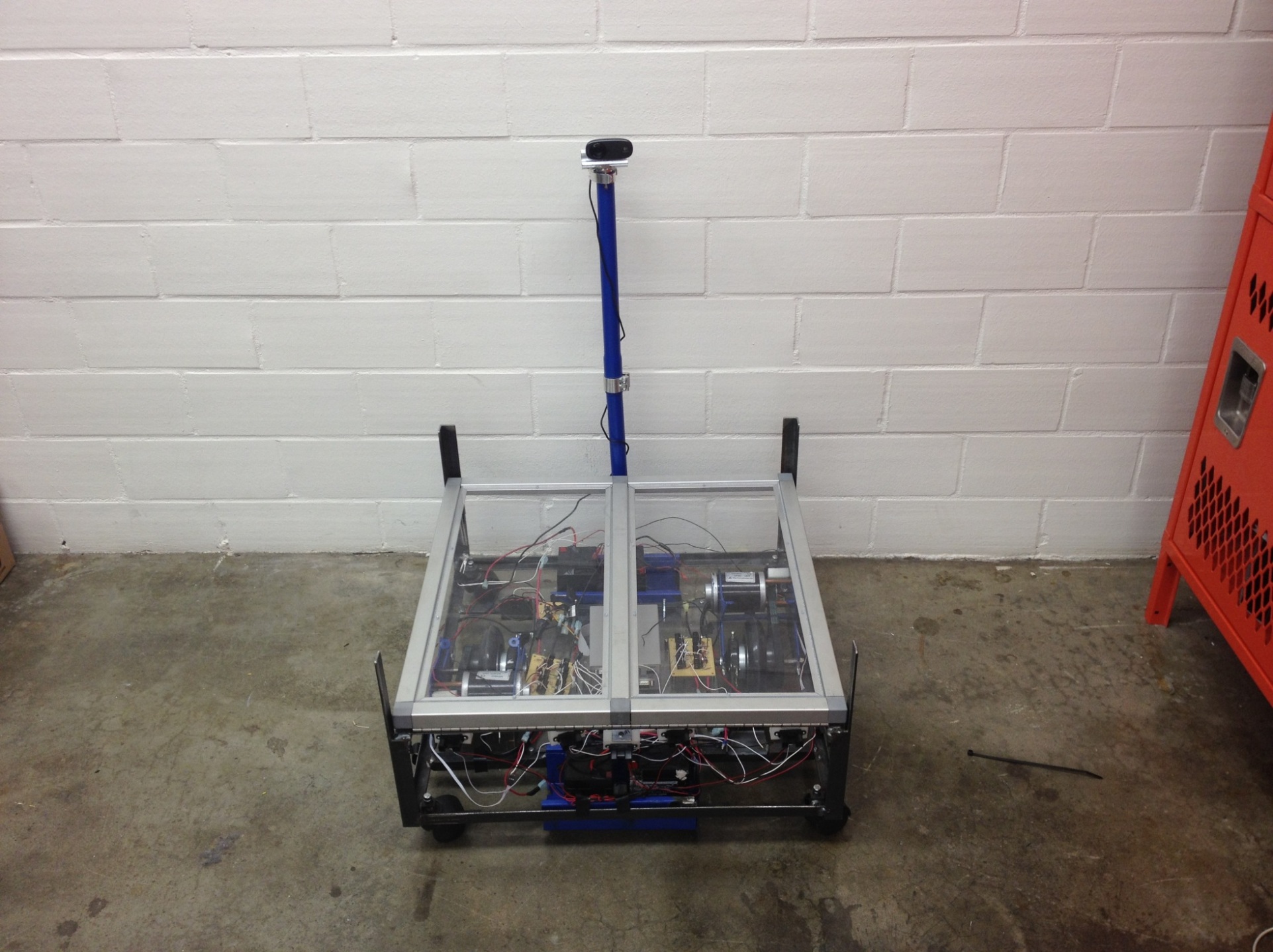
Columbia, Missouri

December 9, 2011

# Abstract

This project uses real time functions to implement pulse width modulation to control DC motors and read optical encoders to determine the speed of the motors. My capstone group had many complications implementing these functions with hardware and other software implementations. This project is intended to solve these problems. The results in this report show that the power sent to the motors can be varied with PWM and the speed of the motors can be read. Both of these processes can run in real time.

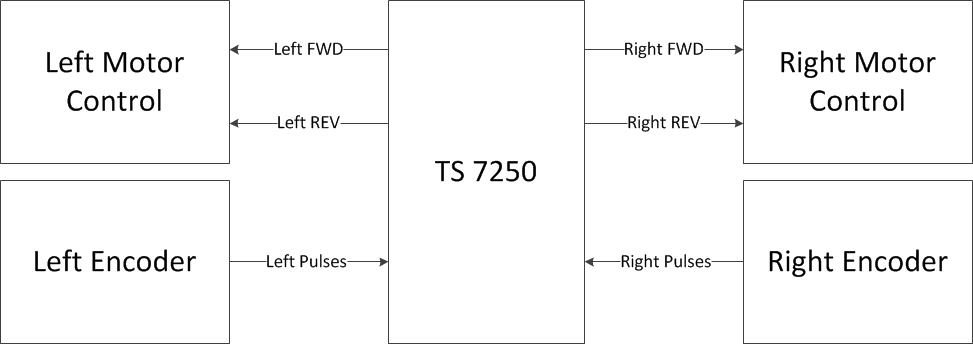
# Introduction

 This semester my capstone group project involved using an electric motor vehicle which is pictured on the right. Originally the motors were to be controlled using the pulse width modulation pins that were on an ATmega microcontroller. The optical encoders were also to be read from the ATmega. The TS 7250 would control the ATmega using serial communication. This implementation proved to be extremely complicated due to many complications. I was unable to contribute to this implementation because I had to work on my own assigned task for the capstone project. For this Real Time Embedded Computing project I implemented the motor control using real time functions on the TS 7250. Because it uses the ADC and the DIO on the TS 7250 it eliminates the ATmega and extra signal wiring required by the other implementation.

The DC motor control involves three threads that I refer to as pwm, adc, and main throughout this report. These threads run in parallel. The pwm thread constantly updates the duty cycle of power sent to the motors. The adc thread determines the speed of the motors. The main thread controls data amongst the threads, such as the set speed and quit command. Semaphores were used to allow the threads to use shared memory to interact with each other.

# Background

The figure below shows the hardware and signals that are used in this project. When the FWD pin signal is high the wheel spins forward, and vice versa for the REV pin. The encoders send out a pulses based on the speed the wheel is turning. I did not participate in the construction of this hardware.



The speed of the motors can be determined by reading the number of pulses sent from the optical encoders that are attached to the rotors of both the right and left motors. Part of the encoder is a circle with 8 notches on the outside. An optical eye on encoder sends a low signal when its path is blocked, but when one of the notches on the circle allow the signal to transmit, the encoder sends a high signal. The axis of the encoder is 5 times smaller than the axis of the wheel, so for every complete wheel rotation, the encoder switches from high to low 40 times. By counting the number of transitions from high to low over a period of time or measure time, you can determine the speed of each motor. The figure below shows how the speed of each motor was calculated. Because the measure time is 0.20 seconds the accuracy of the frequency can only be +/- 5.0 Hz.

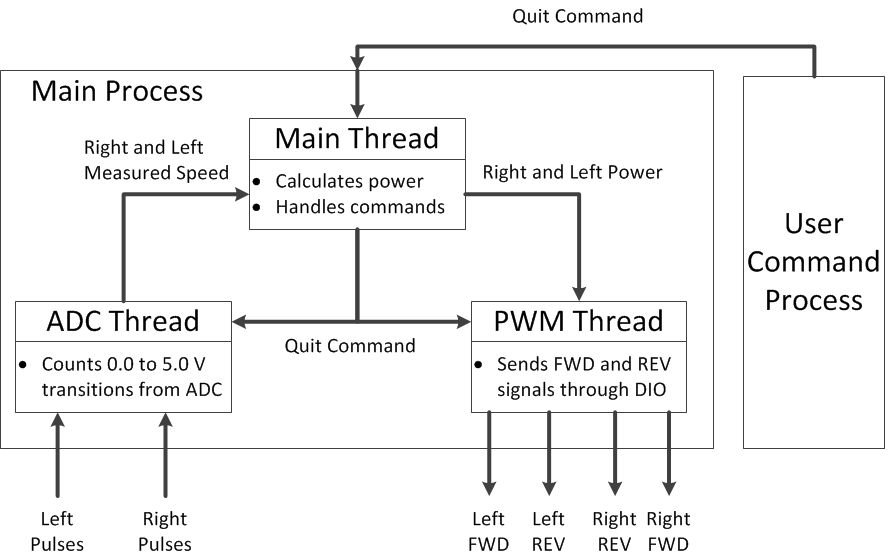


Pulse width modulation is when the average value of voltage fed to the load is controlled by turning the switch between supply and load on and off at a fast pace. The longer the switch is on compared to the off periods, the higher the power supplied to the load. Pulse width modulation is a great way to control DC motors.

There are many alternatives to my implementation. For example other hardware could be used to implement the PWM. This would conserve CPU processing time on the TS 7250 and simplify the software. However, this implementation would require the use of more DIO pins and additional hardware components. Also using hardware could be more difficult to troubleshoot or adjust than software. Furthermore, using the C function usleep instead of the real time functions could have been used to read the encoders. This would simplify the code, but a non-deterministic speed reading could be significantly less accurate. Using a single thread over multiple threads is another possible alternative implementation. Using a single thread simplifies the coding in the sense that it requires no inter-process communication. However, the code becomes less modular and therefore more difficult to change and trouble shoot. Also you would not be able to read the ADC, send power, and handle user commands in parallel.

# Proposed Implementation

The figure below shows my proposed implementation of the project. The three threads were implemented for this project. Each of these threads are described in detail in this section. The User Command Process was not implemented during this project. It communicates to the main thread via a named pipe to control when the program exits. That way the motors are not left running when the program is over.



The pwm thread was used to control the duty cycle sent to the motors. Each motor is controlled by an integer varying power level that ranges from -100 to 100. A positive power sets the DIO forward pin and a negative power sets the DIO reverse pin for each motor. The absolute value of the power level determines the duty cycle for each pin. This power level is updated once every 0.10 seconds so that gradual increases in speed can be achieved. Two semaphores were used to control the writes and reads between the pwm thread and the main thread. Also another semaphore was used to signal the thread to exit. This way the motors would always be turned off when the program was finished.

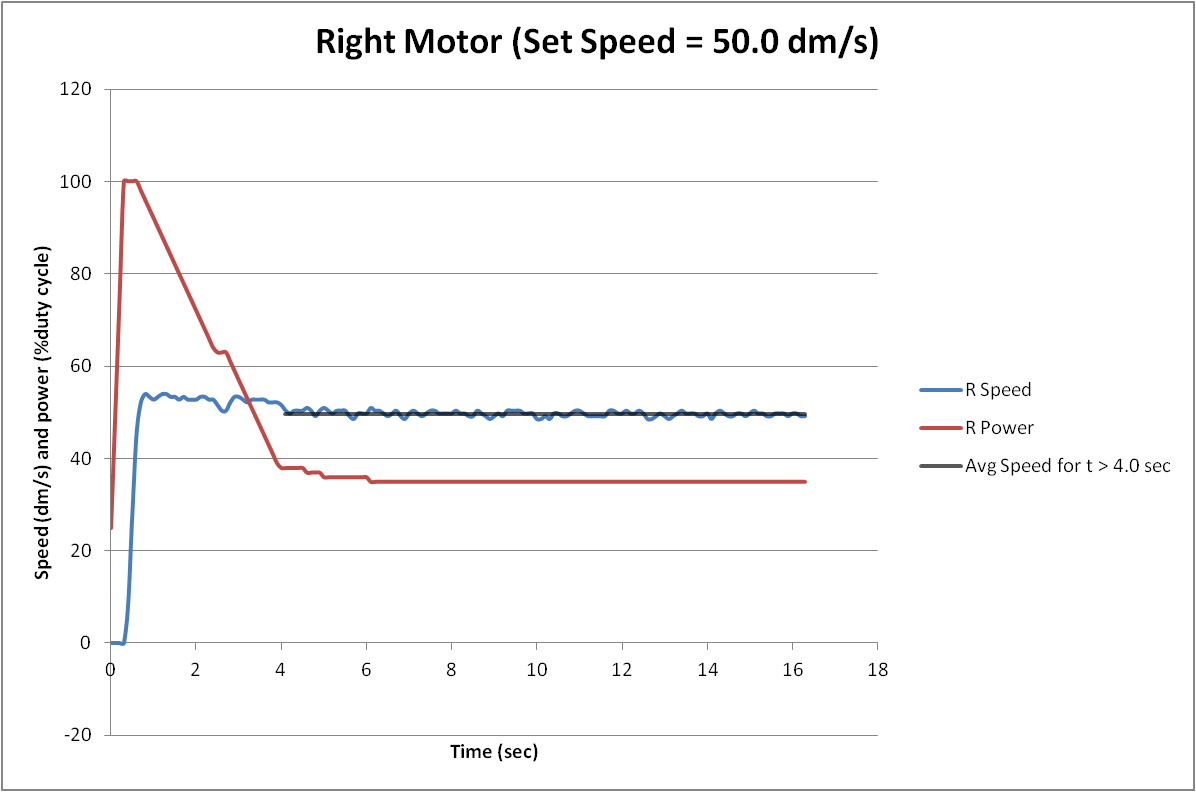
The adc thread reads two channels of the ADC to determine the signals from the encoders. Because the DIO inputs only allow a 0 to 3.3 V, and the optical encoders output a 0 to 5.0 V signal, the MAX 197 ADC on the TS 7250 was used to read the encoders. A better solution would involve using the DIO inputs and voltage dividers.

The maximum speed of the motors with no load is approximately 475 Hz so a 1.0 kHz sample rate for the adc was chosen. To count the transitions from the encoders, the count of transitions for the last 200 ms are stored in a buffer. So the accuracy of the encoder can only be +/- 5 Hz. The frequency can be determined by the number of counts in the buffer. Two semaphores were used to control the writes and reads of the right and left set speeds between the ADC thread and the main thread. Another semaphore was used to signal the thread to exit. Without the use of the semaphores the shared memory could be read while another process is writing. The semaphores eliminate this scenario.

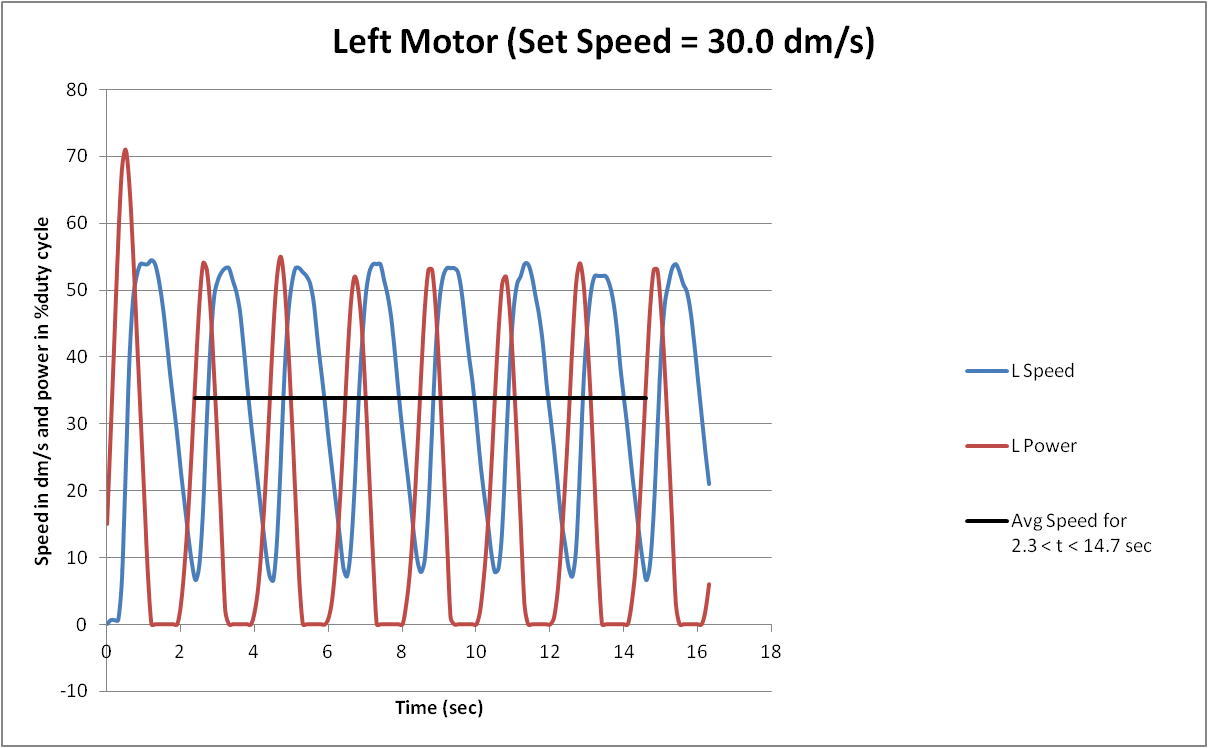
The main thread calculates the power sent to each motor based on the difference between the set speed and the measured speed. The greater the difference between the set speed, the greater the change in the duty cycle or power will be sent to the motors. This thread also reads named pipes for data sent from the user command process and any other processes. The semaphores it uses ensure proper use of shared memory.

# Results

Data was collected from tests of the implementation to determine how the software worked. In the test of the test results that follow, the right motor was set to run at 50.0 dm/s and the left motor was set to run at 30.0 dm/s. During the test the vehicle was set on blocks so that the wheels had no resistance and the vehicle could not move. The plots below show the results of this test.



The plot above shows that initially full power is applied to the right motor to bring it close to the set speed. This high duty cycle causes the wheel to spin too fast from about 0.75 seconds to about 4 seconds. During this time the power or duty cycle is decreased until the motor speed reaches an average speed of about 49.6 dm/s at 4 seconds to the end. The slight error is because the power is not adjusted if the speed is within 1.0 dm/s. Also because integer power levels are used, certain speeds that require precise floating point power levels cannot be achieved.



The plot above shows that initially full power is applied to the left motor to bring it close to the set speed. Because there is no load on the wheels changes in the power level vary greatly from 0 to 100. The power has to be set very low to achieve the 30.0 dm/s speed. In this test power level is set too low and it is set to high. This pattern continues throughout the test. The average speed of the motor oscillates around 33.8 dm/s. This inaccuracy is caused by the same issues that were stated about the right motor plot and because the power level never converged on a value that could set the speed.

# Conclusion

The project was a success in the sense that I was able to accomplish my goals of implementing PWM to control power to the motors and to read encoders to estimate the speed of each motor all in real time. However there are many possible improvements that could be made to this implementation. There was some error between the set speed and the actual speed. Ideally the set speed would always be exactly the measured speed and change to the set speed instantaneously. This is physically impossible, but any changes in the implementation that would decrease the time it takes to reach the set speed and increase how close the measured speed is to the set speed would improve the system. I believe that implementing a proportional–integral–derivative controller (PID controller) would generate a better calculation of the power sent to the motors. This would decrease the time to change to the set speed and the closeness of the set speed to the measured speed. Also by using power levels with floating point variables could allow the motors be to be set to a more accurate speed. The implementation of the speed measurement could also be improved. If interrupts were used then I believe the accuracy of the speed would be much more accurate. Also using the interrupts would allow a much a high frequency to be read because the only limitation would be if the processor could handle the amount of interrupts. Furthermore the encoders could also be switched to be used on the DIO instead of the ADC because this would make more sense.