

NAIT
Edmonton, Alberta

RC Black Box TPS Report

Submitted to
Ross Taylor, CMPE 2960 CNT Instructor
Kelly Shepherd CMPE 2960 English Instructor

Submitted by
Michal Szmaj, CMPE 2960 Student

05/04/2017

Table of Contents

1 Requirements Overview	1
1.1 Parts Requirements	1
1.2 Software Requirements	1
1.2.1 Drivers	1
2 Software Overview	1
2.1 User Application	1
2.1.2 Files	1
2.1.3 Dependencies	2
2.2 Arduino	2
2.2.1 IDE	2
2.2.2 Files	2
2.3 Raspberry Pi	2
2.3.1 Access	3
2.3.2 Modifications	3
2.3.3 Files	3
3 Hardware Overview	3
3.1 Layout	3
3.2 Devices	4
3.2.1 Raspberry Pi Zero W	4
3.2.2 Arduino Nano	4
3.2.3 ADXL335	4
3.2.4 BMP180	4
3.2.5 GPS	5
4 Normal Operation — Test Sequence	5
5 Troubleshooting	6
5.1 Software	6
5.1.1 Raspberry Pi	6
5.1.2 Arduino	7
5.2 Hardware	7
6 Conclusion	9
7 Links	9

Figures and Tables

Figure 1. Fritzing wiring diagram for the RC Black Box device	3
Figure 2. Raspberry Pi Zero W	4
Figure 3. Arduino Nano	4
Figure 4. ADXL335 & BMP180 sensors	5
Figure 5. Adafruit Ultimate GPS Featherwing	5
Figure 6. Software Serial Connection	9

1 Requirements Overview

Ensure that the following requirements are met:

1.1 Parts Requirements

- RC Black Box device
- USB B mini male to USB A male cord*
- Micro USB male to USB A male cord*
- MicroSD Card Reader
- 5V - 2.5A power adapter to Micro USB

* USB A male end can be swapped with any cord that will fit your computer.

1.2 Software Requirements

- Arduino IDE*
- Xcode 8 (*macOS* only)**
- *Cocoapods* (*macOS* only)
- Terminal emulator (or *Microsoft Windows* alternative)***

* Any IDE that can compile and load an *Arduino*, i.e. *PlatformIO* can also work.

** In order to alter *iOS* software you will need *Xcode* in order to target and *iOS* device.

*** Instructions here will be using *Bash* (Unix environment)

1.2.1 Drivers

The current build of the RC Black Box is using an *Arduino Nano* clone and therefore needs a special driver installed in order to compile and load the device. The driver can be found in the 'Links' section.

2 Software Overview

2.1 User Application

The user application is made for *iOS* devices (i.e. iPhone, iPad, etc.) and is made using *Xcode* and *Cocoapods*. In order to work with the application you must be using *macOS* and have *Xcode* and *Cocoapods* installed (explained later).

2.1.2 Files

The RCBB *Xcode* project contains the following files and libraries — any not mentioned here are files automatically created by *Xcode*.

- *DebugViewController.swift* — controls the view which allows the tester to debug
- *DebugDataViewController.swift* — displays debugging information
- *UserDataViewController.swift* — view that displays the information to the user
- *MainMenuViewController.swift* — handles the main menu
- *ConnectViewController.swift* — allows the user to connect to the device
- *Constants.swift* — contains items like the static IP address and other constants
- *MSSocket.swift* — class which allows the application to connect to the device
- *MSJSONParser.swift* — class that parses the sentences returned from the device
- *MSRCBBData.swift* — struct which contains all the data from the device. Used for displaying data to user.
- *MSConnectDelegate.swift* — allows socket communication between views. Used to report Stream status, etc.
- *Extensions.swift* — contains extensions to classes for the entire project

2.1.3 Dependencies

In order to display a map to the user of the current location of the device the application is using the *Google Maps* API for *iOS*. In an Xcode project there are various ways to manage dependencies; for the RCBB, *Cocoapods* are used. *Cocoapods* is a dependency manager for *Swift/Obj-C*. Therefore anyone that is altering the RCBB Xcode project must have the latest version of *Cocoapods* installed. In order to install *Cocoapods* please follow this link and follow the directions: <https://cocoapods.org>.

2.2 Arduino

The Arduino software was written in C/C++ and used several standard libraries which will not be mentioned and several libraries that were written specifically for this project.

2.2.1 IDE

The required software for this project is either the official *Arduino* IDE which is *Java* based and can run on any machine or using *PlatformIO* which is an addon for the *Atom* text editor (also not platform specific). This project was written using *PlatformIO* and this is why the *Main* folder contains a *platformio.ini* file. The choice is up to the developer on which IDE to use, however, if the user chooses to use the official *Arduino* IDE then they will launch the application using the *Main.ino* file in the */Main/Main* directory rather than launching the *platformio.ini* file.

2.2.2 Files

The RCC Arduino project contains several directories and files:

- *Arduino* — contains *Arduino* IDE *.ino* and *PlatformIO .ini* files used for debugging individual components of the device. Directory list is as follows:
 - Accelerometer
 - BMP180
 - EEPROM — Not used within the actual project, but, could be useful for debugging purposes
 - GPS
- *Main* — The actual software that is loaded onto the *Arduino Nano*. This directory contains the *platformio.ini* file and the *Main* application directory
 - *Main* — All files below are contained in source and header files (*.cpp* and *.hpp*) unless otherwise stated
 - AccelerometerLibrary
 - BMP180Library
 - BMP180Debugging
 - Constants.hpp
 - DataConversionLibrary
 - eepromLibrary — Not used in the current build
 - gpsLibrary
 - Main.ino — used to launch the project using the official *Arduino* IDE
 - SerialCommunication

2.3 Raspberry Pi

The *Raspberry Pi Zero W* contains a modified version of the latest *Raspbian* image (latest version is called *Raspbian Jessie*); the modifications will be described below. The *Raspberry Pi* also contains *Python* files which are used to host the server and read data from the *Arduino Nano*.

2.3.1 Access

The device can be accessed by hooking it up to a monitor, a keyboard, and a mouse directly. While logged into the system navigating to `~/Documents/RCBB` will provide access to all the files that are running when the device is launched.

2.3.2 Modifications

The modifications done to the base *Raspbian Jessie* image are comprised of mostly networking changes. They allow the device to use Wi-Fi Direct in order to connect to the user application (it can also be connected to most other Wi-Fi enabled devices). Other modifications run the *Python* scripts at boot.

2.3.3 Files

The scripts that enable the device to function are located at `~/Documents/RCBB`. They are labelled as follows:

- *database.py* — handles all database related functionality
- *parse.py* — retrieves the data from the *Arduino*, parses the sentences, and stores it in the database
- *rcbb.sqlite* — the database that keeps record of the data coming from the *Arduino*
- *server.py* — hosts the server and relays information to the user application

3 Hardware Overview

This is the current layout of the *RC Black Box* hardware.

3.1 Layout

The wiring diagram can be seen below. However, for a more in depth look please follow the link to view the *Fritzing* wiring diagram in the 'Links' section. This will prove to be valuable when troubleshooting the hardware.

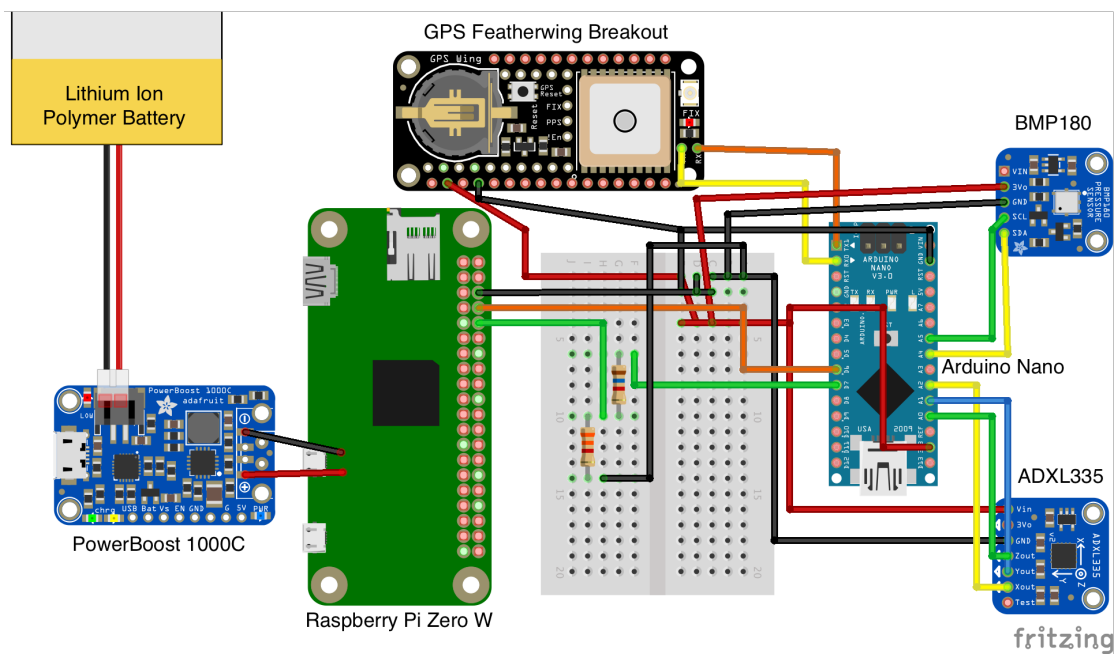


Figure 1. *Fritzing* wiring diagram for the RC Black Box device

3.2 Devices

Provided below is a brief overview of all the components used in the RC Black Box. For more detailed views of the pins and labels please view the *Fritzing* schematic linked in the 'Links' section.

3.2.1 Raspberry Pi Zero W

The *Raspberry Pi Zero W* runs *Raspbian Jessie* OS and has Wi-Fi and Bluetooth capabilities. It hosts a socket server allowing communication between itself and the user application and receives and parses the data provided by the *Arduino*. Figure 2 depicts a *Pi Zero W* with Ground depicted in black, 5V in red, Rx in yellow, Tx in orange.

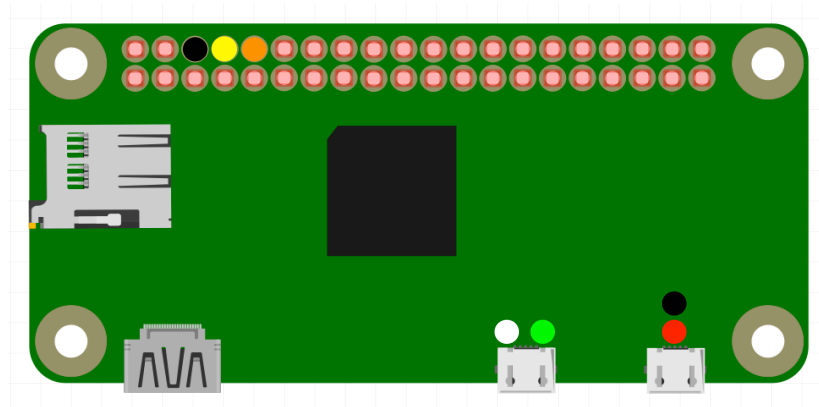


Figure 2. Raspberry Pi Zero W

3.2.2 Arduino Nano

The *Arduino Nano* runs custom written software in order to collect data from the connected peripheral devices and send the data to the *Pi* using *UART*.

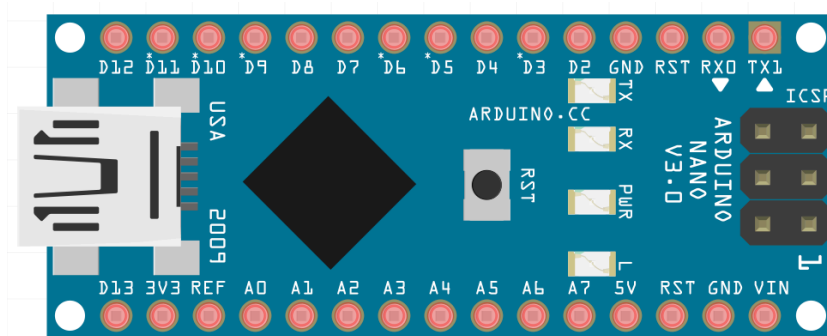


Figure 3. Arduino Nano

3.2.3 ADXL335

The ADXL335 is 3-axis accelerometer. Provides X, Y, Z axes. The device is connected directly to the *Arduino Nano*.

3.2.4 BMP180

The BMP180 is a sensor that provides temperature, pressure, and altitude. The device is connected directly to the *Arduino Uno*.

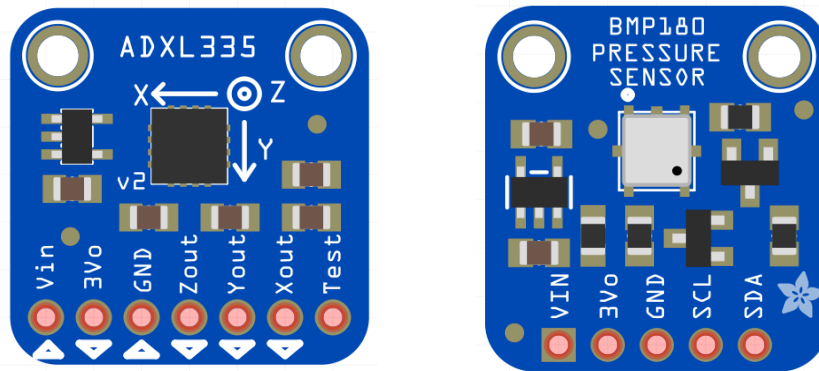


Figure 4. ADXL335 & BMP180 sensors

3.2.5 GPS

The *Adafruit Ultimate GPS Featherwing* breakout board is used to gather GPS information such as longitude and latitude, speed, and altitude. Its data is used to display the device's location using *Google Maps*. Figure 5 depicts the device with 3V labelled with a red circle, and ground as a black circle with white outline.

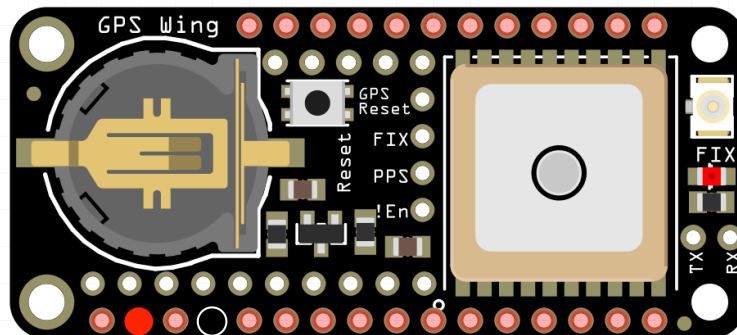


Figure 5. Adafruit Ultimate GPS Featherwing

4 Normal Operation — Test Sequence

The following outlines the testing procedure for the RC Black Box:

1. Charge the device using a Micro USB cable. Once charged, the *charged* light will turn green
2. Turn on the device using the switch
3. Wait approximately 30 seconds for the device to boot. To verify that it has booted try to connect to the 'RCBB' Wi-Fi signal on your *iOS* device.
 - 3.1. If, after approximately 45 seconds, 'RCBB' is still not available in the Wi-Fi list on your *iOS* device, then turn the device back off and on again
 - 3.2. If, after one or two hard restarts, the 'RCBB' network is still not available, then the *Raspberry Pi* might be having issues; see the 'Troubleshooting' section for possible solutions
4. Once connected to the 'RCBB' Wi-Fi signal you can launch the 'RCBB' application
5. In the application tap the 'Start' button.

6. You will be greeted with instructions on how to connect, but, you can skip them and click 'Connect'
- 6.1. If you are indeed connected to the 'RCBB' network, then the application will now show a new page with information streamed from the RC Black Box device
7. You are now ready to test the device. You might notice, if you are indoors, that the GPS will not work; you might have to go outdoors to get a stronger signal.

5 Troubleshooting

This section outlines the procedures for troubleshooting the RC Black Box device software and hardware.

5.1 Software

All of the software that was written for the device should work as intended, however, there is always a possibility of bugs in the code. This section will not describe in depth the code, but, instead will detail key points of interest.

5.1.1 Raspberry Pi

If there are any issues related to the socket server or connection between the *Arduino* and the *Pi*, this is where the software issues might reside. If any alterations to the source code are done or any extra functionality is needed, the code should be placed in `~/Documents/RCBB`.

Python Code

1. Functionality regarding reading the *Arduino* is in *parse.py*.
 - i. *parse.py* gets the latest sentence sent from the *Arduino*, checks if it is a valid sentence, parses it, constructs an *insert* query, and finally executes the query against the *rcbb.sqlite* database. There is not quick and easy way to debug this code; best way would be to run the script in a terminal emulator and uncommenting all of the print statements written in the source code. Using the print statements could be potentially helpful in determining if something is wrong.
2. Functionality regarding transferring data from the *Pi* to the user application is in *server.py*.
 - i. *server.py* hosts a socket server which the user application connects to. Depending on the command sent from the user application (commands are "L" for latest, "A" for all, "T" for today) the server constructs a query, executes it against *rcbb.sqlite*, converts the returned data into a *JSON* string, and sends the data back to the user application. Debugging the server is not as difficult as the parsing segment; all it entails is running the server and using the command 'telnet [ip address] [port]' in a terminal emulator. Once connected you can send the commands "L", "A", or "T" and you should receive a response. The server source code also contains various print statements that have been commented out, so, uncommenting them could provide some insight.


OS Setup

The operating system has had several modifications made to it in order to run accordingly.

1. First, the OS system runs a script located at `~/Documents/RCBB/boot` that runs at every boot which runs *parse.py* and *server.py* simultaneously.
2. Secondly, the wireless configuration has been changed.

5.1.2 Arduino

If there is an issue with the *Arduino* collecting data, then sometimes an easy fix is to reload the software onto the device as it might clear up some caching issues or bad memory. Follow these steps in order to do that:

1. Connect the *Arduino* to any computer that has an *Arduino* IDE
2. Open up either the *platformio.ini* or *Main.ino* file
3. Disconnect the GPS module from 3V on the breadboard (there is an issue when trying to load a program to the *Arduino* when a device is hooked up to Rx and Tx pins, unhooking GPS from 3V prevents this issue)
4. Load the program to the *Arduino* — using the official *Arduino* IDE, this can be done by connecting the *Arduino* to the computer and clicking the round arrow button: 
5. Once loaded the program should start running immediately and the issue should be fixed. If issues still persist, then there may be something wrong with the board itself and it might need replacing.

If there are any issues with any of the sensors reporting incorrect data, then the issue might be found here. To troubleshoot the software you must first figure out which sensor is providing the incorrect data. Once figured out, follow the steps outlined in ‘Hardware’, section 3, article 3 — ADXL335, BMP180, GPS. The steps given there describe how to load the software to troubleshoot individual sensors.

5.2 Hardware

Because there is a risk that the device might fall, there is a possibility of damage to the internal components. There are two issues that could possibly go wrong with the hardware: either a malfunctioning device (*Raspberry Pi*, *Arduino*) or loose wires. Follow these steps to determine which hardware issue it could be and how to fix the issue.

1. Take apart the device by spreading the two clips on the side and gently pulling the two halves away from each other
 - 1.1. This will reveal the components of the device and allow you to inspect the connections
2. Download the *Fritzing* schematics (provided in the ‘Links’ sections) and compare the wiring between the physical layout and the provided diagram
 - 2.1. If there are any loose connections on the breadboard, or if any of the pins have disconnected from any other boards, you may have to either place the wires back into the correct spots on the breadboard or re-solder the wires to the appropriate pins on the *Arduino*, *Raspberry Pi*, or any of the other sensors.
3. If there are no loose connections and all of the wiring is correct according to the *Fritzing* schematics, then there might be a point of failure in any of the internal devices. The following can be used as a guide in order to determine where the point of failure lies:
 1. ***Raspberry Pi***
 - 1.1. First turn on the disassembled device using the switch. This should turn on the *Raspberry Pi* and you should see a green/yellow light along with a bright blue light emitting from the *PowerBoost 1000C* device. If the blue light turns on, but, the green/yellow light on the *Raspberry Pi* does not, then the issue lies with the *Raspberry Pi*. If the bright blue light on the *PowerBoost 1000C* does not light up, then the issue lies there. If the wires are properly soldered


then the *PowerBoost* device may be broken due to impact or some other issues. This part will need to be replaced. If the issue lies with the *Raspberry Pi* continue to the next step.

- 1.2. To troubleshoot the *Raspberry Pi* you will need a 5V 2.5mA power adapter to the *Raspberry Pi* using micro USB. Connect the *Pi* to a monitor, keyboard, and mouse and see if it boots up. If, while connected to a power outlet using the aforementioned power adapter, no lights are present and nothing shows up on the monitor, the device is most likely damaged — if the issue was software based the device would at least turn on. If the *Pi* does not work, then remove the MicroSD card and replace the *Pi*.
- 1.3. If both of these are functional and the 'RCBB' network is still not available on your *iOS* device, then the problem may be a software issue. Please consult the 'Troubleshooting' section under 'Software'.

2. **Arduino Nano**

- 2.1. If the *Raspberry Pi* and *PowerBoost 1000C* device are fully functional, then the next point of failure is the *Arduino Nano*. However, if the problem is that the 'RCBB' network is not available, then please see the 'Software' section of 'Troubleshooting'.
- 2.2. Assuming that all of the pins are correctly connected to the peripheral devices, connect the *Arduino* to a computer using a micro USB B mini to USB A male cord. If, once connected, a red (or green if it's an official *Arduino*) light does not emit from the device this could mean that the device is damaged and a replacement might be necessary.
- 2.3. If the indicator is visible when powered on then this is most likely a software issue and a solution might be found in the 'Software' section above.

3. **ADXL335, BMP180, GPS**

- 3.1. If the readings for the ADXL335, BMP180, or the GPS are either non-existent or are not correct, then the issue most likely lies with the device itself. In order to test that the device is working correctly the device test script will have to be loaded onto the *Arduino*.
- 3.2. The test scripts can be found as follows:
 - ADXL335 — */Code/Arduino/Accelerometer/Accelerometer.ino*
 - BMP180 — *Code/Arduino/BMP180/BMP180/BMP180.ino*
 - GPS — */Code/Arduino/GPS/GPS.ino*
- 3.3. Locate the test script, open this script using whichever *Arduino* IDE you choose and load the script using the chosen IDE's load feature — using the official *Arduino* IDE, this can be done by connecting the *Arduino* to the computer and clicking the round arrow button: .
- 3.4. Once loaded, open the *Serial Monitor* — using the official *Arduino* IDE, this can be done by going to Tools -> Serial Monitor.
- 3.5. Here the device will display the data that is being read. If this data is incorrect, or the device does not turn on, then the device is damaged and needs to be replaced. If the device does turn on and the device data seems accurate then the data corruption most likely will be happening during the transmission between the *Arduino* and the *Raspberry Pi*; if this is the case then see the 'UART' section under 'Hardware'.

4. **Battery**

- 4.1. If the device does not turn on and the blue indicator light does not shine on when turned on then the issue might be the battery itself.

- 4.2. In order to troubleshoot make sure that the battery is charged. If after charging the device still does not turn on, it is time to replace the battery. If the battery is swollen do not attempt to charge the battery. Dispose of the battery immediately and accordingly.

3.5. Software Serial Connection

If the readings of the peripheral devices are correct but the data is still either not showing up or wrong in the user application, then the problem is most likely with the Software Serial connection between the the *Arduino* and the *Raspberry Pi*. To fix the issue with the connection, ensure that the following wiring is still intact:

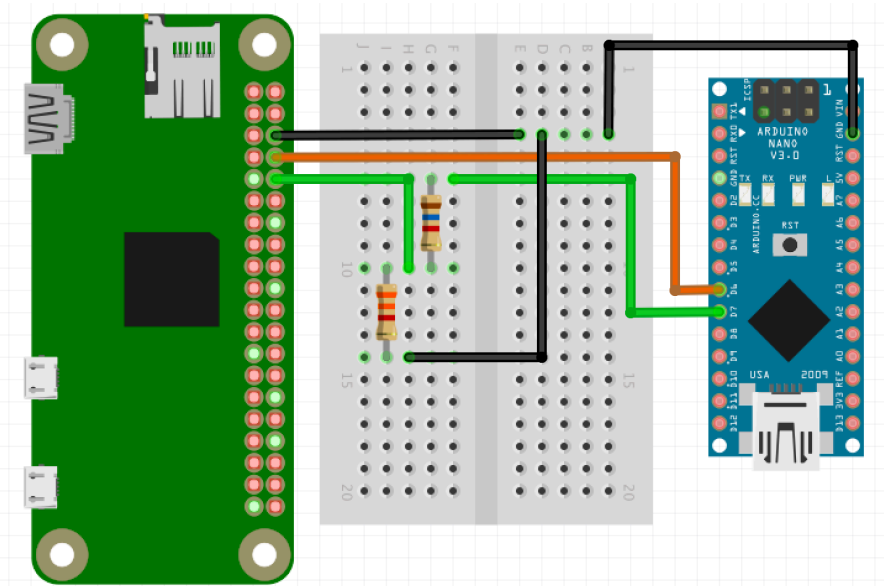


Figure 6. Software Serial Connection

This should be sufficient, however, if a closer look is needed please examine the *Fritzing* schematic found in the 'Links' section.

6 Conclusion

Most issues with the RC Black Box device should be resolved by following the steps outlined above. If, however, the device still does not work as it should more information can be provided. Contact michalszmaj@mac.com

7 Links

1. Driver for *Arduino Nano* clone (*macOS*):
 - i. https://gitlab.com/MSzmaj/capstonereferences/blob/master/Drivers/Arduino%20Nano/CH34x_Install_V1.3.pkg
2. *Fritzing* schematic:
 - i. <https://gitlab.com/MSzmaj/capstonereferences/blob/master/Schematics/BlackBox.fzz>