CSCE 236

Embedded Systems

Robot Design Project 1 Report

“Theseus”

Michael Martin

Department of Computer Science and Engineering

University of Nebraska – Lincoln

Contents

Objectives or Purpose

This is a project for embedded systems using an Arduino board and an Elegoo Basic Robot kit. The objective of this project it to familiarize myself with the physical assembly and software that is required to work with an embedded system. The ultimate purpose of the robot assembly and code is to get the robot to be able to follow a maze containing walls and objects that it needs to move around.

Preliminary design:

Assembly

I first built the robot following the instruction video that was included with the equipment. The idea on how to tackle the project was to break things down to each component starting with the ultrasonic sensor and servo.

Coding for Servo Functionality

I originally chose to put the servo and ultrasonic sensor on two different timers. Using timer 1 for the ultrasonic and timer 2 for the servo. This would allow me to have different pre-scalers on each so that I could do a single setup. I created functions that would set the appropriate direction of the servo with each separate void function. Sensing was done in two ways, I built a standalone detect function that would do the pulse signals for the sensor to send its pulses and other nested functions that could store the returned data into 3 different locations depending on where the servo was pointed.

Coding for Motor Functionality

For this part I decided to go with the approach of using digital signals for the right and left positive and negative terminals and use the pwm signal for the enable pins. This will let me set which direction I want

then send the speed using the timer compare functionality. What I realized is that it would be easiest if I had both my enable pins controlled by the same timer so that I could send a reliably similar signal to both enables. From this I had to rewire/ unplug my servo as it was attached to timer2's pwm pin. For the time being I will leave this unattached and come back to it when I need the servo.

Coding for IR functionality

Since the IR attachment only requires a single input signal, that is not needed to be a pwm, I used the timer1 interrupt pin PD3 (INT1). This allows me to setup interrupts using the remote signal.

I did reuse the code from the IR decoding lab and then analyzing the array of timer counts to set a bit to 0 or 1 and then shift that into a register that would hold the 32 bit encoding of the remote signal. I set up a switch to handle the decoding which signal corresponded to which button on the remote. Coding two functions for the library; one that would output to the serial port which button I was perceiving and the other to control the motors.

Coding for Wall Following and Obstacle Avoidance

I was able to reuse my servo and motor library as I had created very modular code for the motors allowing for individual control of each wheel as well as forward speed and turn by angle desired. I did not change any pin configurations for the motors.

The servo had to be reattached at this point so that I could use it, this required redesign of my code and setting a new pin configuration. I ended up setting each function that would move the servo to reconfigure the pwm and pre-scaler to the needed value before doing each rotation. I then had to also add setup in for each time I started doing sensing. As I already had these setups functionalized this require little reconfiguration. Timer1 and Timer2 have different CS bit

configurations for different pre-scalers, noticing this required some debugging.

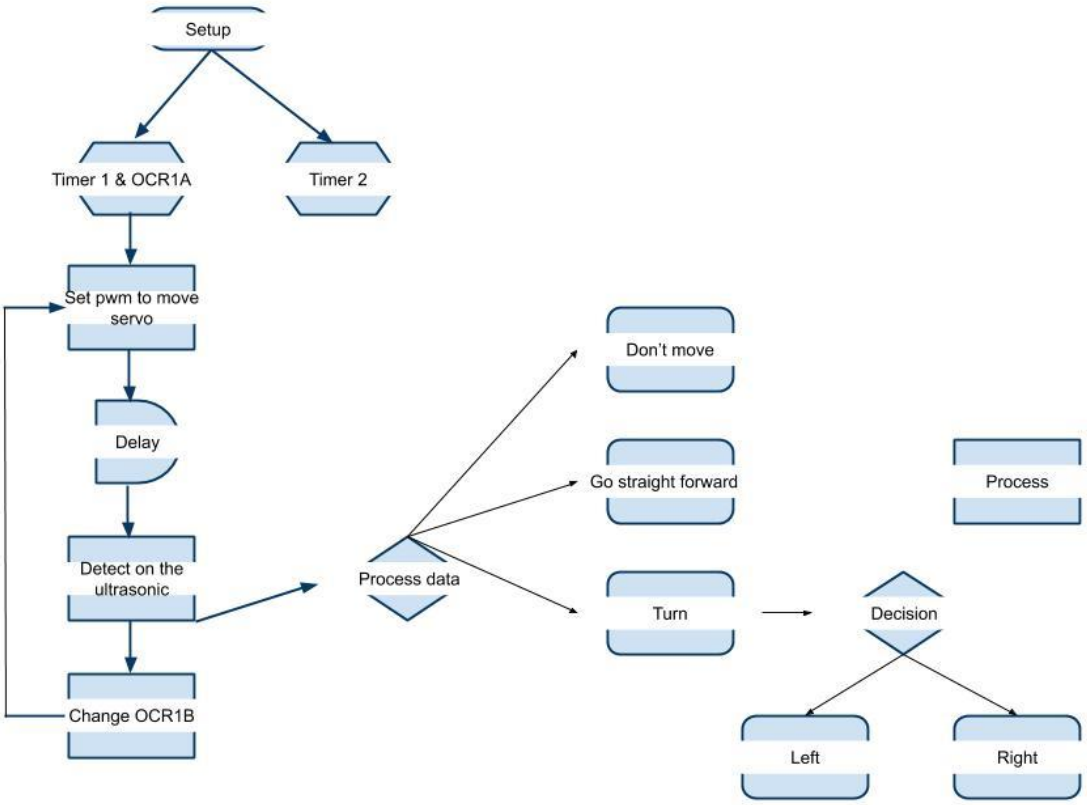
Final Coding Design:

I chose to use timer 1 for both functionalities of the servo and ultrasonic. This frees up timer 2 to be used for the motors. The idea that I had was to use the input capture pin to detect when the signal goes from high to low. Resetting the timer when the signal goes high and then using the ICRA register that captures the time when the signal goes low to count the time the ultrasonic sound took to go out and come back. Using calculations on the speed of sound, we can figure out how far an object is from the sensor.

The sensor is controlled via a pwm signal that is sent using timer 1 features to output a waveform and a specific voltage. The variation of the voltage will determine where the servo points. After a figured time, the servo can stop receiving a pwm signal since it will be in position. Also, once that position is reached a signal to output can be sent to the ultrasonic.

For the motors, I used timer 2 as timer 0 also messes with the delay function from the Arduino library. I plan on using the pwm signals from timer 2 for the enable pins on the H-bridge and just using high and low signals on the positive and negative terminals to choose direction of the motors. This allowed me to set a direction easily and then set the speed on a separate signal.

Software flow chart or algorithms



Hardware schematic

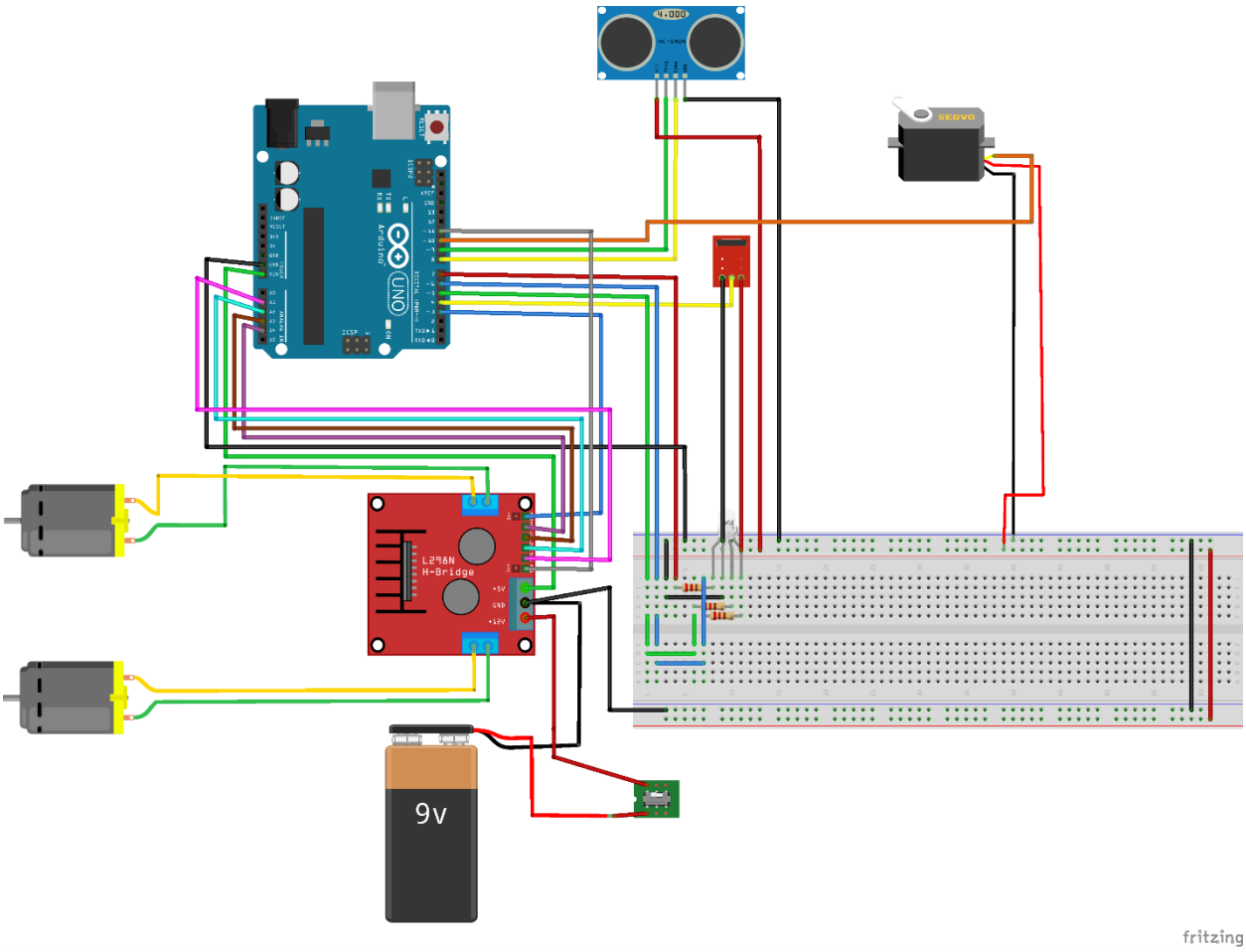


Figure shows the schematic attachment of devices that were used for the project. Using fritzing software to display where connections were made and what setups were used on the physical setups that will work with my code without adjustment.

Completed Build

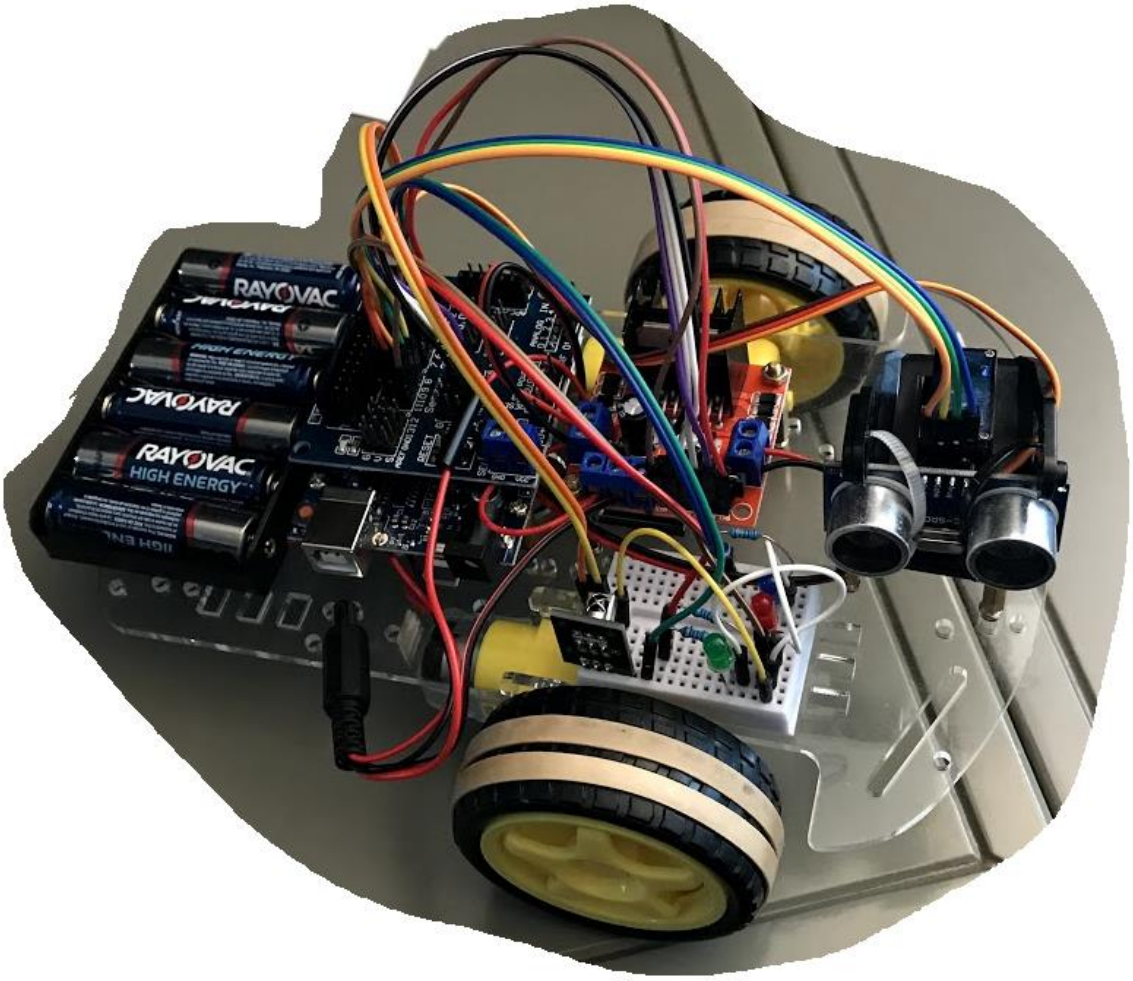


Figure is an image of the completed robot “Theseus”

Debugging

I had issues with each phase of the project, that I ultimately solved. First was figuring out how to send and receive valid data from the ultrasonic sensor. I kept getting complete garbage values back, so after some diving into the data sheets I found out that I needed to reset the timer when the signal coming from the echo pin went high and use the ICR1 to measure the time it took, since that is on the capture pin. After figuring that out I was able to adjust the pre-scaler to get an accurate number and then did a few measured tests to figure out how to scale that to inches.

Motors

When I got to trying to get my motors to run, I knew that everything was wired correctly as I had put the sample code that came with the robot kit onto it and both motors were able to run. But getting my code to have the same effect took some time. I found out that I was sending full signal to the enable pins because I was incorrectly assigning the OCCR2 registers. Once I figured that out, I was able to adjust the speed that I ran the motors to something that was desirable for operation.

Wall-Following

This was the most difficult part of the project as I was shooting for a good time on my first run. I had to make over 40 attempts of running the wall in my house in order to

IR Decoding

I had to switch remotes from the one I had planned to use since the enable signal was indistinguishable from each bit. I ultimately ended up using a LG TV remote for the IR motion. Had to test a few things when it came to bit shifting into getting the MSB into the correct spot so that I could use a switch in my code to decode what each remote signal should output.

Wall Following and Obstacle Avoidance

This one required about 20 different attempts of coding; I wrote out a state diagram that would represent the different states I could encounter going through the maze. This helped diagnose where I needed to take action and what action to take dependent on what inputs I was receiving from my ultrasonic sensor.

Testing Methodology or Results

Testing the ultrasonic once it was working required a measuring tape and using the Serial port to print the returned time from the ICR1 that I was using. After 5 set test measurements I used excel to plot the graph, noting that the pattern was linear I figured a scale to change the time on the timer to inches measurement.

The next thing that required the most amount of testing was the wall following operation. I tested over 40 different variations of code before I was able to find a speed to run the motors at that kept the robot heading in a straight forward direction. Then using the ultrasonic to gauge how close I was to a wall and speed up either motor to adjust the heading to stay within a foot of the wall. Since this operation relied upon the specific motors that I was using it was a lot of trial and error to find a set of values that would work. Battery charge level came into play because after several tests the settings would be off again and would require readjustment.

I tested the remote by outputting to the serial port from a switch statement that would tell me what code it was receiving and then output the button code. This worked out well as I had set a default to catch any erroneous codes and output accordingly.

One thing that I found very helpful along the way was using attached LED's to signal what state or part of my function I was entering when trying to develop an algorithm that would successfully navigate the

maze and wall-following, I setup 3 onboard LED's of different colors that I could turn on using created functions that I would set to turn on when executing specific code or entering a conditional. This helped in both of those functionality builds. Having 3 LED's let me signify 8 different states my code could be in. One specific use that I implemented this for was in the obstacle avoidance. When the robot would get too close to the wall, I would turn on the red LED to signify that the robot detected this and would turn away. When the robot would detect an object in front of itself then I would display all the LED's. If I had situations where the robot ran into the wall or obstacle, then I could diagnose quickly whether it had detected this and acted improperly or failed to enter the correct conditional.

Answers to Lab Questions

1. How fast does the signal pulse from the sensors travel? Speed of sound 343 m/s

2. If the distance away from the sensor is 1 in, how long does it take for the sensor pulse to return to the sensor? 1 cm? sensor pulse to travel 1 inch is 74us to send and receive would be 148us. 1 cm out and back would take 58 us.

3. What is the range and accuracy of the sensor? Accuracy is about 60 ft; the sound spreads out at a 15 degree angle from center so if there are obstacles within that it can severely reduce accuracy.

4. What is the minimum recommended delay cycle (in ms) for using the sensor? How does this compare to the "working frequency"? It is recommended that you wait 60 ms between pulses so that you do not get overlapping signal returns.

The LED does not always turn on and off as expected sometimes. Why?

This is because some other code might be running and the interrupt only goes if it is held for a certain number of clock cycles and the code it is currently running is completed.

Describe the advantages and disadvantages of both solutions used to correct the above problem. Make sure to save the solutions as two different sketches so you can post both approaches in the next two essay questions.

The first does not really solve the problem, it disables other interrupts while this code is being ran and then re-enables them afterwards.

Using the toggle on any logic change is good, except you get an extra call to the ISR when you let go of the button also. That means on a button press it is executed and on the button being let go it is being executed as well.

What happens to the blinking rate when you press the button? Is it consistent? (You can get checked off for this question with the next checkoff.)

The blinking slows way down. This is because there is a serial print function that takes a long time. It is consistent if held.

What value did you use for OCR1A and how did you figure this out? What happens now when you press the button? Does the blinking rate change?

7812, I figured this out because the pre-scaler was at 1024 and this is the number using maths to get 500 ms.

The blinking rate does not change now, even when the button is pressed and there is printing to the serial port.

Observations and Conclusions

Observations that I have made thus far are that careful consideration must be made for each timer that is used and what pins those timers operate on. Messing with timer 0 can cause delay function to not work as desired. Taking the time to go through and search the data sheet for the Atmega 328p is well spent. I have found many useful gems of information in there that has diagnosed issues that I was having.

Interrupts are extremely useful for timing processes that need to run cyclically without delays because of long code. Although care must be made so that the interrupt itself is not too long as to mess up current operations.

Translating simple code or algorithms to real world situations is not as easy as I would have surmised at the beginning of the semester. I had assumed that having a robot follow a wall was a simple task that would not require much debugging. I was wrong. The number of factors that

play into using a single sensor to calibrate the movement of an object takes much more care and precision than first guessed. Having the full understanding that the robot is dumb, and you must tell it exactly how to act under every bit of sensory data was a realization that I came to. The robot had no sense of its own direction, only what its sole sensor had received. Making small adjustments early helped stop a cascading problem of losing the wall entirely and just navigating into the center of the room or crashing directly into the wall.

IR Decoding Chart

Values in hex used to accept remote input from an LG tv remote

Hex values	corresponding button
20df02fd	–UP
20DF827D	–DOWN
20DF609F	–RIGHT
20DFE01F	–LEFT
20DF22DD	–OK
20DF8877	–NUM1
20DF48B7	–NUM2
20DFC837	–NUM3
20DF00FF	–CH_UP
20DF807F	–CH_DOWN

Documentation

Fritzing Software used for the schematic

Google Draw used for the algorithm flowchart