Carduino: CSE321 Term Project

Tucker J. Polomik

December 2, 2017

Abstract

Carduino is an autonomous, Arduino μ-controller-driven vehicle. The project meshes digital and analog circuitry in tandem with embedded & real-time programming theory to break a seemingly complex system down to smaller, simpler components. Special thanks to Dr. Bina Ramamurthy and her fleet of TAs for giving students the freedom to explore a project of their choosing, and being so eager to help whenever possible.

1 Problem Statement

The idea of self-directed vehicles can seem intimidating to some, especially to lay-people. This project (hopefully) serves to demonstrate that if one engineering student with minimal funding can create a self-driven car, then experienced engineers with millions in funding can create much more reliable, large-scale versions.

2 Project Motivation

Working on autonomous systems requires the unique skill-set of computer science knowledge for algorithmic rerouting, computer engineering for controller interfacing, and electrical engineering for circuitry design & power efficiency. I wanted to test my knowledge and see if I could mesh what I've learned from different courses at UB and make something cool. I feel too many students squander their time here and don't take advantage of the endless resources they have available to them as students in the form of funding, professorial mentoring and hands-on learning.

3 Design, Architecture, Components

When mapping out how to attack this project, I made a design plan which is roughly as follows:

- Purchase an RC car; strip it down to just the chassis
- Remove the motor, power supply, and front servos for prototyping

- Breadboard the components with the Arduino Uno controller
- Test code to run the different components separately
- Rewire the components together, test the code as a full system
- Place the system back onto the chassis of the car
- Provide external power to the system, pray that my soldering job was sufficient

Nitty gritty of the hardware for those interested:

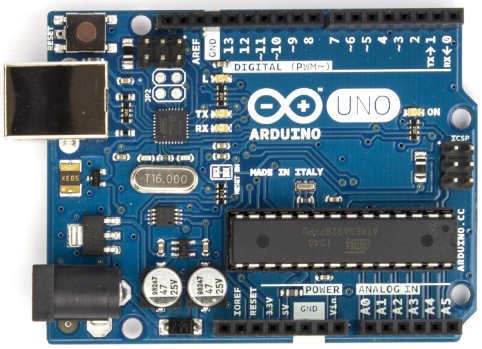

Figure 1: Arduino Uno Microcontroller [1]

The controller used was the Arduino Uno, the brain of which is an Atmega328P chip. The Uno board carries 32 KB of DRAM (only 0.5 KB of which is used for the boot-loader), 2KB of SRAM and 1KB or re-writable EEPROM. The board just barely has enough PWM pins for controlling the motor drivers, and supports standard 5V and 3.3V straight from the board itself. For higher voltage projects it also supports a V_{in} female jumper pin which regulates down to 5V so as not to over-volt the board.

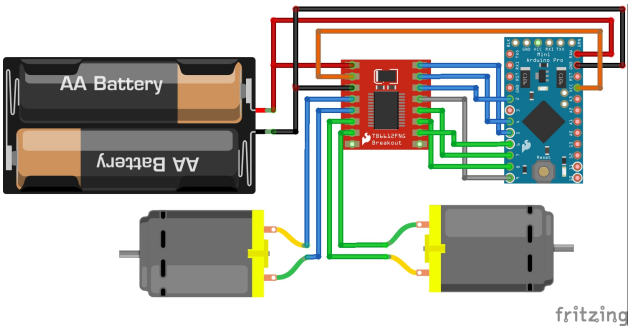

Figure 2: TB6612FNG Dual Motor Driver Wiring Diagram [2]

In tandem with the motor driver, I had an HCSR04 ultrasonic dual emitter/receiver which acted as the car's eyes. It operates in roughly the same way a bat does; emit ultrasonic waves (while moving), wait to get a response, do some math to figure out the distance in front of you the reflection occurred at, respond accordingly. I had initially wanted to work computer vision into this project with the use of infrared cameras since the feedback would be more accurate but time constraints said otherwise.

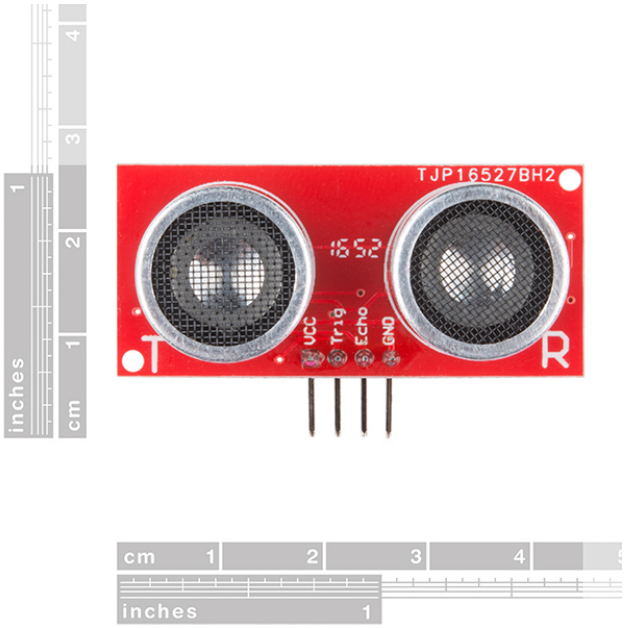

Figure 3: HCSR04 Ultrasonic Emitter/Receiver [3]

The wiring of the HCSR04 was trivial, 5V to VCC, GND to GND, analog out to trig and echo. The final hardware component was the servo motor which controlled the front axle of the vehicle for turning. The servo only has a max 60 degree revolution (30 degrees for each direction), so the car is not capable of making very tight corrections, which I needed to take into account when writing the code for when to turn in direction.

4 The Code

Normally when working with microcontrollers one must interface directly with the hardware to set-up I/O pins, serial ports, LEDs and so-forth. The Arduino boards however come with (optional) pre-built libraries which take care of this for you as it's rather tedious, especially if you don't really know what you're doing. For the sake of saving time, I used the 'Arduino.h' library. The language used is good old procedural ANSI C wrapped in a microcontroller library. Projects of this nature generally require a piece of code which is instantiated and executed just one time upon the powering up of the system, and then a series of cyclic loops which control logic flow until the power is turned off. Specifically, on the Arduino architecture, the entry-point(s) of the code are the setup and loop functions.

```
#include "Arduino.h"
#include <SparkFun_TB6612.h>
void setup() {
        // function initialization here
}
void loop () {
        // value checking here
}
```

Due to interfacing with multiple analog components, I needed more than just these two to keep my code readable. For example, setting up the Ultrasonic emitter/receiver takes quite a bit of code.

```
// prefer constants over preprocessor macros
const int offsetA = 1;
const int offsetB = 1;
const int trigPin = 9;
const int echoPin = 10;
long duration;
int distance;
int STBY = 6;

//Motor A
int PWMA = 3; //Speed control
int AIN1 = 9; //Direction
int AIN2 = 8; //Direction

//Motor B (servo)
int PWMB = 5; //Speed control
int BIN1 = 11; //Direction
int BIN2 = 12; //Direction

// instantiate the active pins on the component itself
void hcsr04()
{
pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
}

// entry point of the program where things are executed once
// serial port is opened to monitor output
void setup()
{
pinMode(STBY, OUTPUT);

pinMode(PWMA, OUTPUT);
```

```
pinMode(AIN1, OUTPUT);
pinMode(AIN2, OUTPUT);

pinMode(PWMB, OUTPUT);
pinMode(BIN1, OUTPUT);
pinMode(BIN2, OUTPUT);
hcsr04();
Serial.begin(9600); // Starts the serial communication
}

// parse the data received by the emitter/receiver to centimeter distance
void parse_hcsr04()
{
// Clears the trigPin
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
// Reads the echoPin, returns the sound wave travel time in microseconds
duration = pulseIn(echoPin, HIGH);
// Calculating the distance
distance= duration*0.034/2;
// Prints the distance on the Serial Monitor
Serial.print("Distance: ");
Serial.println(distance);
}

// continually call for data from the emitter/receiver
// parsing it as we go along for real-time distance measurements
void loop()
{
parse_hcsr04();
}
```

This is just a small example of what the code for this project looks like. One-time setup and preparation followed by cyclic execution of critical commands. For a direct look at the code please visit this GitHub repository. Thanks for reading. Questions or concerns: tuckerpo@buffalo.edu

References

[1] https://www.arduino.cc/en/uploads/Main/ArduinoUno_R3_Front.jpg

[2] https://cdn.sparkfun.com/assets/learn_tutorials/5/6/2/TB6612FNG_bb_1.jpg

[3] https://cdn.sparkfun.com//assets/parts/1/1/6/6/8/13959-02a.jpg