# Electrical Wiring

## Introduction

Wiring was one of the biggest headaches of this project, and a part that underwent a number of adjustments. Wiring for ultrasonic sensors, a Razor IMU, and a Turnigy receiver will be covered below. Note that the specifics for the physical wiring are left somewhat up to the builder, as each group used a different method for attaching everything to the BBBK. Our group went through several failing methods before falling back on Team Hindenburg's PCB, which we modified for our purposes. If the reader decides to go with this method, note that we used the 1st version of the PCB, which had soldering points for logic level converters on the board. We did not use these converters, but the open wires allowed us to modify the board by jumping the sensor inputs to the pins we needed them on as opposed to Team Hindenburg's pins. This PCB may or may not be included in their report.

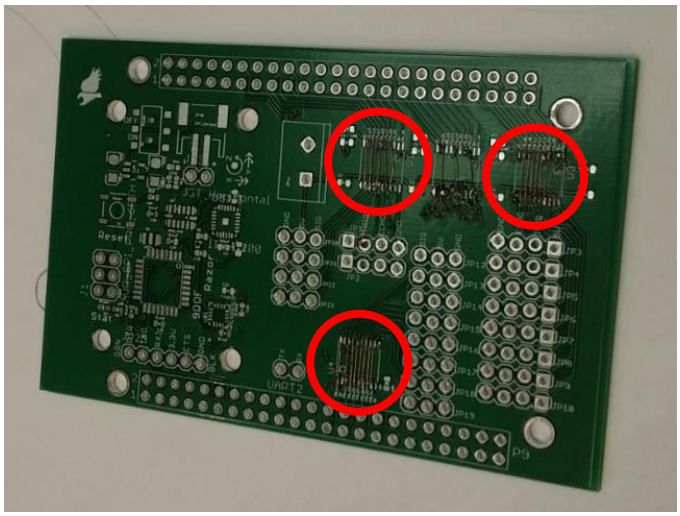

*Image 1: Hindenburg's PCB with Our Modifications*

# Sensor Wiring

## Introduction

Sensors integration was the most complicated part of the project, taking up the first half of the semester and going through several iterations as will be detailed below. The sensors included on the drone are 10 HC-SR04 ultrasonic sensors and 1 9DOF Razor IMU.

## HC-SR04 Sensor

The HC-SR04 is an ultrasonic sensor. It fires a burst of ultrasonic sound, which bounces off of objects and returns to the sensor. The delay from firing to return is measured to calculate the distance from the object. The sensor requires very specific timing to measure accurately, and it is difficult for a non-preemptive OS such as Arch Linux to measure accurately. Therefore, our team utilized one of the BBBK's Programmable Realtime Unit (PRU) CPUs to trigger and measure the sensors. The PRUs use ARM assembly code to operate, with only five nanoseconds per line of code, allowing very accurate measurements. The timing diagram for the HC-SR04 can be seen in Figure 1. One very important note about the HC-SR04 is that it is designed to be used with 5V tolerant systems, which the BBBK is not. Therefore, special care must be taken when wiring the sensors to the board, which will be discussed in further detail later in this report.


*Image 2: HC-SR04 Ultrasonic Sensor*

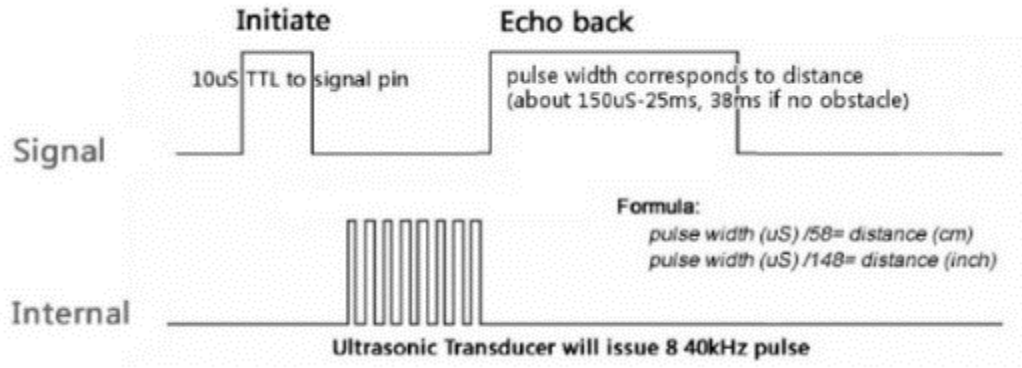


*Figure 1: HC-SR04 Timing Diagram*

## Initial Wiring Plan

Before testing began, we laid out where we thought we would like to wire our sensors to the BBBK, as in Figure 2.

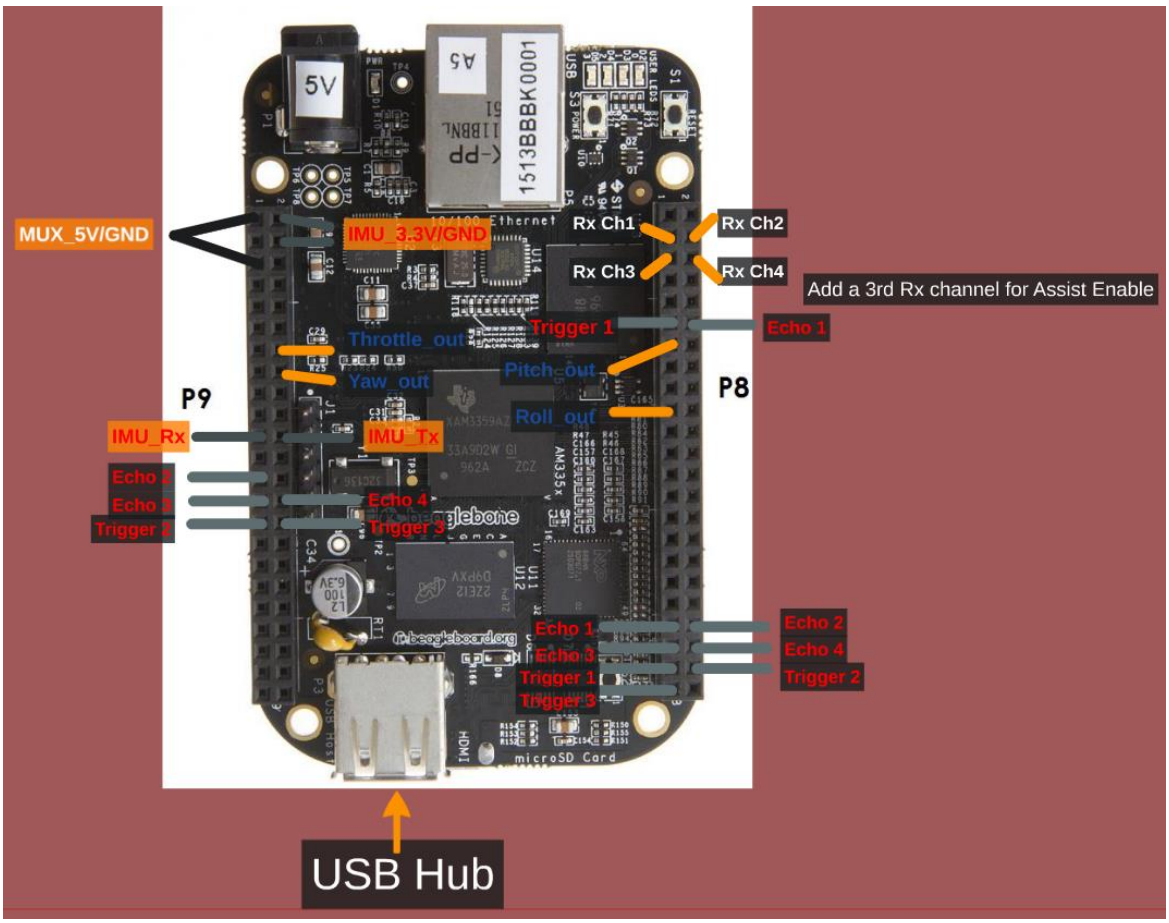


*Figure 2: Initial Wiring*

We initially planned to multiplex our echoes so as to reuse PRU pins, as a large portion of the pins were in use by the BBBK HDMI port. However, we quickly realized that this layout would cause unnecessary

delays in measurement, since not all ultrasonics could be measured at once. This, coupled our learning how to disable the HDMI, led to a change in our wiring layout before we reached the testing phase.

## Special Note on Triggering

Each team struggled with the proper way to trigger the ultrasonic sensors. Multiple methods were tried, with different levels of success. Methods that were NOT successful are as follows. DO NOT try to trigger all sensors from one pin. Triggering the sensor draws approximately 2mA from the BBBK, and the BBBK pins are rated for 4mA or 6mA, depending on the pin. Firing all sensors from one pin will destroy the pin. DO NOT try to trigger the sensors too fast without a current limiting resistor. Another team had trouble with this, so I do not have the exact numbers or rate of fire, however our theory is that since the HC-SR04 is technically rated for a 5V trigger, and we are only providing 3.3V, the current draw is too high to sustain a fast fire rate. Our method of triggering, which works, is detailed later in this section.

## Final Wiring Layout

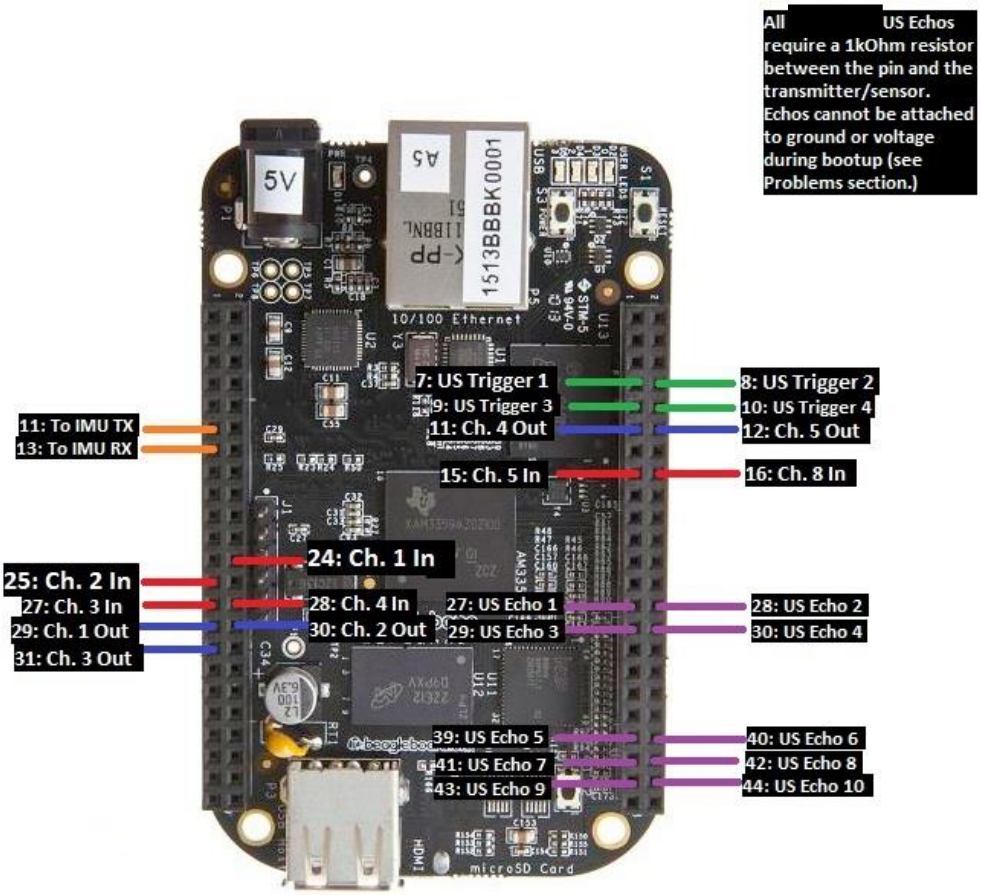


*Figure 3: Final BBBK Wiring*

Figure 3 is our final wiring layout. As can be seen, each sensor has a dedicated echo pin on the PRU. We have 4 Trigger pins, wired to standard GPIOs, which can also be controlled via PRU, albeit at a much slower rate. However, trigger signals are not required to be accurate, only greater than 10 microseconds, so there is no problem using GPIOs. Triggers 1 and 2 each trigger 3 sensors, while Triggers 3 and 4 trigger 2 each. This is to avoid the triggering problems described above. The pins used have been checked to verify that they can supply the 6mA needed to trigger the sensors. Note that while it does not matter which sensors are wired where for the echo pins, we used the wiring layout in Table 1 for the triggers. We chose this layout so that in the event that we had sensor interference from firing all sensors at once, we could fire one trigger at a time and have sensors facing different directions so as to have the least amount of interference as possible. For example, if Trigger 1 were fired, the north, southwest and southeast triggers are facing away from each other, and as such will not interfere.

| Trigger Pin | Sensor Orientation |
|---|---|
| 1 | North, SW, SE |
| 2 | South, NW, NE |
| 3 | East, West |
| 4 | Top, Bottom |

*Table 1: Triggering Layout*

## Wiring the Echoes

Wiring the echo pins properly is very important for not damaging the BBBK. The HC-SR04 is designed to be used with 5V tolerant boards. Since the BBBK pins are 3.3V tolerant, wiring a sensor directly to the board is a great way to fry it. Several methods were tested by different teams when wiring the sensors. Logic level converters were tested by a different team and ultimately thrown out, as the delay made the measurement very inaccurate. They switched to voltage dividers, but did not have time to test them extensively, so we cannot endorse that method either. Our team's initial solution that we carried through to the final design was using a simple 1kOhm current limiting resistor in series with the pin, as seen in Figure 4. We never had any problems with this layout as long as the system is powered up correctly. System powering will be discussed later in this report.

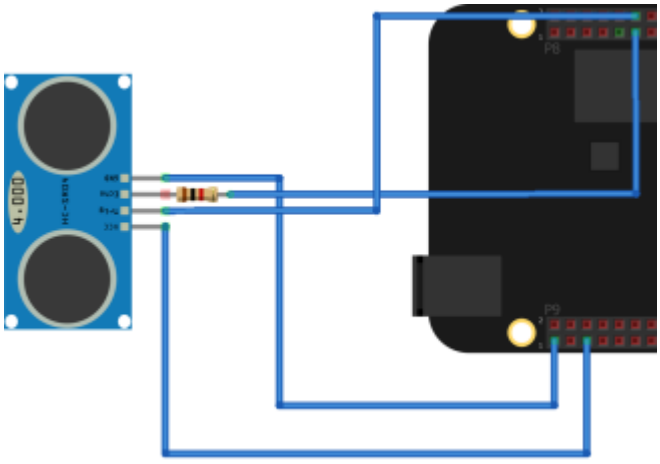

*Figure 4: HC-SR04 Echo Wiring*

# Transmitter/Receiver Wiring

## Introduction

While sensors integration is necessary to get distance and movement data to the BBBK, setting up the transmitter and receiver is equally important to allow the BBBK to read the user's inputs and modify them as necessary with the sensor data. On a standard drone, the user wires the receiver directly to the KK2 flight board. For our project however, we wired the receiver to the BBBK, which reads in the users inputs, modifies them as necessary given the sensor data, and outputs the modified signals to the KK2.

## Turnigy 9X

For this project, we used the Turnigy 9X 9 Channel transmitter/receiver combo. The receiver receives PPM signals from the transmitter, which it converts to PWM signals similar to what is seen in Figure 5. These signals are typically interpreted by the KK2, but we set up the BBBK's PRU 0 to measure the length of time each signal is high and output this number to shared ram, which can then be read and modified by the main program. The wiring layout can be seen in image 3 above.

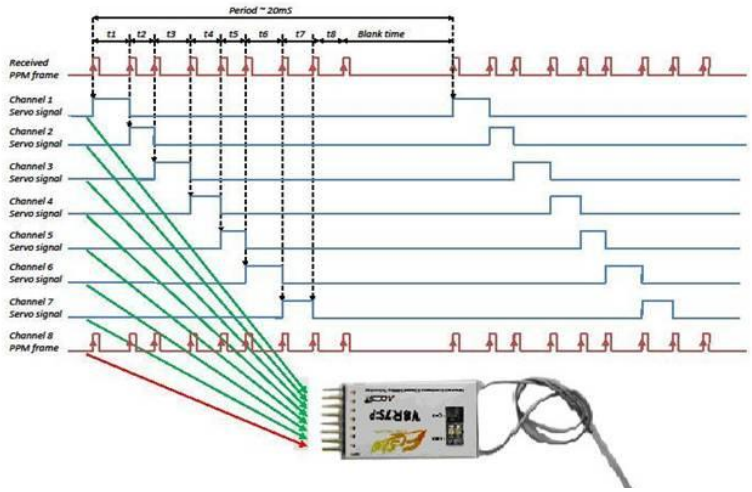


*Figure 5: PPM to PWM Reciever*

## Receiver Input Wiring

Unlike the ultrasonic sensors, the transmitter is 3.3V tolerant and can be attached directly to the BBBK. It must be powered by a 5V line, as well as grounded. System powering is covered later in this report.

## BBBK Output Wiring

PWM output pins are then wired to the KK2 Flight Board. Initially, we wired out KK2 directly to the BBBK. This may have been a fatal mistake however, as the KK2 is expecting a 5V input. Ultimately, we added a logic level converter in between the BBBK and the KK2 as seen in Figure 6. This worked out well in limiting the current draw on the BBBK.

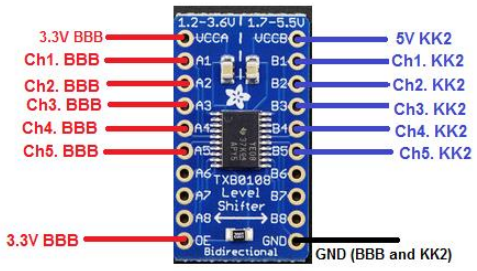


*Figure 6: Logic Level Converter Wiring*

# Control Coding

## Introduction

Control coding utilizes the ultrasonic sensors, the IMU and the receiver inputs to modify user input and send updated signals from the output pins to the KK2 control board. The objective of control coding is to use the IMU to prevent drift in from drafts, and use the ultrasonic sensors to prevent crashes. Our team only completed part of this goal, implementing control coding for 4 sensors, north, south, east and west. We are currently also reading data from the IMU, but not using it to modify the outputs. The code also uses the channel 8 input signal to switch between modified and unmodified signals. All program files can

be found in the Github linked in the Downloads section of our website. This Github also has a readme specifying the proper methods for getting the code up in running on the BBBK. Note that code has only been tested for Arch Linux ARM.

## PWM I/O Programming

The receiver inputs and BBBK outputs to the KK2 are both PWM signals as discussed in the previous section. These signals are nearly identical to the echo inputs from the ultrasonic sensors, and therefore are decoded and encoded on PRU 0. The PRU code is PWMInOut.p. The program uses a timer to measure how long each signal is high and saves this data into a shared memory that can be accessed by both PRUs and the primary CPU and main control program. The memory layout can be seen in Table 2. The modified memory locations contain the PWM signals that have been modified by control code. The software mux is also located in this program. When the program reads the value of channel 8, it checks to see if it is above or below the median time. If the signal is high for over the median time, the switch is high, assist is on, and the program will generate PWM output signals from the modified memory locations. Otherwise, the output signals are generated from the same memory locations that the input signals were saved to, essentially just passing them through without modifying them. If the main program crashes for any reason, the user can flip the assist switch to take direct control of the drone. This solution does not solve complete BBBK failure however, and should therefore be replaced with a hardware solution. More details on code functionality can be found in the annotations in the files themselves.

## Ultrasonic Sensor Coding

Ultrasonic sensor triggering and measuring is controlled entirely by PRU 1 on the BBBK. The PRU code is loaded via PruInit.py and sensor measurements taken by the PRU are read from memory in US.py. The PRU code itself is in Ultrasonic.p. There is a delay of .03 seconds before each trigger to allow the sensors to reset and the previous sound bursts to die away, resulting in a frame rate of approximately 33 measurements a second per sensor. This frame rate can be easily increased by decreasing the delay before the trigger, however we felt that this frame rate was more than enough for our goals while limiting the amount of sound interference and avoiding the triggering problem discussed above. The PRU stores the values of the ultrasonic measurements to a shared memory that can be accessed by both PRUs and the primary CPU and code. The memory layout can be seen Table 2. More details on code functionality can be found in the annotations in the files themselves.

| **Signal** | **Offset from Shared Ram Start (0x10000)** |
|---|---|
| Ch. 1 In | 0x10200 |
| Ch. 2 In | 0x10210 |
| Ch. 3 In | 0x10220 |
| Ch. 4 In | 0x10230 |
| Ch. 5 In | 0x10240 |
| Ch. 6 In | 0x10250 |
| Modified Ch 1. Out | 0x10200 |
| Modified Ch 2. Out | 0x10200 |
| Modified Ch 3. Out | 0x10200 |
| Modified Ch 4. Out | 0x10200 |
| Modified Ch 5. Out | 0x10200 |
| Echo 1 | 0x10000 |
| Echo 2 | 0x10010 |
| Echo 3 | 0x10020 |
| Echo 4 | 0x10030 |
| Echo 5 | 0x10040 |
| Echo 6 | 0x10050 |
| Echo 7 | 0x10060 |
| Echo 8 | 0x10070 |
| Echo 9 | 0x10080 |
| Echo 10 | 0x10090 |

*Table 2: Shared Memory Mapping*

### Modifying the Output Signals

Control coding can be found in the PWM.py file. Our control coding is relatively simple, since we did not have a lot of time to work on it. We are currently only using the North, South, East and West ultrasonic sensors and a simple proportional algorithm to modify code. PWM.py contains annotations on the specifics of the control code.

## System Powering and Boot Up

### Practical Problems

Powering the BBBK and sensors systems can be tricky for a few reasons. Pins 27-46 are SYSBOOT pins, which means they are integral during the board boot process. If there is any signal over these pins, the BBBK will probably not boot properly. Also, supplying current to a pin before the BBBK is booted will most likely result in damage and destruction of the board. These problems make powering order very important on the drone. No team found one foolproof solution, however, we found a fairly reliable system.

### Switches

Our team used 3 switches to ensure proper boot order. Note in the figure below that switch 3 disconnects the sensors and receiver entirely from the system. This is because even a ground connection can mess with the SYSBOOT pins. Although the KK2 is not connected to SYSBOOT pins, it too requires a switch. If the KK2 does not receive a PWM signal at its inputs upon power up, it will enter a failed state and must be turned off and on again. Integrating a switch is much easier than manually disconnecting the power wire each time this must be done. Given more time, we would like to replace the switches with MOSFETS so they can be controlled from software.

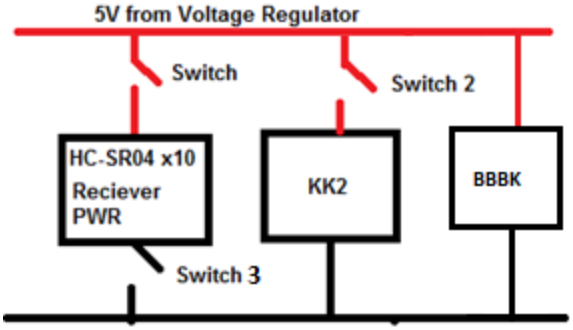


*Figure 7: Switch Wiring*

### Booting from MicroSD

An important aspect of the boot up sequence is boot order. Normally, the system first attempts to boot from eMMC, then from microSD card. However, if there is an error with the SYSBOOT pins, this boot order can be corrupted and the BBBK may hang. We therefore found it useful to boot only from microSD. This is accomplished by installing Arch Linux onto a microSD card and wiping the eMMC memory via the line

```
sudo dd if=/dev/zero of=/dev/mmcblk1 bs=1024 count=1024
```

This way, no matter what the boot sequence is, the BBBK will always fall back to the microSD. We found that this combined with the switch system provided reliable booting. Team Hindenburg claims to have found a more reliable method, so it may be useful to read their report on the subject.

# IMU Integration

## Introduction

During the planning phase of the project, we wanted a method of detecting and resisting drafts that could be caused by a fire inside a building. To do this, we would need a way to measure drift. The class settled on the 9DOF Razor IMU from Sparkfun for its ease of use.

## IMU Wiring

The IMU is one of the easiest components to wire to the BBBK. It is already 3.3V tolerant, so no special wiring is needed to integrate it. Simply power the board from the 3.3V rail, ground it, and attach the Tx and Rx pins as specified in Figure 3 above.

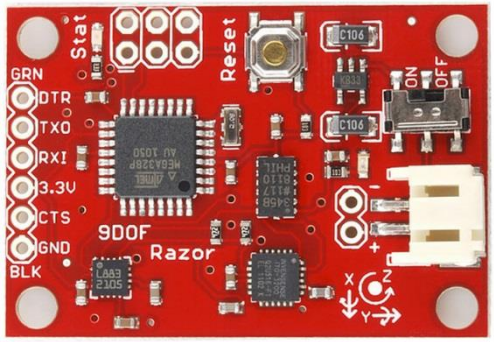


*Image 3: 9DOF Razor IMU*

## IMU Control Coding

Unfortunately, the IMU was never actually used for controls, as we had planned. Time was very limited and this was a low priority on our list. In terms of base functionality, it is unnecessary; however, it could add some very interesting capabilities to the drone with the included gyroscope, magnetometer, and accelerometer.

Although it wasn't used, if wired IMU readings will be outputted to the GUI when the Main.py program in the Github is run. These signals are read in the imu.py file.

# Wi-Fi Configuration

NOTICE: This guide assumes the use of a TP-LINK Wifi Adapter, model no. TL-WN7200ND. If you do not have the exact model, this exact same process outlined below should work as long as your adapter has a Ralink chipset (prefix = RT) and has access point capability. It also assumes **Arch Linux ARM** is the distro of Linux in use. These instructions, and indeed all of the instructions on the Github are only tested for Arch.

## Before Wireless Capability

To connect your BBBK to the internet (so that you can download packages), follow the steps below:

1) Plug the BBBK into an internet capable router via Ethernet cable
2) Plug the 5V power adapter into the BBBK
3) Check the list of devices connected to your network. If you're using a NETGEAR router, this can be done by visiting routerlogin.net. On this list, you'll see a device labeled as "ALARM." This is the BBBK! It should have an IP address next to it that will be something like 198.162...
4) In putty (or something like it) use the IP address (or "alarm" should work to, in the same field) to establish a connection.
5) `login as: root`
6) `password for root: root`
7) Good to go!

User Guide by TP-Link:

http://www.tp-link.com/resources/document/TL-WN7200ND_V1_User_Guide_1910010859.pdf

## Driver Details

When you first connect the wifi adapter via the micro-USB cable, the LED will most likely begin to blink slowly. If this occurs, it means that the driver has been installed correctly and the device is recognized. If this happens, you're good to go! The driver required for this adapter is native to the Arch distribution (distro) our team is using, so this *should* be plug and play. If you're using our setup, it will be.


*Image 4: TP-Link Wi-Fi Adapter*

To check driver status, typing

```
# dmesg | grep usbcore
```

Should return something like "`usbcore: registered new interface driver rtl8187`" if it worked correctly.

OPTIONAL: Using the Adapter as a Router for an Internet Connection when in SSH

This mode may be helpful at first when you don't have to worry about portability. If you don't like working while being connected to a router, skip this section. The usefulness of this method as opposed to just using the AP (Access Point) comes from the fact that you don't have to enable internet sharing amongst your ports.

With this setup, you plug the BBBK's Ethernet port into a router, and the adapter into the USB port of the BBBK. This will enable you to connect to your BBBK wirelessly through the preexisting wifi network you have setup for that router (using the same SSID and password).

This configuration was set up using steps entirely from the ArchWiki at this link:

https://wiki.archlinux.org/index.php/Wireless_network_configuration#Manual_setup

Get required packages:

```
# pacman –S iw wireless_tools wpa_supplicant
```

Before continuing to the next step, check the name of the wifi interface. By default, it will most likely be wlan0, which is used in the commands below. To check this interface, type the command below and read the output. Interface names *usually* start with a 'w'.

```
# ip link
```

Set the adapter to ad-hoc mode so the adapter can share a peer-to-peer connection with your router. Then, type the following commands to set the interface up and match it to your personal network: **Notice the subtle difference between the '`1(one)`' and the '`l(L)`', the top portion of the one is bent downward.

```
# iw dev wlan0 set type ibss
```

```
# ip link set wlan0 up
```

```
# wpa_supplicant –D nl80211,wext –I wlan0 –c <(wpa_passphrase "Your_SSID" "Your_key") –B
```

"`–B`" allows the process to run in the background so you can do other things while connected to the internet.

```
# dhcpcd wlan0
```

The LED on the adapter should now be blinking quickly to indicate data transmission/reception.

```
# iw dev wlan0 link
```

The output of this will show you whether the link is actually up or not.

## How to Setup a Customizable Access Point

Get required packages:

```
# pacman –S hostapd dnsmasq iptables brctl dhclient
```

A detailed guide for setup and configuration, as well as customization options, from installation, can all be found here:

https://github.com/oblique/create_ap

Our method utilized the WPA + WPA2 passphrase:

```
# create_ap wlan0 eth0 MyAccessPoint MyPassPhrase
```

"`eth0`" is there to set where the `wlan0` interface will get internet connectivity, if one is available. Typing the line above will run the script. The script will not run again when you restart unless you type that line again. In order to set it to start at boot, type the following:

```
# systemctl enable create_ap
```

To configure what AP is being created by the script running on boot, using a program such as WinSCP, edit the file “create_ap.service” found in the bin under ~/root and change the “ExecStart” line. The command already there should look similar to the examples on the github page.

Your AP will only allow you to connect to the BBBK wirelessly and independent of any internet connection or router. To connect your AP to the internet, the BBBK will need to be connected via the Ethernet port to an internet-connected device with internet sharing enabled (laptop). You do not need an internet connection to SSH to the BBBK.

To enable Ethernet port internet sharing on a Windows computer, simply right-click the wifi icon in the bottom right corner and open the “Network and Sharing Center.” On the left side of that window, select to “Change adapter settings.” You should be in the Network Connections window now, under Control Panel\Network and Internet\Network Connections. Next, right-click on the adapter built into your laptop with the wifi connection on it and open “Properties.” Navigate to the “Sharing” tab and check the box next to enable Ethernet connection sharing. Your AP should now have internet access while your Ethernet cable is connected and you will be free to download packages.