**University of Colorado Boulder**

**ECEE Department**

**ECEN 2270** **Electronics Design Lab** – **Spring 2024**

**Instructor:** Mr. Steven Dunbar

**Lab 5**

**Date of Submission:** 05/05/2024

**Team:** Yankee

**TA:** Mia Tomasulo

**Names:** Samiha Abonty, Matteo Coscia, Arjun Dalwadi, Matthew Gorbold

# What We Implemented

## Hardware

We made a lane-assist system for our robots. This consisted of designing, testing, and manufacturing custom infrared sensors and writing the code to use those sensors as eyes for the robot which could see the difference between black and white.

To build the IR sensors, we first referred to SparkFun, which is known for publishing the schematics for their various sensors and hardware modules. We found the QRE1113, which is an infrared sensor that can either be used as a proximity/motion sensor, or to build a line-following robot, the schematic for which is in Fig. 1.

A diagram of a circuit

Description automatically generated

Figure : The schematic for SparkFun's QRE1113 infrared sensor.

The diode is an IR transmitter, and the other component is a phototransistor. These parts were available for free in our kits and in the ITL, so we soldered this equivalent schematic onto protoboards. The wiring diagram is shown in Fig. 2 and the finished product is shown in Fig. 3.

A diagram of a circuit

Description automatically generated

Figure : The wiring diagram for the IR sensors.

A hand holding a green circuit board

Description automatically generated

A hand holding a green circuit board

Description automatically generated

Figure : The top (above) and bottom (below) of the completed IR sensor.

We made two sensors, mounted on the front bumper on each side, shown in Fig. 4.

A close up of a toy

Description automatically generated

Figure 4: The IR sensor mounted on the robot. The transparent component is the IR transmitter, and the black component is the phototransistor.

Through simple testing using *analogRead()* on the Arduino, we found that the IR sensor output approximately 40 when seeing white and approximately 600 when seeing black. However, we noticed that the numbers changed when we ran the robots in different environments, e.g. a different room with dimmer lighting. Thus, we designed a comparator circuit using the TLC3702 and a potentiometer to be able to adjust the sensitivity of the sensor at will and implement interrupt service routines in our programming. The schematic is shown in Fig. 5.

A diagram of a circuit

Description automatically generated

Figure 5: The complete schematic for the input system to the Arduino. The comparator positive power supply is +5V. Note that the nodes into the positive inputs of the comparators are realized by jumper wires.

Changing the potentiometer’s resistance allows us to change the reference voltage for the comparator at will. For example, if we are in a brightly lit room, we can adjust the potentiometer so that the sensitivity of the comparator circuit is a bit lower, thus eliminating false “black line detected” signals. Fig. 6 is an image of the comparator circuit implemented in hardware.

A circuit board with wires and switches

Description automatically generated

Figure 6: The comparator circuit implemented in hardware.

Finally, we wanted to add visual and audio feedback when the robot sees the edge of the lane (black tape), just as in modern cars. This is easily accomplished using a piezoelectric buzzer in series with an LED, and the schematic is shown in Fig. 7.

A diagram of a yellow led

Description automatically generated

Figure 7: The schematic for the visual/audio feedback. R = 330 Ω.

In order to generate a sound with the piezoelectric buzzer, we must use the *tone()* function, which outputs a square wave with 50% duty cycle. The frequency is an argument which can be passed into the function, which dictates the pitch of the buzzer. We settled on 2 kHz, which generated a sufficiently loud sound while being only slightly annoying, as lane-assist systems are. This also illuminates the LED for as long as the buzzer is toned, and since the frequency is so high, it appears to be solid-on. We took advantage of the LED-lighting paradigm of using a PWM signal by piggybacking off of the necessary signal for the buzzer, limiting space usage. The implementation in hardware is shown in Fig. 8.

A circuit board with wires and wires

Description automatically generated

Figure 8: The visual/audio feedback in hardware.

There is one of these on each side of the robot, corresponding to a digital high signal from the IR sensor on their respective side.

## Software

We wrote ISRs for the outputs of the comparator circuits and used them to toggle flags. Each side had its own flag, and when an interrupt was detected, that ISR would set the flag for that side to HIGH. We designed our code such that the interrupt would be handled in the main loop by keeping the code in our main loop lean. Thus, we can guarantee the interrupt sequence would be handled before the robot went too far out of the lane because the main loop iterates so quickly.

Depending on which side’s ISR was executed, the robot would accelerate that side’s motors to gently return itself to in between the lanes. This is a time-based execution, as we don’t want the robot to barely get off the line and quit adjustment. If, for some reason, the movement is executed and the robot is still on the black line, the ISR will be triggered again, and the robot will adjust again.

The full code can be found in Appendix A.

# Further Additions

Given more time, we believe we could add merging and parallel parking using the hardware we’ve already manufactured and implemented, in addition to a Bluetooth-supported Arduino. It would only be an addition to the software, but we’d like a chip with a faster clock to be able to never miss an event.

Merging is straightforward, as this could be implemented in a number of ways. Using Bluetooth communication, we could send the instruction to merge left or right to the robot. Depending on the instruction, the robot would adjust the motor speeds on either side to begin veering into the desired lane. When the opposite side’s IR sensor throws an interrupt, the robot understands it is now completely in the next lane and will straighten out.

Parallel parking is a similar implementation to merging but it requires proximity sensors in addition to the Bluetooth-supported Arduino and two additional IR sensors for the back of the car. The proximity sensors would be situated at the front and back of the car for obstacle avoidance (other parked cars) and the two additional IR sensors on the back of the car would be used for positioning. Since the robot now must accurately move backwards, we may need those two extra IR sensors for accurate vectoring.

# Successes and Obstacles

## Successes

This project was overall a success. The components were easy to understand, and their testing and implementation was simple. Nothing really went wrong throughout the development of this robot. All of our circuits and code worked the first time, save a few tweaks and additions, such as the potentiometer in the comparator circuit. We applied the hardware and programming techniques we learned throughout the semester and our robots performed accurately and consistently. We are very happy with how our final robots turned out.

## Obstacles

As aforementioned, there weren’t any significant obstacles. The biggest problem we encountered was with the variation in behavior of the IR sensors in environments with different lighting, however slight. However, the integration of the potentiometer in the comparator circuit solved that problem efficiently and elegantly and brought an additional feature to our robots.

Another issue we faced was brought on entirely by human error. After testing the orientation of the phototransistors, we’d accidentally solder them into the protoboards in the wrong orientation. This happened more than it should’ve, but in the end we figured it out.

# Our Takeaways and Advice for Future Students

## Our Takeaways

Throughout this semester, we learned a lot about hardware and software techniques in the testing and development of a robot system. We jumped through many hoops throughout the development of the robot, and sometimes we weren’t always sure if what we were doing was necessary. However, this final project gave us the opportunity to freely design, test, and manufacture our own circuits to carry out specific functions. We used *SIMetrix*, *Waveforms*, and various measurement techniques with both the AD3 scope and the DMM. On the software side, we learned about ISRs and how to ensure our robots would work as intended in regard to clock speed and machine cycles. Overall, we spent all semester honing various skills in hardware design, and in this project, we got to truly appreciate the importance of them and recognize how much we’d learned over the course of the semester.

## Advice for Future Students

The following is a summarizing list of key points from each member of our group:

* Set milestones within labs to ensure you’re on track.
* Stay on top of those milestones.
* Work on the robot outside of lab.
* Work closely with your groupmates whenever you can: it eliminates silly errors and mistakes which can cost you a ton of time.
* Ask and/or attend office hours if something doesn’t quite make sense. Everything in this class builds on itself.
* Write the lab reports as you go, this makes it easier to stay organized as a group.
* Sometimes you’ll get frustrated, and that’s okay. Just walk away and take a 10-minute break, and when you come back it’ll be a lot easier to catch a mistake.

# Appendix

**Article A – Arduino Code**

volatile bool leftBlack = LOW;

volatile bool rightBlack = LOW;

const int pinLeftBlack = 20;

const int pinRightBlack = 21;

const int leftIndicator = 14;

const int rightIndicator = 13;

const int pinRightForward = 7;

const int pinRightBackward = 8;

const int pinRightPWM = 9;

const int pinLeftPWM = 10;

const int pinLeftForward = 11;

const int pinLeftBackward = 12;

void setup() {

pinMode(pinRightForward, OUTPUT);

pinMode(pinRightBackward, OUTPUT);

pinMode(pinRightPWM, OUTPUT);

pinMode(pinLeftForward, OUTPUT);

pinMode(pinLeftBackward, OUTPUT);

pinMode(pinLeftPWM, OUTPUT);

digitalWrite(pinRightForward, HIGH);

digitalWrite(pinRightBackward, LOW);

analogWrite(pinRightPWM, 0);

digitalWrite(pinLeftForward, HIGH);

digitalWrite(pinLeftBackward, LOW);

analogWrite(pinLeftPWM, 0);

pinMode(pinLeftBlack, INPUT);

attachInterrupt(digitalPinToInterrupt(pinLeftBlack), service20, RISING);

pinMode(pinRightBlack, INPUT);

attachInterrupt(digitalPinToInterrupt(pinRightBlack), service21, RISING);

pinMode(14, OUTPUT);

pinMode(13, OUTPUT);

Serial.begin(9600);

}

void loop() {

if (leftBlack == HIGH) {

moveForward(5, 2);

tone(leftIndicator, 2000);

delay(750);

noTone(leftIndicator);

leftBlack = LOW;

}

else if (rightBlack == HIGH) {

moveForward(2, 5);

tone(rightIndicator, 2000);

delay(750);

noTone(rightIndicator);

rightBlack = LOW;

}

else {

moveForward(2, 2);

}

}

void service20() {

leftBlack = HIGH;

}

void service21() {

rightBlack = HIGH;

}

void moveForward(int ls, int rs) {

analogWrite(pinLeftPWM, ls \* 51);

analogWrite(pinRightPWM, rs \* 51);

}