REMOTE CONTROLLED SMART ICE CREAM TRUCK (WITH MUSIC PLAYER)

PROJECT 4

ARTHUR MAKUMBI

CS342 FALL 2018

horizontal line

## Abstract

My goal this project is to try and pull everything together. I want experiment and get comfortable with DC motor control, analog to digital conversions, pulse width modulation, and Infrared Remote control, while also configuring, triggering, and servicing interrupts. In this project, I will program my board to interact with digital and analog inputs (buttons, IR receiver, and IR distance sensor) and outputs (motors, piezo element,, and LEDs) to create a remote controlled smart ice cream truck. The resulting product will be remote controlled and able to move in any direction while also being able to come to a halt as it approaches an obstacle. It will also play some lovely jolly ice cream tunes as it travels. One will set the smart ice cream truck in motion by pressing a button either on the remote or on the board.

I have designed this project because I have always loved ice cream trucks, especially the music they play, since my childhood. Additionally I am excited about automation, especially the kind aided by data from sensors.

## 

## Resources

### Parts list

* Any computer with the Arduino IDE installed (x1)
* Metro Mini development board (x1)
* USB-micro USB data cable (x1)
* insulated copper wire (40 cm should do it)
* full-sized breadboard (x1)
* LEDs (x2)
* resistors in the
* [500 Ω, 2.2 kΩ] range (x2)
* resistors in the [2.2 kΩ, 10 kΩ] range (x2)
* L293D H-bridge (x1)
* DC barrel jack (x1)
* DC motors (x2)
* TSOP382 IR receiver
* IR remote control
* piezo element (x1)
* Battery pack (x1) with 4 rechargeable AAs
* Push button (x1)

Chassis

* Wheels (x2)
* Cardboard rectangle (x1)
* Large binder clip (x1)
* Twist ties (x4) ( or spare wire)
* Double-stick tape (1-2 in2)
* Rubber band (x1)

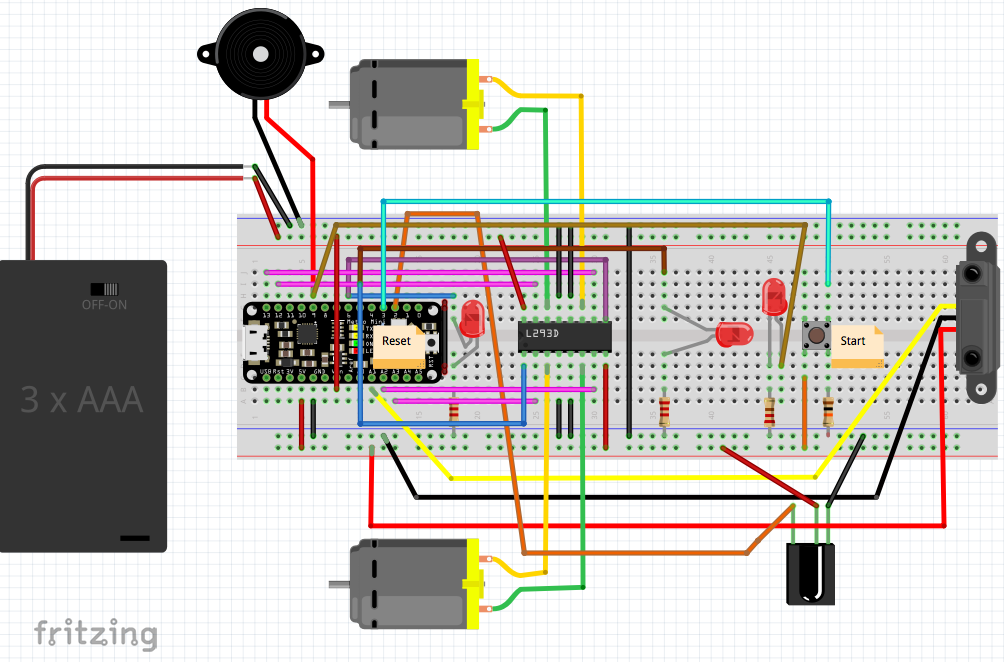
### References

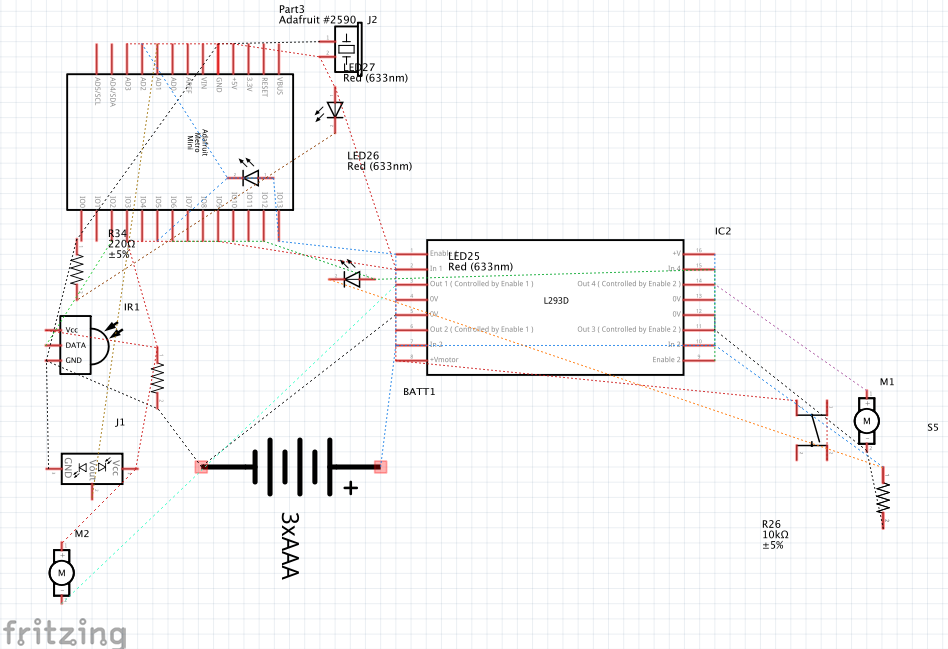
* ATmega datasheet
* IR sensor datasheet

## 

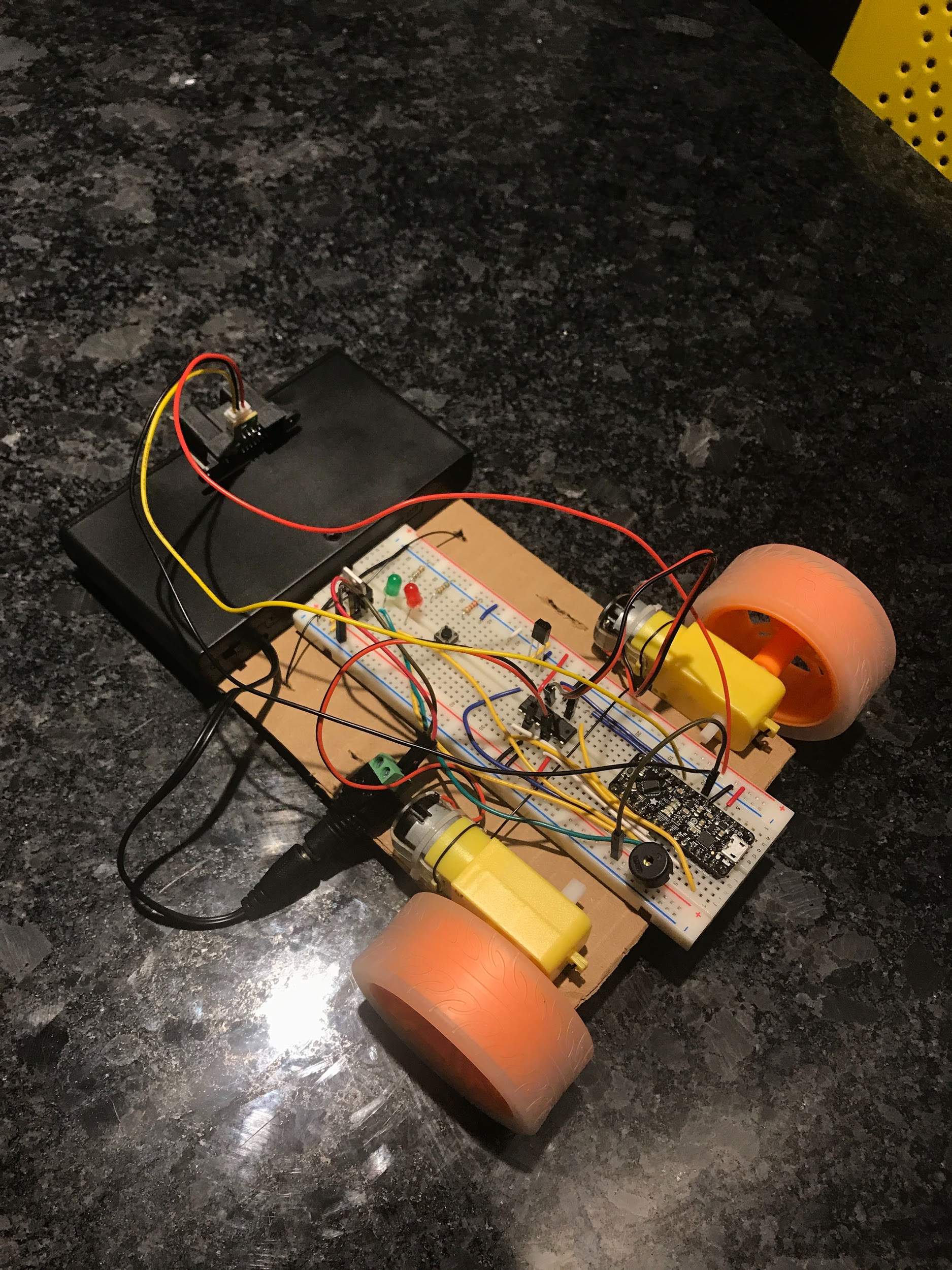
## Procedure

### Rough sketch





Given the above circuit, I construct a chassis for it using the material listed above. I attach the circuit to the chassis using tape and wire. The resultant rover is shown below:



For this project, I build on top of my previous project. I started off by wiring a push button (input) to PD3 by writing to the GPIO registers, and connect it in series with a 10 kΩ resistor, a “pull down resistor”. This is to ensure that when no high voltage is arriving on the input side of the button, this resistor “pulls” the button’s voltage down to 0 V, and does not let much current leak to GND in the case that there is a voltage arriving. I then configure PD3 to be a start / stop button for the whole system, by triggering an external interrupt on INT1 that toggles between start and stop. Using one button for both on and off helps me avoid wasting pins and buttons.

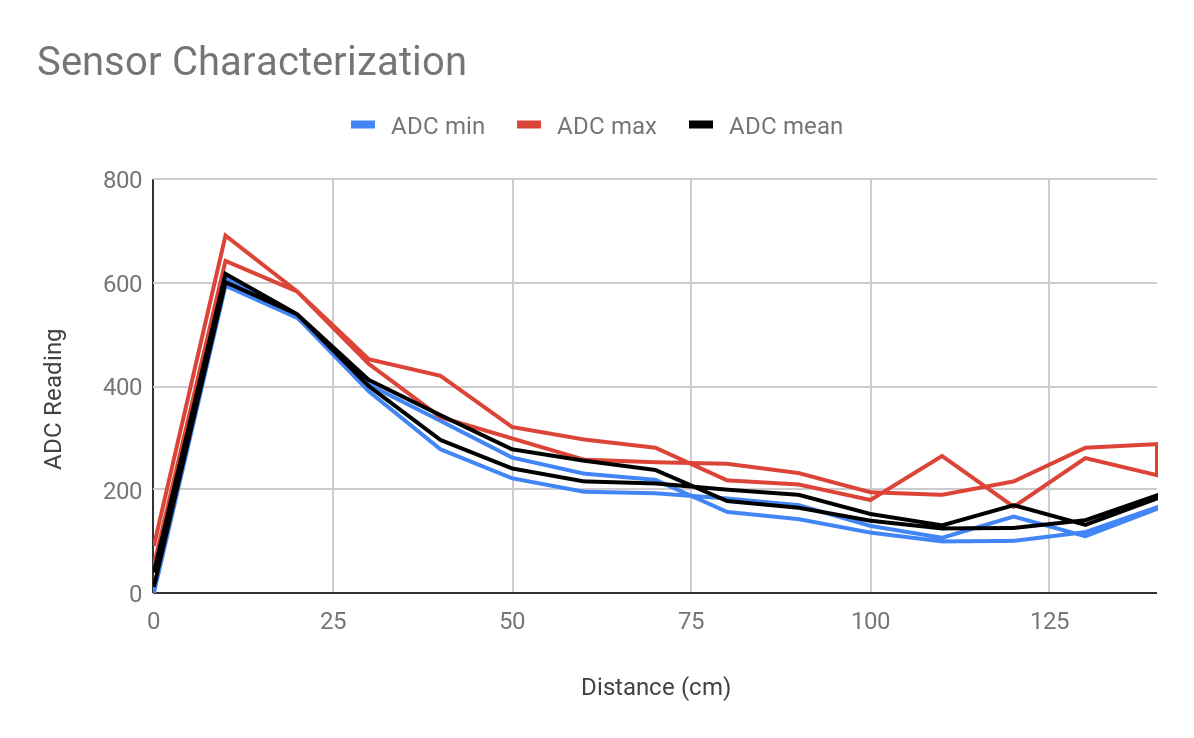
I then wired two DC motors (outputs) to my board by the help of the L293D H-bridge as shown above. Lab 6 helped me a lot in this step. However, turns out I had to swap the leads on the left motor to get it to spin in the correct direction. PD6 controls when the left DC motor rotates (it is the enable) and PB[4:5] control its direction. PD5 is the enable, and PC[2:3] control the direction for the right DC motor. I leverage Pulse Width Modulation (PWM) to control both motors whereby Timer 0 drives both the motors. I configured it to be in non-inverting mode, that is they clear on compare match and set on bottom. This makes it easier (more intuitive) to control the speed of the motors using the value in OCR0A and OCR0B. As the value in OCR0A increases, the motors’ speed increases since the duty cycle would be higher. I additionally set a prescaler of 1024 for added accuracy as we control the motors since the clock controlling the duty cycle would have many more subdivisions.

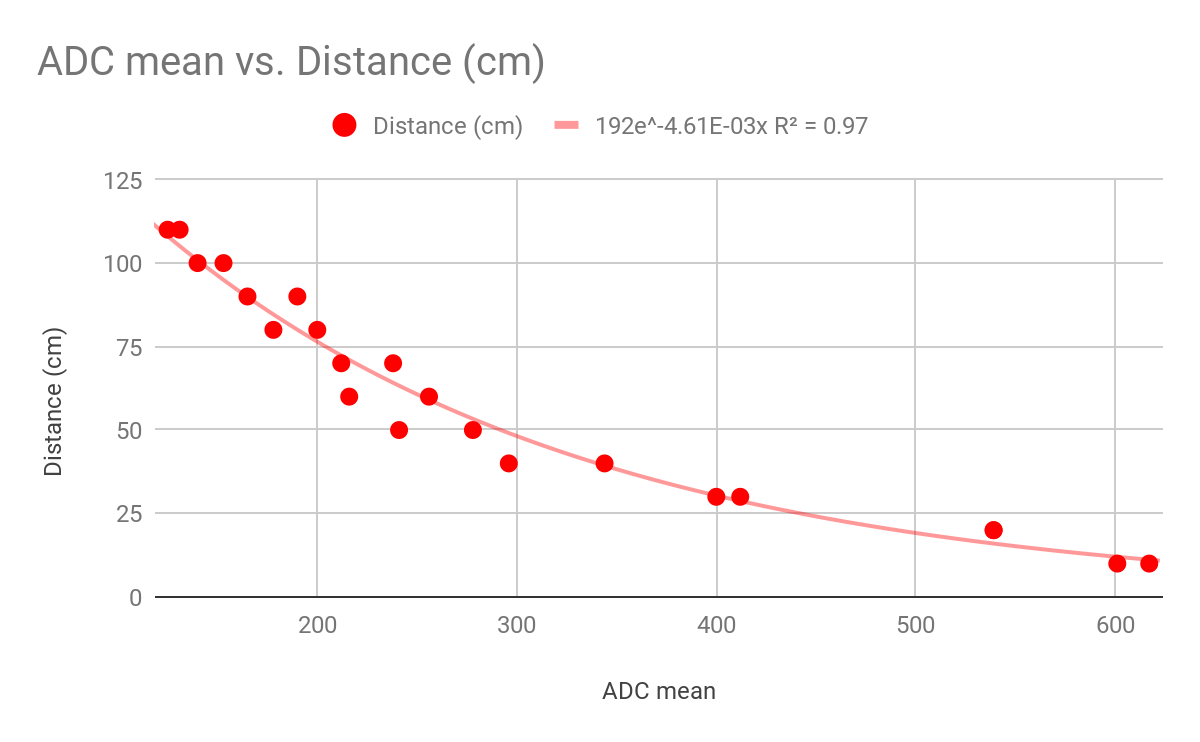
I then added two LEDs (outputs) connected to PD6 and PD5 to detect when the motors are on and working correctly. I wired each of them in series with a 2.2Ω resistor to limit overall power consumption, and avoid short circuits. They both light up when the motors are running.

Next step I did was configuring an IR distance sensor (input) to my board which collect my data for my Analog to Digital Conversions (ADC). I started by setting up a compare match interrupt A on timer0 to autotrigger an analog read from the IR distance sensor, which is connected on pad A1. I went ahead and disabled the other analog input pads to save energy. This is important since my available energy is limited since all I have is a battery pack.

I handle all my interrupts in function calls in my code. I do not do anything in the timer0 compare match interrupt A while the INT1 interrupt toggles the start state to let my program know when to run the rover or stop it. The ADC interrupt processes the analog input from the IR sensor and computes the mean value by using a sampling window of 10. Thanks to Lab 8, I already produced a “clean” characterization for my IR distance sensor. I did this by taking twenty ADC samples from the IR distance sensor 10cm apart, starting from 0cm first in increasing order then in decreasing order and then plotting the mean, min and max ADC values. Knowing my monotonic region for my IR sensor starts from about 20 cm, I configure my ice cream truck to stop the rover when an obstacle is about 20cm away. This is about 590 IR sensor units, meaning if the computed mean ADC value is greater than 590, we halt the rover. For my setup, I halt the rover by setting the OCR0A and OCR0B values to zero, otherwise they are both set to 60.

My characterization graphs and equation are shown below:





Adding sound was the next step in my journey to make an ice cream truck. I started by I wiring 1 piezo element (output) to PB[1], in series with a 2.2 kΩ resistor. This further limits the overall power consumption of my circuit. It is this piezo element that generates the sound of my ice cream truck. I then configured timer1 for waveform generation by switching it into CTC mode, toggling PB[1] on compare match while prescaling the clock by 1. I use TCCR1A and TCCR1B to achieve this. Additionally I use this timer to generate an interrupt every 0.001 seconds to sample the music notes list. I create a helper interrupt routine that handles which note is being played at a given point during the program’s runtime. The interrupt checks whether the program is in the play state, and depending on how much time has passed, it decides which note to play by setting a generated period value into OCR1A.

I then move on to incorporate the IR receiver to my ice cream truck and I use the IRLIB2 library to handle infrared communication. I start by declaring the IRRecv object to use PD2 on my circuit. I then created a loop that handles whichever IR remote button is pressed. The select button starts the truck, up button moves the truck forward, down stops, left moves it left, right moves it right, 0 is the power off button, 5 plays the fire tunes, and 8 stops music playback.

All my code is written in c++ and it is included among files that I handed in.

## Results

My Ice cream truck was indeed responsive as I set it up. Demonstration of results are attached below:

1. [Testing without chassis](https://www.youtube.com/watch?v=EdQMJ_YeUR4)
2. [Wall detection](https://www.youtube.com/watch?v=4UNA3WLgEQM)
3. [Ice cream truck, the whole SHEBANG](https://www.youtube.com/watch?v=Mzk3WGeWRHc)
4. [bloopers](https://www.youtube.com/watch?v=PjfigHYDUeU)

As seen in the videos above, when you push the start button, my ice cream truck responds to remote button presses: pressing up moves it forward and stops when it gets too close (about 15-20 cm) from an obstacle (this means my IR distance sensor ADC is working correctly). I tried it with walls and a chair as demonstrated above. Additionally, pressing left moves it left, and right moves it right, and pressing down stops it. You can also hear the fire tune play when you press 5, and you can pause it by pressing 8.

## Discussion

Considering the level of complexity of my setup, I think it do a good job considering how noisy the IR distance sensor and receiver can be. However, as seen in my bloopers, when the rover is moving too fast, it cannot react to an obstacle fast enough and will crash. My workaround for this was reducing the truck speed. However, I think this might be solved by using better online filtering and using a smaller sampling window. Auto-triggering an analog read from ADC more frequently might also help. However, this might make my system’s responsiveness even slower despite making it more accurate.

Additionally, my chassis quickly succumbed to the weight of the battery pack which was inconvenient. This can be solved using a firmer chassis.