IEEE MICROMOUSE

Workshop Week 5:

Analog Read and Infrared Sensors

Objective:

Learn the difference between analog and digital voltage signals. Use the analogRead function on the Arduino to read infrared sensor data. Learn how an infrared sensor LED works for both the transmitter and receiver. Use the infrared sensors to do basic collision detection.

Background information:

So far you have been dealing with digital signals to control a LED, read a button press and control the motors. These are straightforward ways to use the Arduino, since it only involves 2 voltage levels, HIGH or 5 volts and LOW or 0 volts. You can do a lot with only digital signals, but real-world variables cannot be expressed with only 2 values. As an example, think about measuring the temperature of the environment with a digital signal. You could only represent the temperature as either hot or cold, there would be no way to express anything in between. We need to expand our measurement system to accept more than 2 values and luckily for us, the Arduino has a function that does that. Now, this is where knowing the difference between an analog signal and a digital voltage signal is important. Like previously mentioned, a digital voltage signal has only 2 values. For an analog signal there is an infinite amount of voltage values (you could have a voltage of 5V, 6.12321V, 3.1415V or any real number you could think about). The Arduino can read these values by using an *Analog to Digital Converter* (ADC) through a simple function.

At this point you might be asking, how this is helpful or relevant to micromouse? Well, we need to measure distance (or at least approximate it) with infrared sensors, and these sensors produce an analog voltage signal. The infrared sensors on the micromouse work by emitting infrared light using an infrared LED and then capturing the reflected light with a phototransistor.

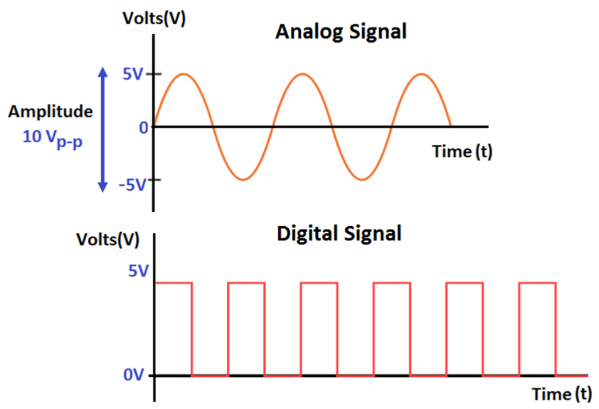

Figure 1 Analog and digital signals. Observer how the analog signal has an infinite amount of voltage values between -5V and 5V

There are four infrared LED pairs on the micromouse, and they will help us to detect if there are side and front walls on the maze. For now, we are going to focus on writing code that makes the infrared transmitter and receiver work. Then we will expand this code to avoid objects using the forward pointing infrared LED pairs.

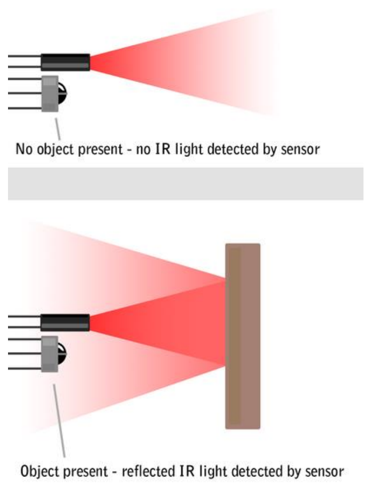
No object present - no IR light detected by sensor
Object present - reflected IR light detected by sensor

Figure 2 Image showing the principle in which the infrared LED pairs work on the micromouse

Infrared transmitter

The infrared transmitter must send a short high energy light pulse. To do this, we can use a digital pin to turn on and off the transmitter in short bursts.

```
#include "micromouse.h"

void setup() {
  pinMode(IR0_TX, OUTPUT);
  digitalWrite(IR0_TX, LOW);
}

void loop() {
  digitalWrite(IR0_TX, HIGH);
  delayMicroseconds(150);
  digitalWrite(IR0_TX, LOW);
}

```

WARNING: DO NOT EXCEED MORE THAN 150 MICROSECONDS OF PULSE TIME. Ignoring this will likely damage the transmitter LED.

Infrared receiver

To read the reflected light, we need to use the analogRead function of the Arduino. Analog pins on the Arduino are enabled by the default and do not need to be configured at startup. We are required to measure the reflected light immediately because the pulse duration is extremely short. We will place the analogRead function immediately after we wait for the delay and before we turn off the transmitter infrared LED. We can also use the serial console to read the values received for the emitted light.

```
#include "micromouse.h"

void setup() {
  pinMode(IR0_TX, OUTPUT);
  digitalWrite(IR0_TX, LOW);

  Serial.begin(9600);
}

void loop() {
  digitalWrite(IR0_TX, HIGH);
  delayMicroseconds(150);
  int value = analogRead(IR0_RX);
  digitalWrite(IR0_TX, LOW);

  Serial.println(value);
}

```

If you run the previous code, you will see that the values vary between 0 and 1023. These values signify the amount of light that it is capture by the receiver infrared LED. The higher the value the more infrared light the LED is receiving and the closer the mouse is to a surface. Pretty cool right!

Infrared code refactoring

The previous code works as we expected, but we still need to control the other 3 infrared LED pairs. We could simply copy and paste the previous code and tweak it for the other LEDs, but this will be extremely messy, and it will have lots of repetition that we can avoid with set of functions. Using the same principles as last week, let's create a set of functions that read the infrared LEDs. We are only going to show you how to do it for the front pair of LEDs, you will have to do the rest as an exercise.

On the micromouse.h file have these line of code added to the end

```
#ifndef MICROMOUSE_H_
#define MICROMOUSE_H_

#include <Arduino.h>
#include <stdbool.h>

/*Motor definitions*/
#define ENABLE            A6
#define MOTOR_0_1A        11
#define MOTOR_0_2A        5
#define MOTOR_1_1A        6
#define MOTOR_1_2A        9
#define ENCODER_0_CHA     2
#define ENCODER_0_CHB     13
#define ENCODER_1_CHA     3
#define ENCODER_1_CHB     8

/*Infrared sensors definitions*/
#define IR0_TX       A1
#define IR1_TX       12
#define IR2_TX       7
#define IR3_TX       4
#define IR0_RX       A0
#define IR1_RX       A7
#define IR2_RX       A2
#define IR3_RX       A3

/*Extras*/
#define BUTTON      10

void motors_init(void);
void motor_0_speed(int speed, bool direction);
void motor_1_speed(int speed, bool direction);
/*NEW CODE ADDED BY YOU************************************************/
/*New IR read functions*/
int ir_0_read(int delay);
int ir_1_read(int delay);

#endif

```

On the micromouse.cpp file add this code to implement the new ir_0_read function

```
#include "micromouse.h"

void motors_init(void)
{
  pinMode(ENABLE, OUTPUT);
  pinMode(MOTOR_0_1A, OUTPUT);
  pinMode(MOTOR_0_2A, OUTPUT);
  pinMode(MOTOR_1_1A, OUTPUT);
  pinMode(MOTOR_1_2A, OUTPUT);

  /*Enable motors by default*/
  digitalWrite(ENABLE, HIGH);
}

void motor_0_speed(int speed, bool direction)
{
  if(direction == true)
  {
    /*Move motor forward at given speed*/
    analogWrite(MOTOR_0_1A, 0);
    analogWrite(MOTOR_0_2A, speed);
  }
  else
  {
    analogWrite(MOTOR_0_1A, speed);
    analogWrite(MOTOR_0_2A, 0);
  }
}

void motor_1_speed(int speed, bool direction)
{
  if(direction == true)
  {
    /*Move motor forward at given speed*/
    analogWrite(MOTOR_1_1A, speed);
    analogWrite(MOTOR_1_2A, 0);
  }
  else
  {
    analogWrite(MOTOR_1_1A, 0);
    analogWrite(MOTOR_1_2A, speed);
  }
}
 /*NEW CODE ADDED BY YOU***************************************************/
/*New IR 0 read function*/
int ir_0_read(int delay)
{
/*If delay is bigger than 150 us, clamp the value to this limit*/
  if(delay > 150)
  {
   delay = 150;
  }

  digitalWrite(IR0_TX, HIGH);
  delayMicroseconds(delay);
  int value = analogRead(IR0_RX);
  digitalWrite(IR0_TX, LOW);

  return value;
}
/*New IR 3 read function*/
int ir_3_read(int delay)
{
  /*If delay is bigger than 150 us, clamp the value to this limit*/
  if(delay > 150)
  {
   delay = 150;
  }

  digitalWrite(IR3_TX, HIGH);
  delayMicroseconds(delay);
  int value = analogRead(IR3_RX);
  digitalWrite(IR3_TX, LOW);

  return value;
}

```

Now in your main code you can call these set of functions to read the sensor values for the front infrared LEDs

```
#include "micromouse.h"

void setup() {
  pinMode(IR0_TX, OUTPUT);
  pinMode(IR3_TX, OUTPUT);

  digitalWrite(IR0_TX, LOW);
  digitalWrite(IR3_TX, LOW);

  Serial.begin(9600);
}

void loop() {
  /*Read infrared LED values. Only front leds*/
  int ir0 = ir_0_read(150);
  int ir3 = ir_3_read(150)

  /*Print values to serial console*/
  Serial.println(ir0);
  Serial.println(ir3);
}

```

Way cleaner and easier to understand than just copying and pasting code into the loop function!

Basic collision avoidance

Now that we have the basics under our belt, we can do some basic collision avoidance with a simple stop, turn, and move algorithm. The idea is to measure the front (IR0 and IR3) infrared sensors, average their values (as we did with the encoders), then see if we exceed a value threshold. If we do exceed this value, we stop the mouse, turn to either the right or the left and then keep moving. This is not particularly useful robot, but we can see how sensors and actuators interact towards a basic autonomous goal, which is not to crash. The pseudo code to do this goes as follows

```
/*This micromouse workshop will teach you:
 * 1. Infrared sensors
 * 2. Basic navigation
 */

#include "micromouse.h"

const int ir_threshold = 800;

void setup()
{
  /*Initialize motors and infrared sensors*/

  /*Wait for button press*/

 /*Start moving mouse*/
}

void loop()
{
  /*Read front infrared sensors*/

  /*Get average from front infrared sensors*/

  /*Use this average to stop if colliding with an object.
  if ir_average > ir_threhsold*/
  {
    /*Stop*/
    /*Delay 500 ms for turn to complete*/
  }

  /*Keep driving forward*/

}

```

Exercise:

Implement the pseudo code mention on the previous section. Finish writing the code for the ir_1_read, ir_2_read functions. Add a ir_init function to initialize the infrared transmitters.