Advanced Programming in Engineering

Christiaan Reurslag – s1495089

Description of the assignment

Instead of doing the two standard Arduino assignments for a total of 1 EC, I only did the second standard Arduino assignment for 0.5 EC. To acquire the other 0.5 EC, I finished two projects. The first project I did, I made an Arduino controlled robot which can self-drive without colliding. It uses an ultrasonic sensor to detect obstacles and both wheels are fitted with encoders to measure the rotation rate. The second project is about an Arduino controlled battery charger. The battery charger keeps track of the charge time, the voltage across the battery and the temperature of the battery. These parameters are real time plotted using MATLAB.

To program the Arduino for both projects, I didn't use the standard Arduino IDE. Instead, I made use of Microsoft visual studio 2017 together with the extension Visual Micro. This IDE is much more advanced. It has IntelliSense which makes programming easier and faster. It also has a debugger, so bugs in your code are easier to find.

Arduino controlled robot

Introduction

The Arduino controlled robot is shown in figure 1. The ultrasonic sensor used to detect obstacles is mounted on a servo motor so the robot can look in different directions. The robot uses two engines to propel itself. By rotating the engines in a different direction, the robot can steer. When both engines rotate with the same speed and direction, the robot will drive in a straight path. The used engines are quite cheap and the quality is not very good. Because of this, the engines encounter a different resistance and will rotate at a different speed even when they get the same voltage. To compensate for this, both engines are equipped with an encoder consisting of an encoder wheel, ir-diode and an ir-phototransistor. A close-up of this encoder is shown in figure 2. A H-bridge is used to power the engines. On top of the robot there is a button to start the program and a potentiometer to set the speed of the robot. The robot is powered by a battery pack usual used to charge cell phones.

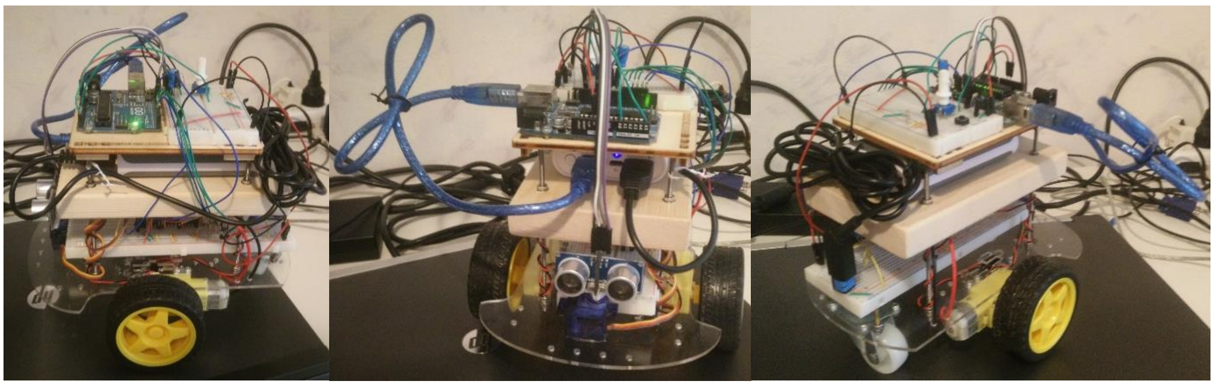

Figure 1: Arduino controlled robot

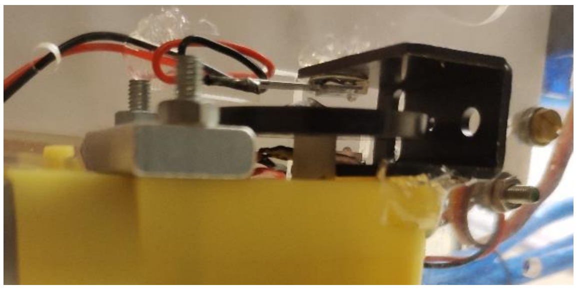
Figure 2: Close-up of the encoder

Schematics

The schematics of the Arduino controlled robot are shown in figure 3. As said earlier, the button is used to start the program and with the potentiometer it is possible to set the speed of the motor. The engines are connected to a H-bridge and the speed is controlled using PWM. A voltage divider consisting of a resistor and a phototransistor is used to turn the rotating speed in an analog signal. Using a NAND Schmitt-trigger (MOS 4093) this analog signal is converted into a digital signal. The rotation direction for every engine is set using two input connections on the H-bridge. These two connections should always have opposite polarity, so when one connection is low the other should be high and vice versa for the other rotating direction. This is accomplished using the same MOS 4093 IC because every Schmitt-trigger also has a NAND gate at the end and this is used to invert the signal. The button and potentiometer is mounted on the top prototyping board for easy access. All the other components are mounted on the larger prototyping board below the battery pack. A close-up of the larger prototyping board is shown in figure 4.

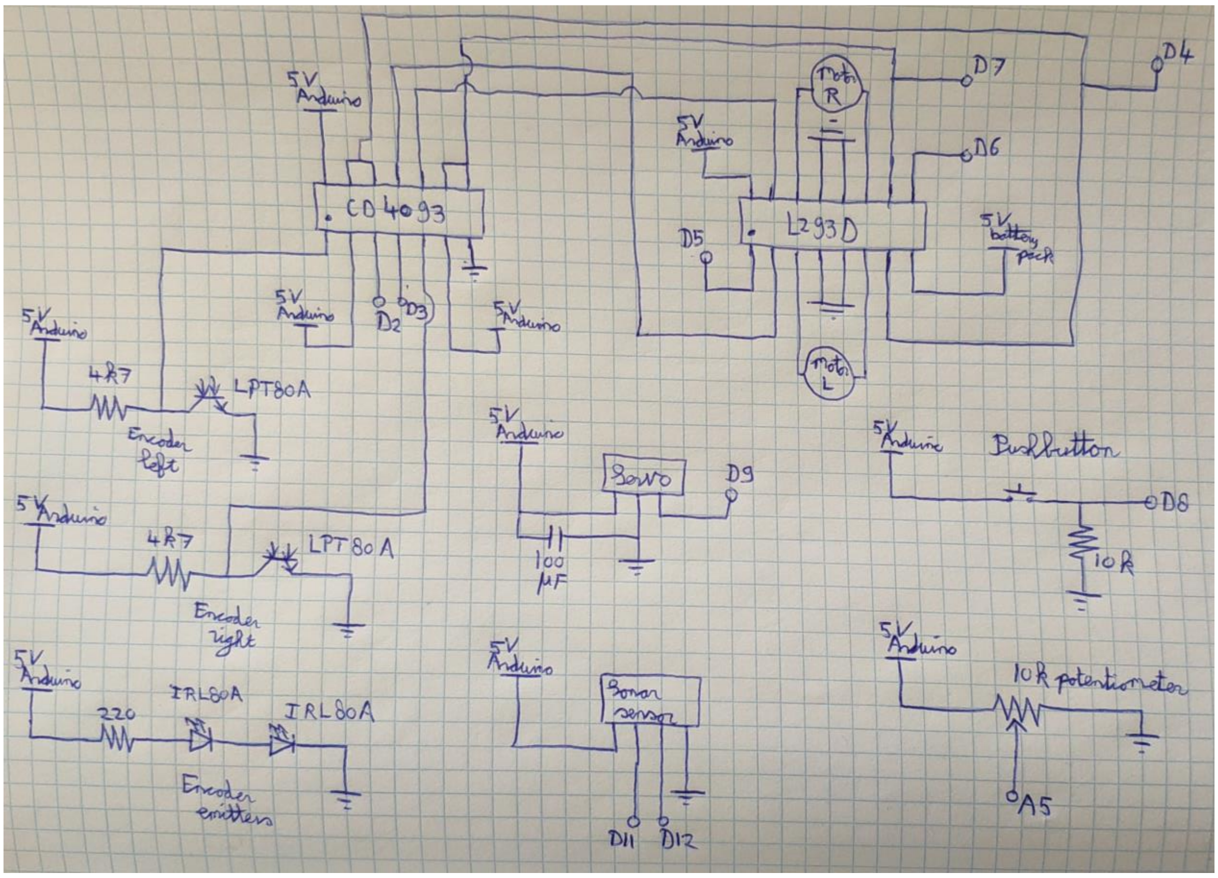

Figure 3: Schematics of robot

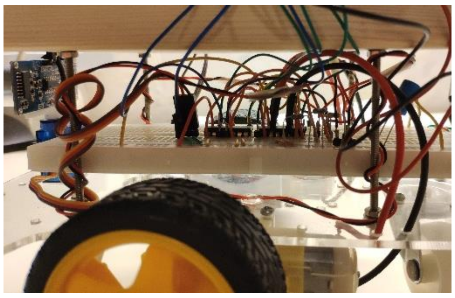

Figure 4: Close-up of prototyping board

Software

The program is object oriented written, I tried to create for every part of the robot a separate class. An Arduino program always starts with an .ino file (Arduino_Robot.ino). This file is used to create an instance of the robot class. This class will handle all the functionality of the robot (except for the two encoders) as long as the run method is called with a high frequency. The Arduino keeps track of both encoder values using interrupt service routines. It is quite difficult to create an interrupt service routine within a class. A class method always takes one invisible argument, an pointer to the object itself (this) and this is not allowed for an interrupt service routine. To work around this problem, the interrupt service routines are declared outside every class. I will continue this section by giving a short explanation of every class.

Robot

The robot class adds everything together. The run method should be called with a high frequency. This method calls other methods from other classes with the right frequency.

Timer

The timer class is used to make the timing easier. A timing object is set with a certain frequency and when the right amount of time has passed the fire method will return true.

CollisionAvoidance

The collisionAvoidance class will handle the avoidance of obstacles. The class controls the servo motor and the ultrasonic sensor so it can measure the distance from the robot to an obstacle in different directions. When the robot detects an obstacle in front of the robot, the robot will look to the left and right and steer in the direction with the most free space.

Button

The button class is used to read the input of a button. It detects changes in the state of a button and distinguish between a short press and a long press of the button. The class is immune for debounce effects because very short presses of the button are ignored. By creating a separate class for a button, it is very easy to add another button in the software. You can just declare another object of this class.

Oscilloscope

The oscilloscope class is not really necessary but is there for debugging. When the Arduino is connected to a computer it can be used to visualize certain variables in real-time. MATLAB is used to plot the real-time graph.

Encoder

The encoder class is there to keep track of the encoder values. Both engines have their own instance of this class. The updateEncoder method is called in an interrupt service routine. The conversion from encoder value to the actual speed of the engine is also done by this class. To get proper speed values, the method responsible for calculating the speed (calcSpeed) should be called with the right frequency.

Motor

Both engines are controlled by an instance of this motor class. The motor class can rotate an engines with a constant speed using a PID controller.

SpeedController

The speedController class implements the PID algorithm. It is used by the motor class to calculate the PWM value to let the engines rotate with a constant speed.

Propulsion

The propulsion class is responsible for the propulsion of the robot. It owns both motor class instances. The propulsion class is able to steer the robot or let the robot drive a certain distance at a constant speed. It uses a PID controller to reduce the error (encoder value of the left motor minus encoder value of the right motor) to zero. So in total, the engines are controlled by three PID controllers. Both engines have their own PID controller to keep the rotation speed constant and besides that, there is a PID controller that ensures both engines rotate the same amount of revolutions so the robot will drive in a straight line.

Conclusion

The robot can do the tasks it is designed for. The robot can drive with a constant speed, drive a certain distance, rotate a certain angle and is able to detect and avoid obstacles. During the development, the oscilloscope class was often used to improve the encoders and tune the PID controllers. It was quite difficult to let this work properly. To keep all the parts in place, I used screw threads and hot glue. This works very well, especially the hot glue was convenient to use. The electrical components were placed on a breadboard and so I had only little to solder. Right now, the functionality of the robot is quite limited but can be easily extended by adding more sensors.

Arduino controlled battery charger

The Arduino controlled battery charger is shown in figure 5. MATLAB running on a laptop is used to plot in real time the charge current, the voltage across the battery, the PWM signal used to control the current source, the PWM signal after it is filtered with a low pass filter, the voltage at the gate of the MOSFET and the temperature of the battery. The Arduino is powered from the laptop. The charge current comes from a 24V voltage source. During charging, a temperature sensor is put on the battery using tape. The battery charger charges a battery with a constant current. This is ideal for nickel based batteries. The charge current can be adjusted. Most of the time a charge current of 10% of the capacity of the battery is used. This is a save charge speed and a full charge cycle takes around 16 hours at this speed. The charger can also be programmed for a different charge cycle, like to charge the battery with a constant voltage or a combination of a constant voltage and constant current. During the charge cycle, the temperature of the battery is monitored and when the battery becomes too hot, the charge process will be abort. Three leds (red, yellow and green) are used to show what the charger is doing. The charge process is started with the press of a pushbutton.

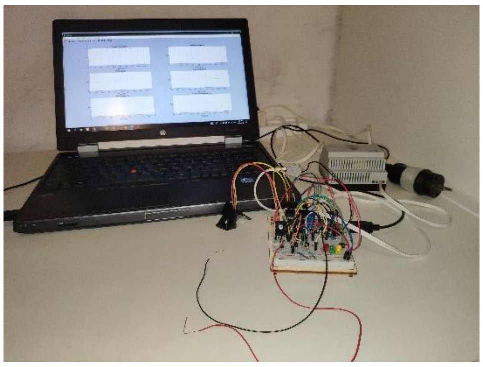

Figure 5: Arduino controlled battery charger

Schematics

The schematics are shown in figure 6. A MOSFET is used as a voltage controlled current source. The current that flows into the drain of the MOSFET is controlled by the voltage on the gate. The voltage that controls the MOSFET comes from a PWM output of the Arduino. This (digital) PWM signal is first transformed to an analog signal using a low pass filter. The low pass filter causes a small voltage drop and because of this, the voltage after the low pass filter is too small to control the MOSFET. The voltage is amplified using two transistors and this amplified voltage is connected to the gate of the MOSFET. I think it is better and easier to use an opAmp to amplify the voltage instead of the two transistors right now. However, I didn't have access to an opAmp so I kept using the transistors which works fine at the end. The Arduino can only measure an analog voltage in the range of 0-5V and sometimes a voltage divider is used to lower the voltage to this range so the Arduino can measure the voltage. The voltage across the resistor of 10Ω connected to the source of the MOSFET is measured and using this, the current through the battery can be calculated. To prevent the MOSFET from overheating, I attached a heatsink to it. A diode is used to prevent the battery from draining when the charger is off. I also made another circuit where a regular transistor is used to regulate the current through the battery. This circuit is much simpler because it is not necessarily to amplify the voltage coming from the low pass filter. This circuit is shown in figure 7 and works very well when a voltage supply of 12V is used. However, the transistor I used is only capable of handling 100 mA and because of this, the charge current is also limited to 100 mA. On the other hand, the circuit with the MOSFET can handle currents up to a few ampere and so this circuit is used. The red, yellow and green led are connected to respectively D4, D5 and D6 (in series with a 220Ω resistor to limit the current). The pushbutton is connected to D2.

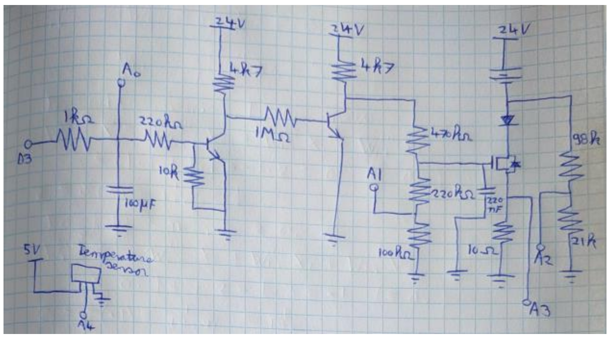

Fig. 6: Schematics of charger (using MOSFET)

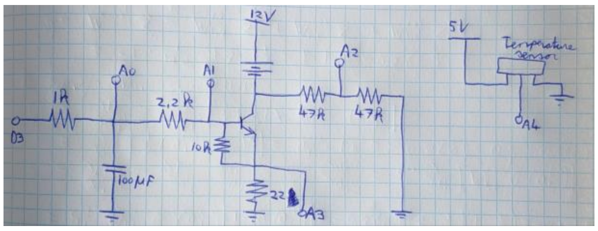

Fig. 7: Schematics of charger (using transistor)

Software

This program is also object oriented written. As always, the program starts with the .ino (Arduino_BatteryCharger.ino) file. This file is used to create an instance of the batteryCharger class and calls the run method of this class with a high frequency.

<u>BatteryCharger</u>

This class combines all the other classes together. The run method of this class is the main part of the whole program and should be called with a high frequency. It uses the timer class to call methods of other classes with the right frequency.

<u>Button, Timer,Pid</u>

This program also makes use of the button class. This is an advantage of object oriented programming because it becomes easier to reuse code. The same holds for the timer and Pid class. The Pid class has the same functionality as the speedController class from the previous project. This time the charge current is regulated by a PID controller.

<u>Oscilloscope</u>

The oscilloscope class is used to plot variables in real time using MATLAB. This time however, the class is not only used for debugging but it also visualize the charge process. The data is saved on the computer so it can be used later. For example, the data can be compared to a previous charge cycle of the same battery and the deterioration of the battery can be investigated.

Conclusion

I used this charger to charge 3 different nickel based batteries. The result of one such charge process is shown in figure 8. The battery being charged was a very old 1.2Ah NiCd battery from a drilling machine. The charge current was set to 125 mA and the charge process took 16 hours. As you can see in the graph, the PID controller works because the current was always between 120 and 130 mA. Somewhere in the middle of the charge cycle I touched a wire and that caused the deviation shown in the figure. The temperature of the battery increased steadily during the charge process because of heat development inside the battery. In the top right corner it can be seen that first the voltage across the battery increased and at the end the voltage started to decrease. This is typical for nickel based batteries. The other nickel based batteries I charged have a smaller capacity and so the charge current (around 10% of the capacity) was below the 100mA. For these charge cycles I used the circuit where a regular transistor was used instead of a MOSFET. During these charge cycles the charge current was better controlled and the fluctuations were smaller because the charge current was less dependent on the PWM value of the Arduino.

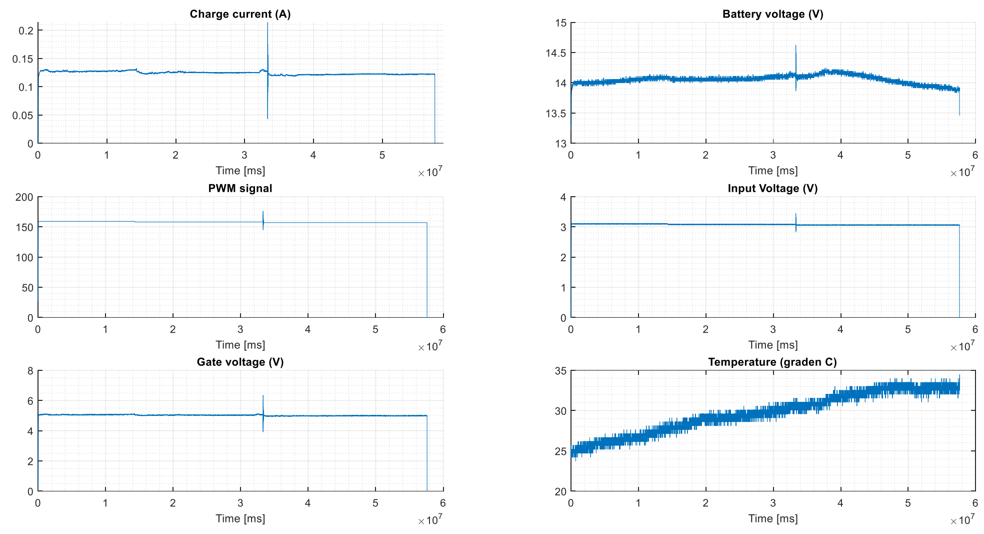

Fig. 8: Charge cycle of a 1.2Ah NiCd battery