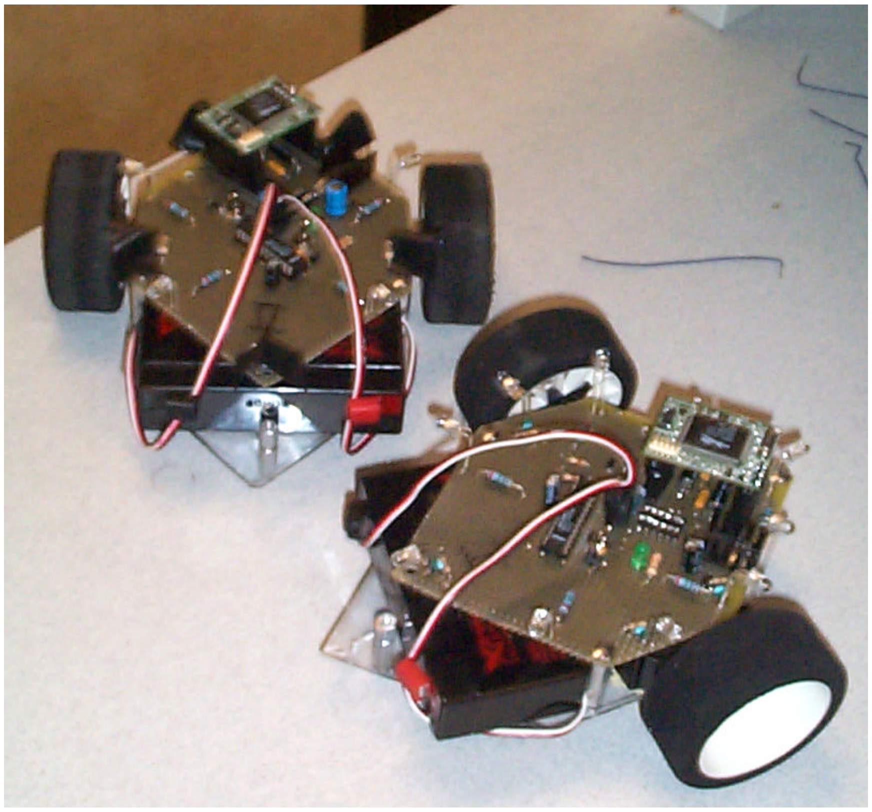
IR-Tropic Mice



Darrell Ross
Jason Holbrook
Advanced Microprocessor Design

Table of Contents

1. Introduction

Computer Engineering 123: Advanced Microprocessor Design has one assignment: to build anything we choose as long as it incorporates a microprocessor. The first and most important part of this class is choosing a project. Darrell's design proposal was to build a micro mouse that shot baskets. Jason's design proposal was to build a micro mouse that was phototropic (followed light). Cyrus suggested working in groups, so Darrell teamed up with Jason.

The basic design of our project was to build two phototropic mice. We envisioned the following possible extensions to the project, if time allowed: enable the mice to find each other; enable the mice to play a game of tag; and enable the mice to detect obstacles. We built two phototropic mice. We enabled the mice to find each other. Tag was not far off, but we ran out of time. If we knew at the beginning of the project what we know now about designing and building a micro mouse, we could have progressed further. Even without any infrared knowledge, the basic experience of putting everything together was invaluable.

2. Design and Construction

The design and construction of our project was slow at first. We invested a lot of time in designs we never used. In the end, while we had spent 10 hours per week on our mice in the first six weeks, we spent 40 hours per week on the mice in the last four weeks. At the start we realized we were both over scheduled this quarter and our budget was small. We also knew we wanted small, aesthetically pleasing mice.

2.1 Processor

Cyrus recommended the Rabbit microprocessors from Rabbit Semiconductor because they were affordable, they had the I2C bus (a simple-to-use bus protocol) built in, and UCSC had a site license for the development software, making it unnecessary to purchase software. Low on funds, we chose the most affordable processor: the RCM2300 Core Module (Appendix A.1). The RCM2300 had 29 I/O ports and plenty of RAM and ROM for our ideas. We reasoned that we would need a maximum of 14 of the I/O ports: five detectors, five emitters, and two wires for each motor. Unfortunately, there were some issues we overlooked. Our biggest problem with the Rabbit chips is their 2mm pin spacing, while most companies use 0.1" pin spacing. To solve this problem we chose to elevate the Rabbit chip on a riser and run loose wires directly from the chip to headers or other necessary places. We also investigated building our own 2mm:0.1" adapters (see Appendix D). Having adapters instead of hard wiring enabled us to remove the chip any time we wanted to do something with the board that might have endangered the Rabbit.

2.2 Motors

Keeping to our low budget, we bought two 24-volt DC motors and two 5-volt DC motors for prototyping. Each set cost $1.60, right in our price range. Over the first six weeks, we went through three different motor designs.

The first design used two I/O ports per motor. We sent either a positive or negative signal through each wire depending on which direction we wanted the motors to go (Appendix B.6).

Unfortunately, the RCM2300 Core Module did not have a D/A converter. We had to add our own D/A converters, but this caused further complications: the 5-volt motors were too fast with too little power; and using D/A converters would require more than two I/O ports for each motor if we wanted more than four possible speeds.

Our second design used D/A converters to convert digital signals from the Rabbit to analog voltages for the motors (Appendix B.7). We decided on 4-bit D/A converters since they were abundant and would occupy eight I/O ports on the Rabbit. Unfortunately, we made an erroneous assumption about the D/A converters. We assumed it was possible to find a D/A that could output between –4.5-volts and +4.5-volts with an input of +9-volts. If a D/A does exist that can do this, we didn't find it in time. We did not notice this error until three weeks before the end of the quarter (see Appendix C).

We ran out of time for our patient, budget conscious search for motor design solutions. We discarded both the budget constraints and our second motor design. We quickly surveyed the motors other design groups were using. We selected the Futaba S3003 pulse driven servomotors. The S3003's were made for steering on RC cars, so we modified the motors for our purposes (Appendix A).

The Futaba had three inputs: ground, power, and signal. The symmetry of the pulse signal controlled the forward, reverse, and stop of the motor. This made the motors easier to control because we no longer had to change the voltage; we only had to change the signal from the Rabbit output (For further discussion on driving the motors, see Section 3).

The next unanticipated turn of events was burning out three of our four motors. We thought they were 12-volt motors, but they were 6-volt motors. Next, we found that the motors drew so much current that starting them up reset the processor, creating an infinite reboot loop. We had to make separate circuits for the motor and the chip. The separate circuit used a second voltage regulator that regulated at 5-volts using an input of 7-volts to 12-volts. Unfortunately, our input ended up a 6-volt lead acid battery because the motors ate up a 9-volt battery in less than two minutes. We removed the regulator and everything worked great!

2.3 Sensors

Our original design called for phototransistors to enable the mice to head for a certain color of light; in theory good phototransistors can tell color differences. However, we quickly scratched that idea and decided to use infrared instead. Due to time constraints, we only lightly researched different IR detectors and emitters. Basically, we went to Santa Cruz Electronics and purchased the IR emitters and detectors they had in stock. At first, four of each seemed right but that required a 90-degree spread for each one, and they would have to work perfectly. To solve this we decided to use five of each. Conveniently, this placed an emitter pointing straight backwards and a detector pointing straight forwards.

2.3.1 Infrared Detectors

Originally we were going to use 4-bit A/D converters. That would have meant five converters and 20 I/O ports, leaving us with only nine ports left to run the motors. If we used eight ports to run the motors it would be perfect, except we wouldn't be able to add obstacle detection or any other features. Also, five chips would take quite a bit of board space, potentially violating our small design goal. While searching for A/D converters, Darrell stumbled across one capable of four different analog inputs and a single wire bus output. Jason found a MAXIM chip that did the same thing and conveniently used the I2C bus. We were stoked! We later found a single chip capable of eight analog inputs. Using the I2C bus, we would only use two I/O ports for all five sensors and could add up to three more without increasing chip or I/O port count. We finally settled on the MAXIM 128 (see Appendix A) because we could set our own voltage references (values used for maximum and minimum voltages) making opAmps to magnify sensor readings unnecessary.

2.3.2 Infrared Emitters

The design for the emitters was simple. We knew we would need a resistor in front of each emitter, but we did not know the emitters had dynamic resistance. When Darrell calculated the needed resistance and built the circuit, the emitter received only 1-volt while it needed a minimum of 1.5-volts. Jonathan Casper helped Darrell find the current through the emitter and then found that a 70 ohm resistor would work perfectly. Unfortunately, 70 ohms was still too much. By testing different values it was discovered that 10 ohms was the correct resistance. This meant that the 10 ohm resistor, handling 3.5-volts, was getting much too hot. Jason bought some 10 ohm resistors that could handle up to 1/2 watt and were flame proof.

Our next issue was the current the emitters drew (see Appendix A). We thought it was 50mA for each emitter, a total of 250mA for all five. In fact it was 250mA for each emitter, a total of 1250mA for all five. Hooked up to the same circuit as the motors that circuit drew about 1500mA, and our battery was rated at 1300mA. This meant a little less life out of the battery before recharging.

2.4 Complete Hardware

The complete mouse is nice. It has a low profile and compact layout. One part overlooked was the limited overhead space. With motors and batteries mounted under the proto board, there was almost no room for high profile wire wrapping pins (see Appendix D). A secondary result was that the wiring job was impossible to do neatly as everything always crossed in the center. The wiring looks a lot like a rats nest on each mouse, but they work.

3. Software

The software has three main parts. Driving the motors with a pulse, reading data off the I2C bus, and using that data to change the speed of the wheels for turning.

3.1 Motor Driver

The motor was the first thing that was attached to the mouse, so this was the first thing that was programmed. Initially the idea was to use the Delay MS command that comes with co-states to make our timing work out properly. The downfall of this was that we could only use whole numbers in the delay time. This did not work. Instead we used a for loop for timing. Turned the bit high for an amount of time and then shut it off for an amount of time. This worked really well and we were able to make the mouse go forward, backward, and almost stop. The for loop was still not precise enough to make the motors come to a full and complete stop (before looking both directions and proceeding with caution). The code for the initial motor driver, used to see if we could produce a pulse with a certain frequency, can be found in motors.c.

After the motors were both working using the same for loop, we decided to use co-states. This worked well and we were able to drive each motor independently of the other. The code for the co-state motor drive can be found in motorscostate.c.

3.2 Sensors and the Maxim 128

The maxim 128 chip uses the i2c bus. This required a great deal more research than had we ordered an A/D converter that used the standard 8 bit bus. The i2c bus required that we use the i2c library. After looking at some example code and reading up on the data sheet for the maxim 128 we were able to understand how the i2c bus and the protocol worked. The process is a lot simpler than it was first anticipated. The master (Our Rabbit chip) would send a start bit, then a 7-bit address followed by a write signal. The device containing that address (Our Maxim Chip) would acknowledge and then the master would send a control byte. Again another acknowledge bit, then a stop bit. This told the max128 to start reading data off of one of the A/D converters. The rabbit would then send a start bit, the slave address and a read bit. An acknowledge would be sent by the maxim, then 8 bits of data (the MSB of the 12 bit digitized signal. The master would send an acknowledge that it received the data, then the next 8 bits sent by the max128 would contain 4 bits of the LSB of the address followed by 4 0's. A No Acknowledge was sent from the master and then a stop bit, and this signified the end of the i2c cycle. This program took a while to get working, due to understanding of the i2c bus and how it was supposed to receive data. Eventually this worked and we were able to read in data.

Because the data was in the form of 8 bits, we read it into a char value. From the data acquired we combined the values of the first 8 bits and the second 8 bits to form an integer value. This allowed us to compare the sensors, as well as print the values of the sensors to the screen. The code for this whole process can be seen in sensortest.c

3.3 The Final Program

The initial plan was to use costates. One costate would read the sensor values, one would drive one motor and one would drive the other motor. This never ended up working out because of the way that costates are designed. Ideally we would be running each process in a different thread and they would run practically simultaneously. As the values of the sensors would change, we would be able to change the values of the pulse and delay used to drive each motor. After finding out that the costate runs completely, and then switches control to the next costate we realized that this would not work.

We decided that using a gigantic while loop would work best. First we would read the sensors, send a pulse to the motors and then do it again. This produced strange frequencies, because of A the way we were comparing the sensors, and B, the delay times and pulse times were not equal. Initially the front sensor was compared to the rest of the sensors first, assuming that this would be the sensor that would most likely be the highest value. Then the next two side sensors were compared with the rest of the sensors and finally the rear sensors were compared. Compsensor (Our function for comparing the sensors) would return the value of the greatest sensor, however because of the way it was comparing them, it would return as soon as it found the greatest sensor.

This produced unequal frequencies for the pulses because of the time it took to read in a sensor. The difference would range from 48 Hz to 38 Hz, far too large for our purposes. This was changed for the final code so that there would be a competition between the sensors. The first and second sensors were compared and the winner would go on to the next round to be compared with the next sensor. This only required 4 comparisons and had a constant run time. From this we were able to produce a constant frequency.

Next we determined the delay and pulse times for each sensor to be sent to the motors. We used a case statement to adjust the values of delay and pulse depending on which sensor was returned from comp sensor into our variable bigsensor. To continue to have a constant frequence we determined that the delay and pulse would have to both add up to 300. Pulse would enter a for loop for however many times it was set to, during this time the bit that drove the motor would be turned to high. The bit was then turned to low and delay started to run. This allowed us to maintain a constant frequency for the motors.

The next step was to determine the delay and pulse times for 5 states of the motor. Unfortunately because servomotors are never exactly equally we were forced to use different values for each motor. This required testing. We determined forward fast, slow stop, and reverse fast and slow for each motor. From those values depending on the case statement we would control the motors. To view the code that the final project was built upon look at mouse1.c and mouse2.c. Because the delay and pulse times were different for each motor, we were required to use two separate programs for each mouse.

4. Complete Mice

The mice work almost as we had hoped. Actually, they work exactly as we had hoped. One motor on one of the mice is a bit slow (we think it might be on it's last leg). So, that mouse is called the Coyote and the other mouse, which moves much quicker, is called the Road Runner. Our other idea was to have one be a Rabbit and the other Elmer Fudge so that we could be "hunting wabbits." The batteries don't last more than ten minutes, but this is acceptable for our first working prototype. The I2C bus reads perfectly with 16-bit numbers and the mouse heads for the strongest IR source, or runs from it depending on the mode of the mouse.

4.1 Hindsight

What would we have done differently? We would have won the lottery so we could afford whatever we wanted. We could have purchased state-of-the-art IR detectors, emitters, and servomotors. We also would have had sonar for object avoidance and some sort of tag mechanism ideally using radio signals. We would have bought lithium batteries for more power and longer life.

4.2 The Experience

Darrell Ross: I have been waiting for a long time for a class that would use what I have learned from previous courses. This class was the one. It used my microprocessor knowledge and abilities to read spec sheets from CMPE 121, my programming knowledge from CMPS 12A, 12B, 101, my knowledge from CMPE 12C on binary, hex, etc, and brought it all together. I also got to design and complete a little robot. I am looking forward to building more robots on my own as funds allow. I have already begun designs for a radio-controlled submarine.

Jason Holbrook: I have really enjoyed this course. It revealed to me that we have actually learned quite a bit here at UCSC. I feel prepared to do more complex design projects. The robots were loads of fun to make. Ideally had we more time.

Supplemental Materials

Appendix A

Appendix B

Intended Timeline:

Week 1: Think of design project and write proposal.
Week 2: Order/get initial parts (movement & base start looking for sensors)
Week 3: Start building, get stepper motors & drive control working
Week 4: Complete turning, forwards & reverse motion
Week 5: Start to get the light sensors to take readings
Week 6: Start programming to get the mouse to turn towards highest light sensor reading & maintain course
Week 7: Most likely still working on the light sensors & programming
Week 8: Start with the bumper & get the mouse to turn on light & run when it hits another bumper until a bumper is hit again, then resets into finding the brightest light (supposedly the other mouse)
Week 9: Create a second mouse (exactly like the first)
Week 10: Still working on second mouse/Start Report

Actual Timeline:

Week 1 through Week 5:	Research & waiting for parts to arrive. This was a large setback from our intended research and parts time.
Week 6 through Week 7:	Built movement of first mouse. Made the motors move back and forth. The largest obstacle was getting the wiring the way we wanted it to, and not having the motors create a re-set on the rabbit.
Week 8 through Week 9:	Built the sensors, and attached the max 128. As we were building we were also writing the code, so the code was finished about the same time that the sensors were built. This let us test both the motors and the sensors using separate programs.
Week 10:	Worked on some final touchups, completed the code and tested delay and pulse times for the motors. Made the things run around and find eachother.

Appendix C

Appendix D