Arduino Car Distance Sensors

Nicu-Serban POP

Polytechnic University Timisoara

3rd Year CTI-En

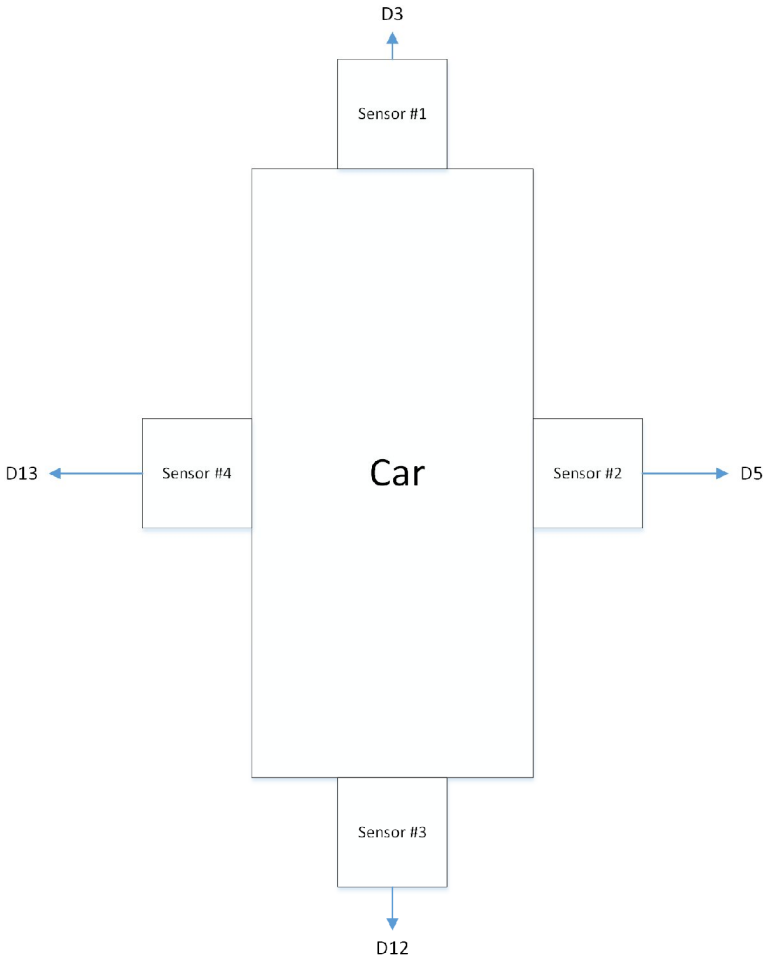
1. Introduction

The project is an application used very often in the automotive domain. The project is based on a modified R/C car. The R/C module was replaced by new circuits containing motor drivers, joystick controller and distance sensors, all controlled by an Arduino Uno. We use a simple buzzer that beeps according to the distance from all four sensors.

On the software side we use threads for controlling the motors, getting data from the four sensors and beeping the buzzer. The threads method was initially used only for the buzzer but was later extended to other modules as well.

2. Architecture

The platform used for this project is an Arduino Uno Rev.3. Power for the Arduino is supplied by an external power bank and four AAA batteries are used for driving the two motors.

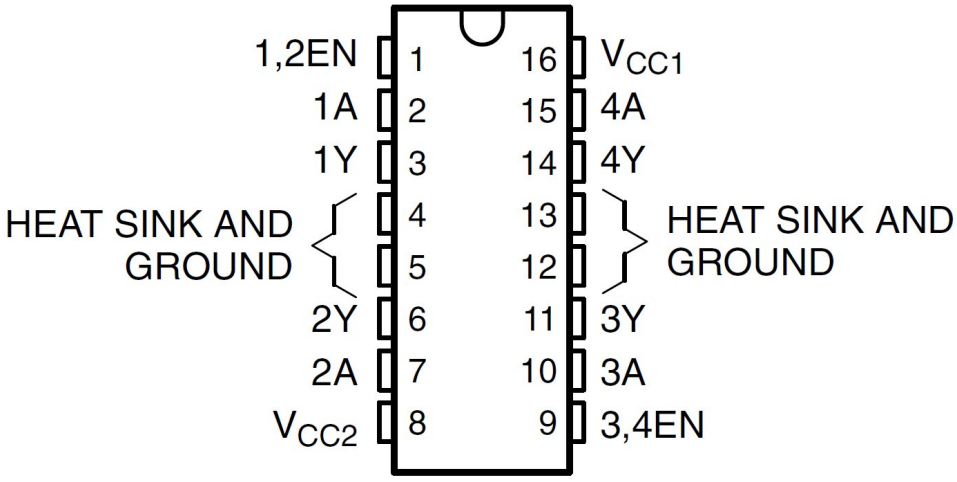


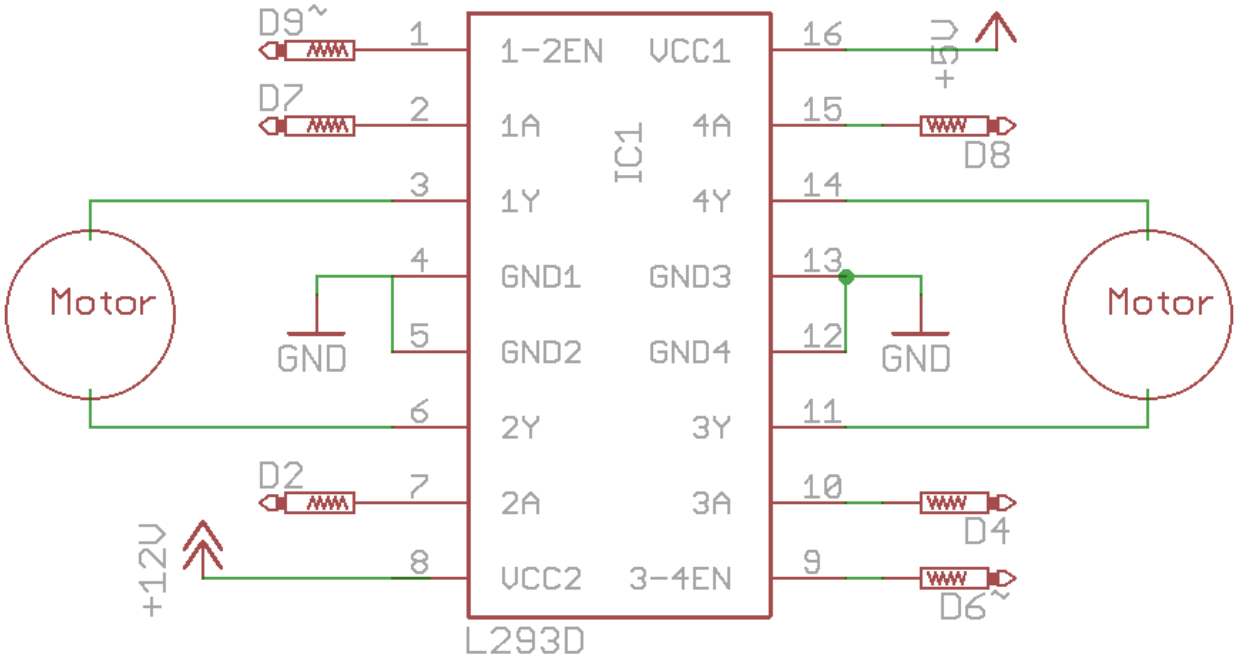
- D3 – Connected to Arduino Digital Pin 3
- D5 – Connected to Arduino Digital Pin 5
- D12 – Connected to Arduino Digital Pin 12
- D13 – Connected to Arduino Digital Pin 13

3. Hardware Implementation

a. Motor Driver

The driver used for driving the front and rear motor is the L293D, a quadruple Half-H driver. We used this type of driver because we are using two bidirectional motors. The front one for left and right steering and the rear one for forward and backward movements, since the need of a quadruple driver. The motor diagrams are the following:

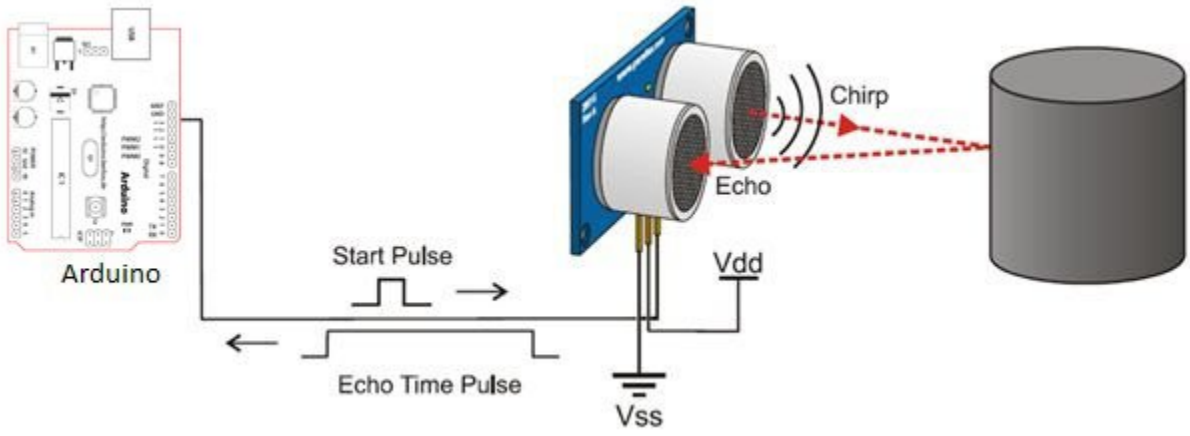




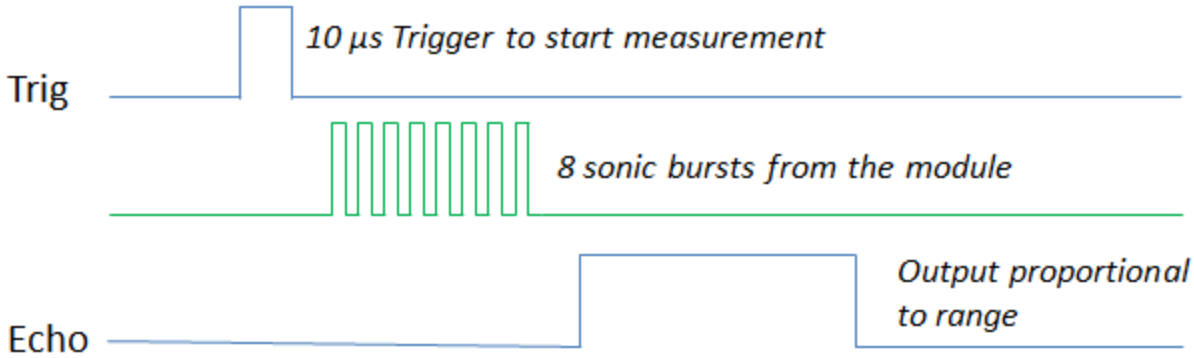
b. HC-SR04 ultrasonic distance sensor

The distance sensor we are using is the HC-SR04 one. We chose to use this one because it is very flexible and easy to implement and interface with other devices. We use four of them, one for the front, one for the back and two for the sides of the car. The sensor works by emitting an ultrasonic signal and listening for its echo. After the echo was received, the time between the sent signal and the received signal is converted into distance. The module automatically sends eight 40 kHz and detect whether there is a pulse signal back. Sensor diagram, connection and timing diagram are below.

- The way the sensor works:



- Timing Diagram:

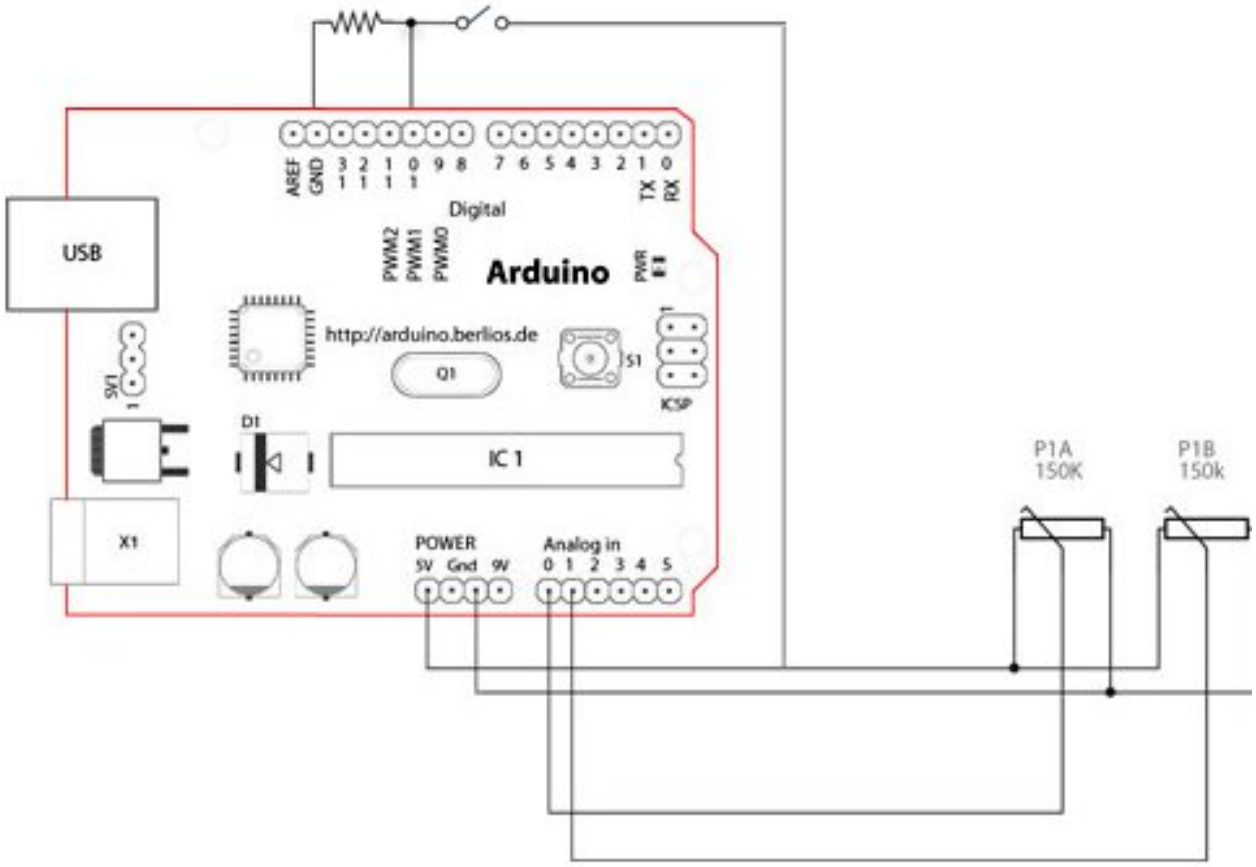


c. Joystick

In order for us to control the RC car we use a simple, dual axis joystick as shown in the picture below. The joystick consists of two potentiometers and a push button. The X and Y axis are connected to the Arduino analog input A0 and A1in order to receive the analog voltage the joystick outputs and is then converted to digital values by the Arduino itself, with a range of 10 bits (1024 digital values, from 0 to 1023).

- Joystick:

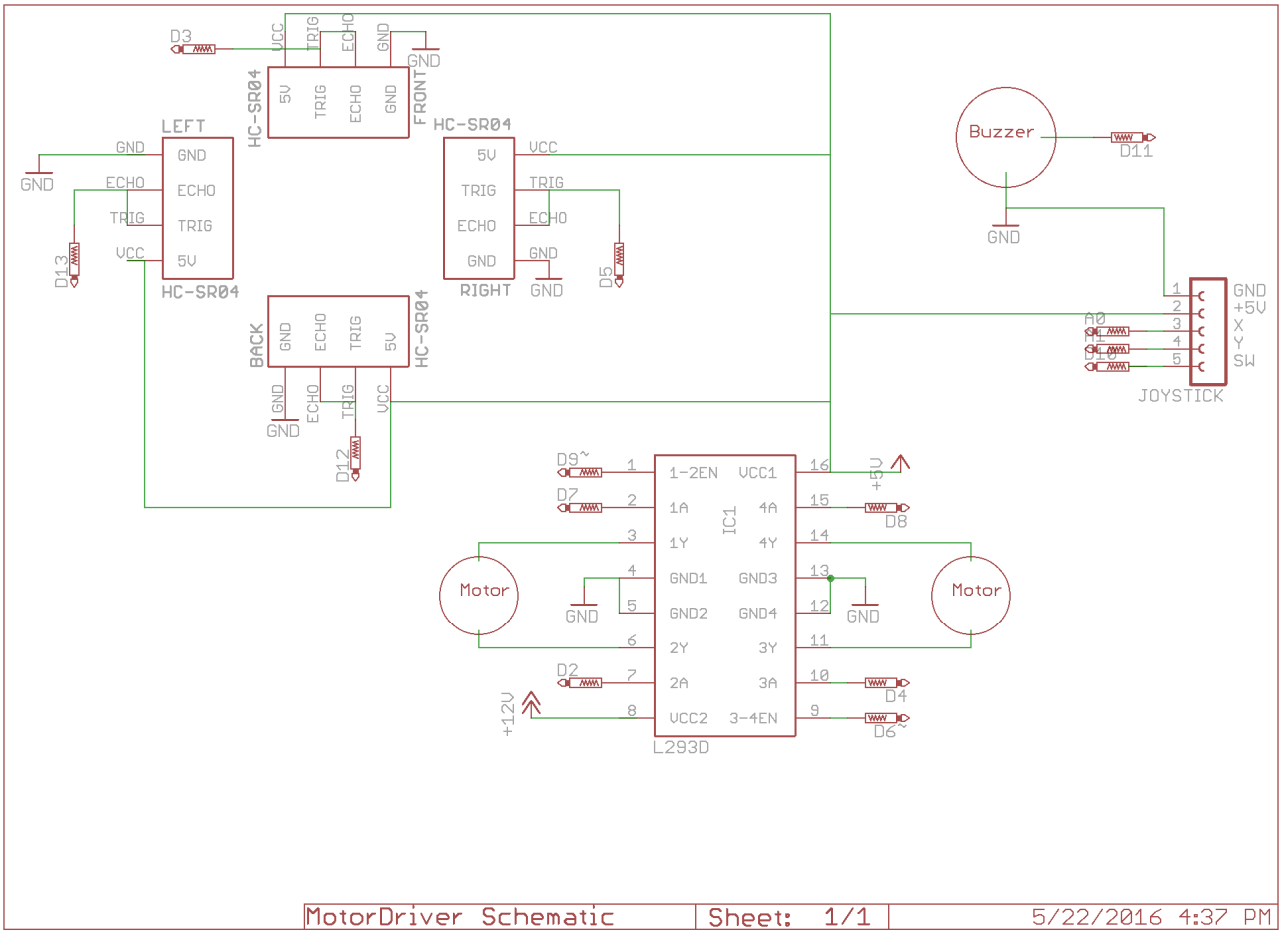
- Joystick Schematic



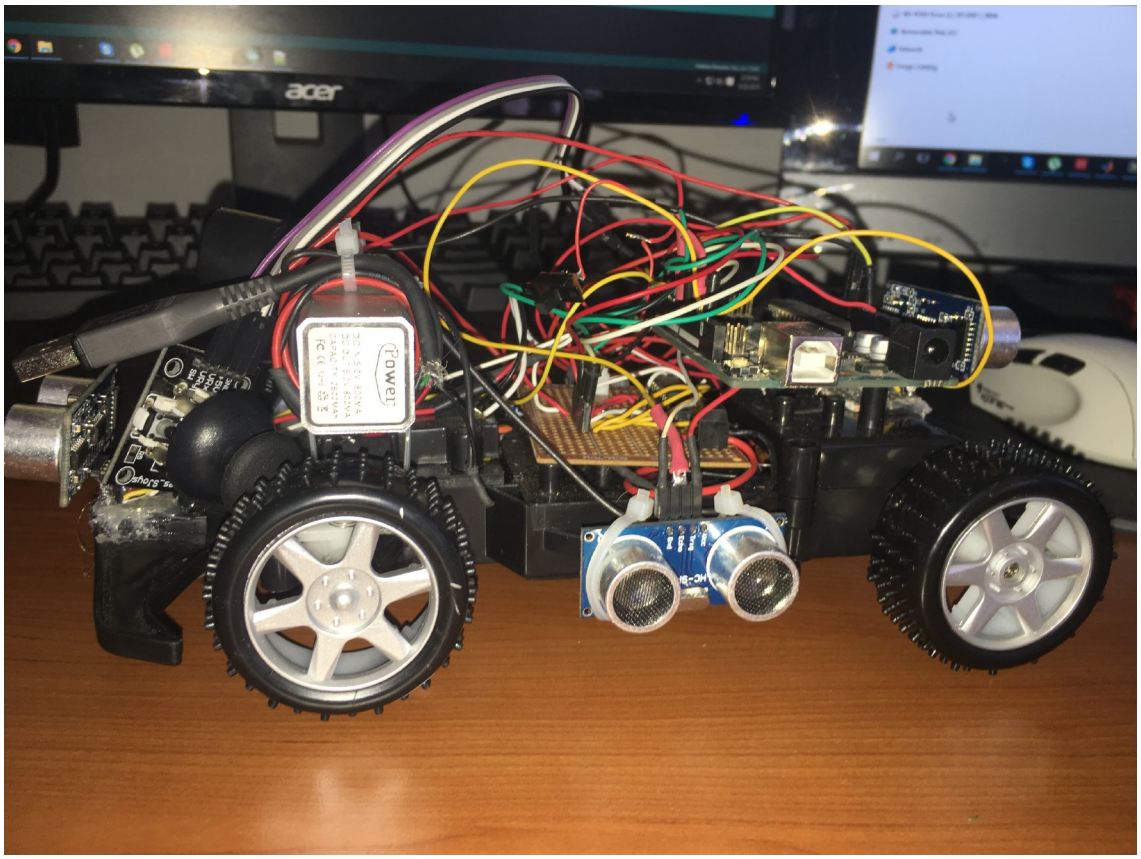
d. <u>Combined Hardware</u>

Below is a schematic with all the connected hardware and a picture of the car itself. More images and a video presenting the project can be found in the project directory.

- Schematic:


D3
VCC
TRIG
ECHO
GND
GND
HC-SR04
5V
TRIG
ECHO
GND
FRONT
LEFT
GND
ECHO
TRIG
VCC
GND
ECHO
TRIG
5V
D13
HC-SR04
HC-SR04
5V
TRIG
ECHO
GND
VCC
TRIG
ECHO
GND
RIGHT
GND
D5
BACK
GND
ECHO
TRIG
5V
HC-SR04
GND
ECHO
TRIG
VCC
D12
Buzzer
D11
GND
1
2
3
4
5
GND
+5V
X
Y
SW
A0
A1
JOYSTICK
D9
D7
D2
1
2
3
4
5
6
7
8
1-2EN
1A
1Y
GND1
GND2
2Y
2A
VCC2
VCC1
4A
4Y
GND3
GND4
3Y
3A
3-4EN
IC1
16
15
14
13
12
11
10
9
5V
D8
D4
D6
L293D
Motor
Motor
GND
GND
+12V
MotorDriver Schematic
Sheet: 1/1
5/22/2016 4:37 PM

- Car Picture:



4. Software Implementation

The software used for driving this project is written in the Arduino default IDE and allows us to get data from all four sensors, the joystick, drive the two motors and beep the buzzer at the same time very efficiently using threads.

The code is well written and easily readable and very easy to understand.

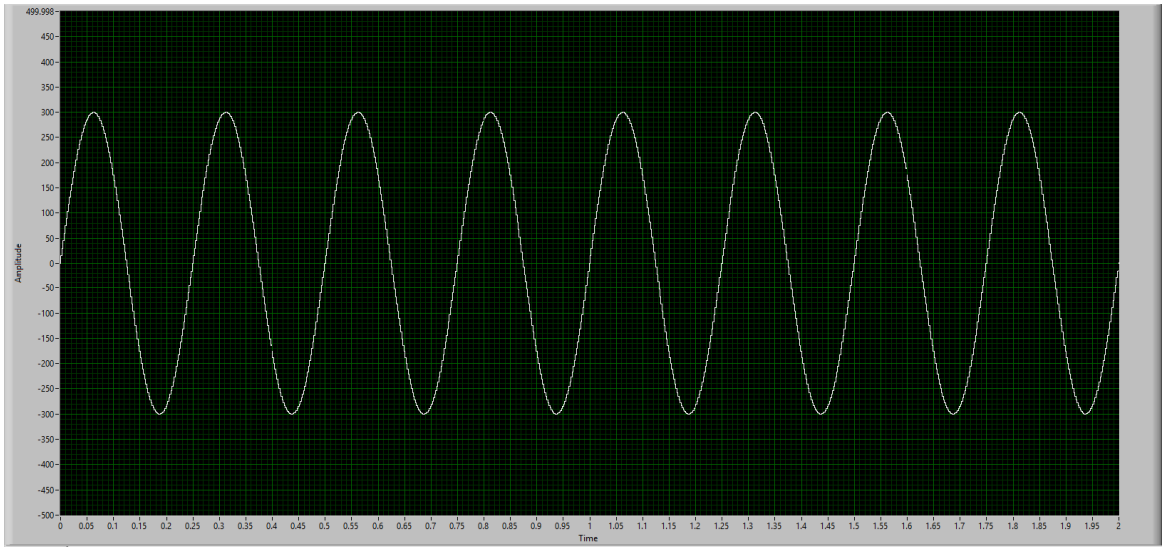
In order for us to maximize the efficiency of the project we are using external libraries for pinging the data from the sensor and beeping the buzzer at the same. This can only be done by using threads as there is no “pause” in a tone. The libraries we are using are the following:

- NewPing – we are using a self-modified version of NewPing that doesn’t use interrupt timers as it conflicts with the interrupt of the “tone” function.
- PhotoThreads – allows us to use threads. Photothreads are lightweight stackless threads designed for severely memory constrained systems and provide sequential flow of control without complex state machines or full multi-threading which cannot be done on the Arduino.

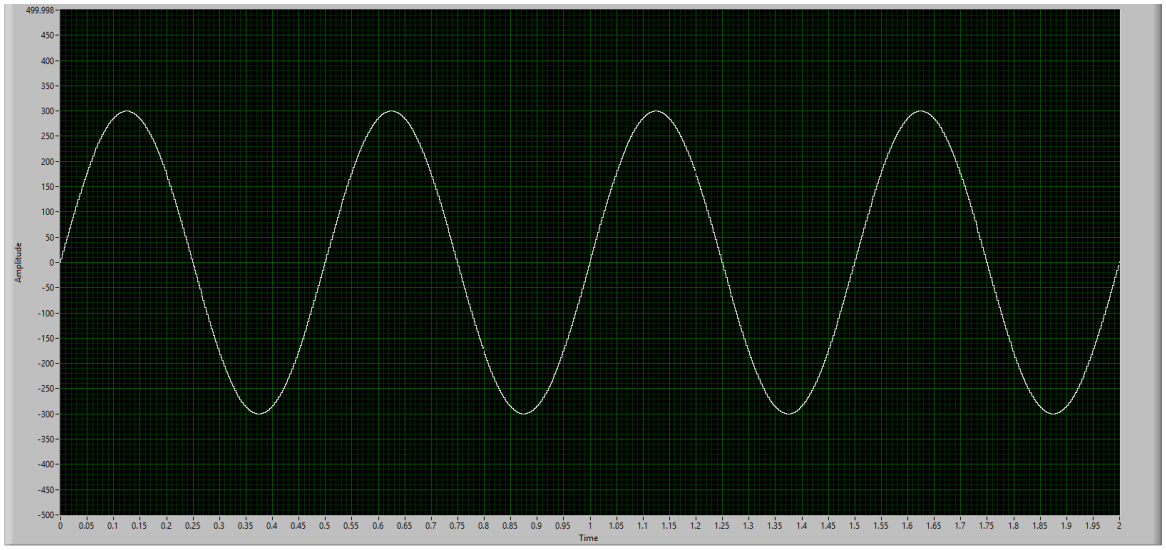
The given code also contains parts of a different, unfinished summer project I'm working on (*self-parking car*) that is disabled by a #define macro as of now.

The sensors beep the buzzer according to the lowest distance from any of the sensor. The project is programmed to use five different distance threshold for beeping and can be very easily modified by the #define macros that are already set. The levels of distance as of now are the following, together with their frequency graphs generated in LabVIEW:

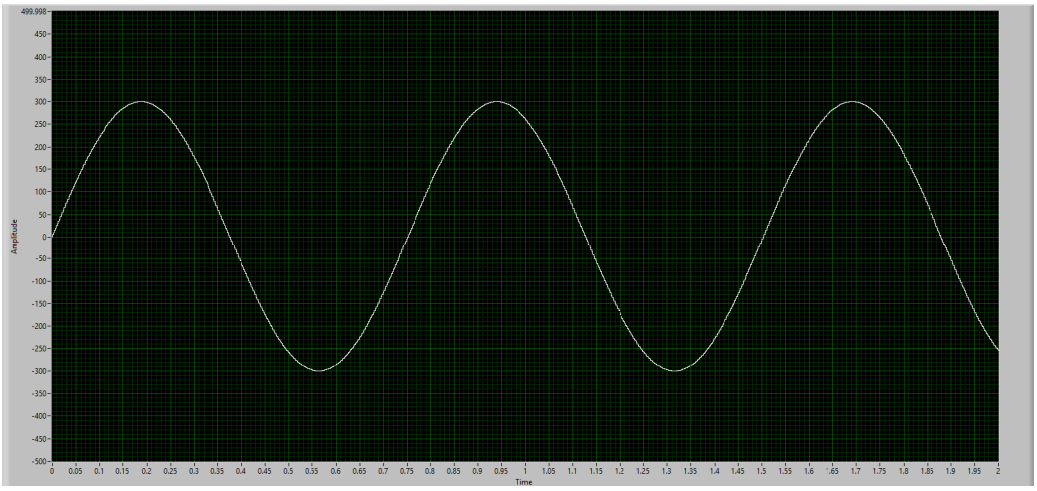
- 5 centimeters => 4 Hz (beeps every 0.25 seconds)



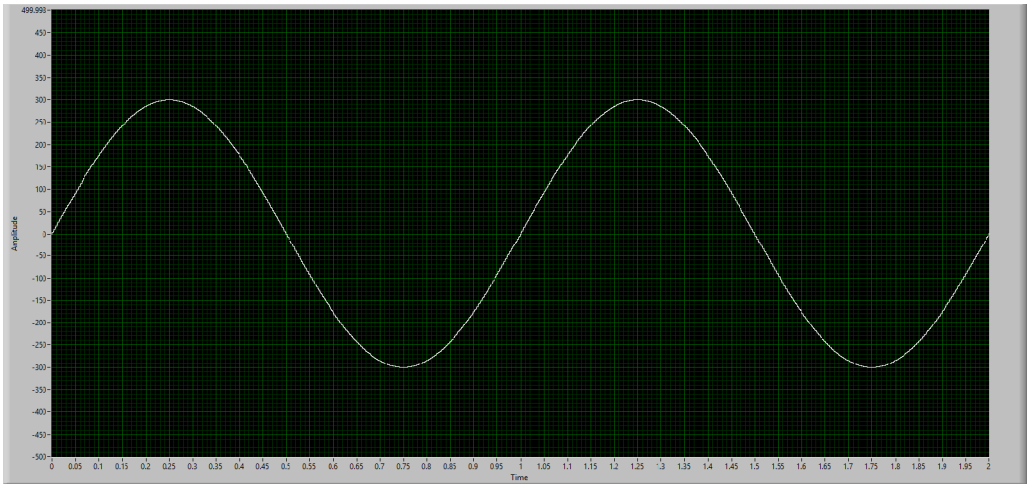
- 10 centimeters => 2 Hz (beeps every 0.5 seconds)



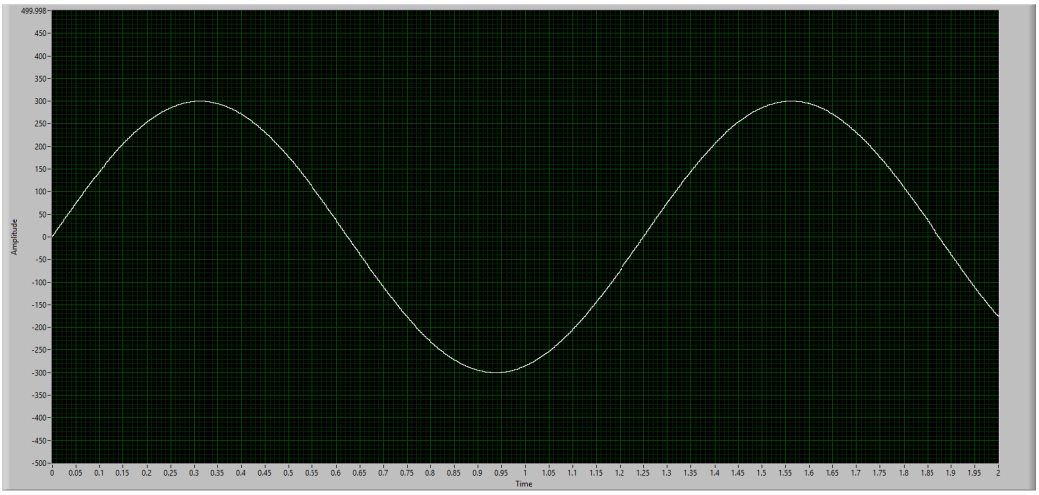
- 15 centimeters => 1.33 Hz (beeps every 0.75 seconds)



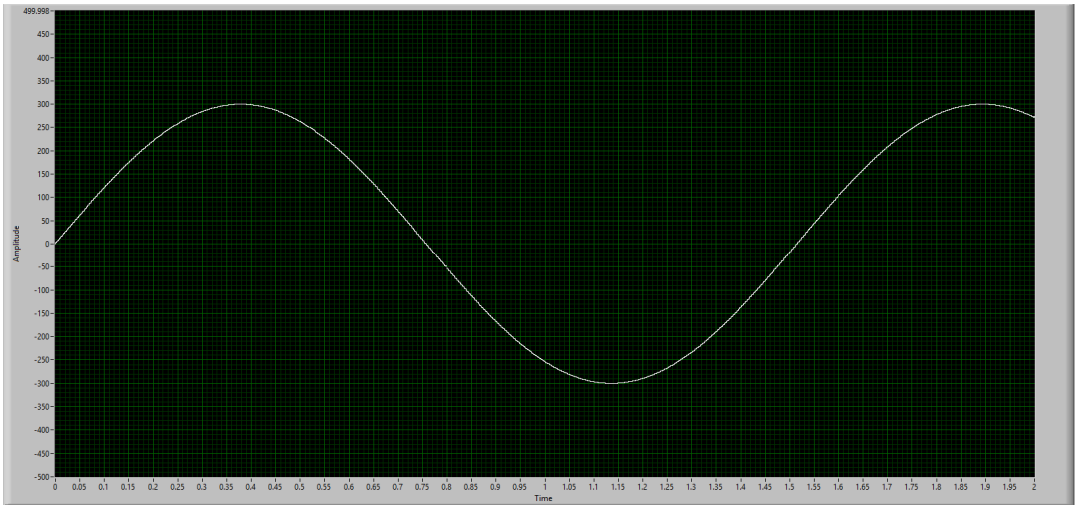
- 20 centimeters => 1 Hz (beeps once every second)



- 25 centimeters => 0.8 Hz (beeps every 1.25 seconds)



- 30 centimeters => 0.66 Hz (beeps every 1.5 seconds)



The interval and frequency of the beeper can be very easily changed by modifying the nBuzzerLevel, and nBuzzerInterval macros.