SPACE COAST UNMANNED

Cerberus User Manual

Table of Contents

Table of Figures

List of Tables

1 Overview

The Space Coast Unmanned (SCU) Cerberus is a multispectral detection system designed for use in the Force Protection Kit challenge. SCU is pleased to offer a fully compliant solution that covers seismic, acoustic, magnetic, infrared, radiofrequency(RF), and visible light anomaly detection in an extremely low cost package. Telemetry including battery voltage and tamper detection is provided via LoRa long range radio units with an advertised range of 1.25 miles (2km) line of sight. The transmission is protected by AES128 encryption to prevent interception of data. The radio units can be configured to transmit periodically, or only upon anomaly detection to prevent detection of the sensor system. All hardware used inside the unit is openly available online, and open source or GPL software was used to minimize unique software. Adaptations specific to the unit such as wiring and code changes were unavoidable but kept to a minimum to allow easy reproduction or future functionality upgrades. The unit draws only 4 Watts of power, allowing long runtimes. The current enclosure is rated to IP55 ingress protection standards. The unit is compact and lightweight with a carrying handle, allowing easy transport by a single individual. Deployment of the sensor can be completed by one person in less than one minute for maximum versatility. Addresses from 0-255 are available, meaning a single receiving station can uniquely identify multiple sensors simultaneously.

2 Components

2.1 Arduino Mega

An Arduino Mega serves as the controller for the radio transceivers in the Cerberus and the receiver station. The mega was chosen due over a normal Arduino to its performance gains and larger memory. The current utilization is roughly 23% of the available memory, leaving plenty of room for future expansion.

2.2 Raspberry Pi

A Raspberry Pi was chosen to monitor the Bosch 9 degree of freedom sensor as well as the RTL-SDR RF sensor. This allows extremely powerful computing to be done in a compact package. As delivered, the scripts utilize around 30% of the Raspberry Pi's CPU. In the future, additional functionality can be implemented with plenty of overhead.

2.3 Acoustic Sensor

The acoustic sensor is a MAX9814 electret microphone with automatic gain control up to 60dB. This allows for extremely dynamic sound ranges.

2.4 Visible Light Sensor

The VC0706 weatherproof camera communicates via serial. It has motion detection built in, which allows easy integration. The camera is infrared sensitive, giving it greater performance in low light conditions. There is also a low light level sensor, which triggers infrared LEDs once ambient lighting becomes too dark for the camera alone.

2.5 Infrared Sensor

There is a Passive Infrared (PIR) sensor installed on the motion camera. It functions by monitoring the infrared spectrum in front of the sensor. When an object that emits infrared (such as a human) moves past the sensor, it will trigger. These sensors are commonly used in household motion detectors and flood lights. This extends the low light capabilities of the Cerberus beyond the range of the infrared LEDs on the VC0706 camera.

2.6 Magnetic/Seismic Sensor

The magnetic and seismic detection is accomplished via a Bosch BNO055 9 degree of freedom orientation sensor. This sensor combines a 3 dimensional accelerometer, magnetometer, and gyroscope. Initialization of the device includes reading the sensor orientation and magnetic headings. A threshold is set via software for magnetic and seismic activity. Because the magnetometer is very sensitive, rotations of the sensor can produce magnetic disturbance alerts as well as seismic alerts. The sensor has also been test triggered using standard household magnets near the sensor.

2.7 Radiofrequency (RF) Sensor

The RF sensor is an RTL-SDR unit attached to the Raspberry Pi. This unit is capable of reading frequencies between 24 MHz and 1850MHz. The unit as delivered is monitoring FRS channels 1-7 and 15-22. Channels 8-14 are at a higher frequency and were not included in this demonstration. This unit is compatible with gnuradio, or other software defined radio suites. The unit is currently using the Osmocom RTL-SDR open source library's built in command line functionality to monitor FRS radio channels. By changing the parameters such as center frequency, bandwidth, gain, and integration time, it can easily be modified to monitor any frequency the RTL-SDR is capable of reading.

3 Telemetry

3.1 Temperature Sensor

A Texas Instruments LM35 sensor is providing temperature readings via analog output. It is advertised as having an accuracy of ±0.5 °C.

3.2 Battery Monitor

The battery monitor is implemented using a simple voltage divider circuit and analog output to the Arduino Mega. The accuracy is around 0.1 Volt.

3.3 Radio Transceiver

The radio transceiver is a HopeRF RFM95 LoRa unit. The power output is adjustable from 5-23 dBm, and is currently at the default of 13 dBm. With directional antennas and tuning, range is advertised up to 2km (1.25 miles) line of sight.

3.4 AES Encryption

To prevent third party monitoring or exploitation of the Cerberus, AES Encryption is implemented via an open source AES library, tailored specifically for the Arduino processors. Hardware initialized random numbers are added to the message for proper "salting" to prevent repetition of cipher data. This means that even if the same data is sent over and over, a third party without the key cannot analyze the message and know that the results are the same each time.

4 Physical Characteristics

4.1 Size

The outer dimensions of the protective case are: 16"x20.5"x7.5" (LxWxH).

4.2 Weight

The Cerberus weight as delivered is 15.5 pounds.

5 Power

The Cerberus draws roughly 4 Watts continuously. This low level of power draw facilitates easy customizability of battery size to meet mission requirements. As delivered the Cerberus has two 6V sealed lead acid batteries with 4.5 Amp-hours of capacity for a combined total of 54 Watt-hours. This should result in over 12 hours of runtime for demonstration purposes. For future units, larger batteries can be incorporated that easily extend the runtime past 24 hours. Another recommended approach would be to install a solar panel to charge the battery during daylight hours. With a properly sized power system the Cerberus should be capable of indefinite fair weather run times.

6 System Usage

The Cerberus is completely contained within the IP55 enclosure provided. This allows easy transport with protection from the environment as well as convenient storage.

Figure 1. Stowed Cerberus unit ready for deployment.

6.1 Assembly

To assemble the unit, start by opening the four latches on the case. Take the Antenna (Item 1 in Figure 2) and screw it clockwise onto the jack on the left side of the case (Item A in Figure 3). Next take the Camera/PI assembly (Item 2 in Figure 2), extend the legs of the tripod and place the camera facing the direction to be monitored. Take the connector and align the notch with the notch in the receptacle on the left side of the case (Item B in Figure 3). The connector will slide straight into the receptacle until it bottoms out, then take the locking ring and turn it clockwise until the detent is felt engaging (roughly a

quarter turn). Next connect the two yellow connectors (Item 3 in Figure 2). Close the lid and latch all four latches. On the right side of the case there is a power button (Item C in Figure 4), toggle it to the “1” position and the light should indicate power. Do not make loud noises during the initialization period, as the sound sensor uses ambient noise levels during startup to set the noise threshold dynamically.

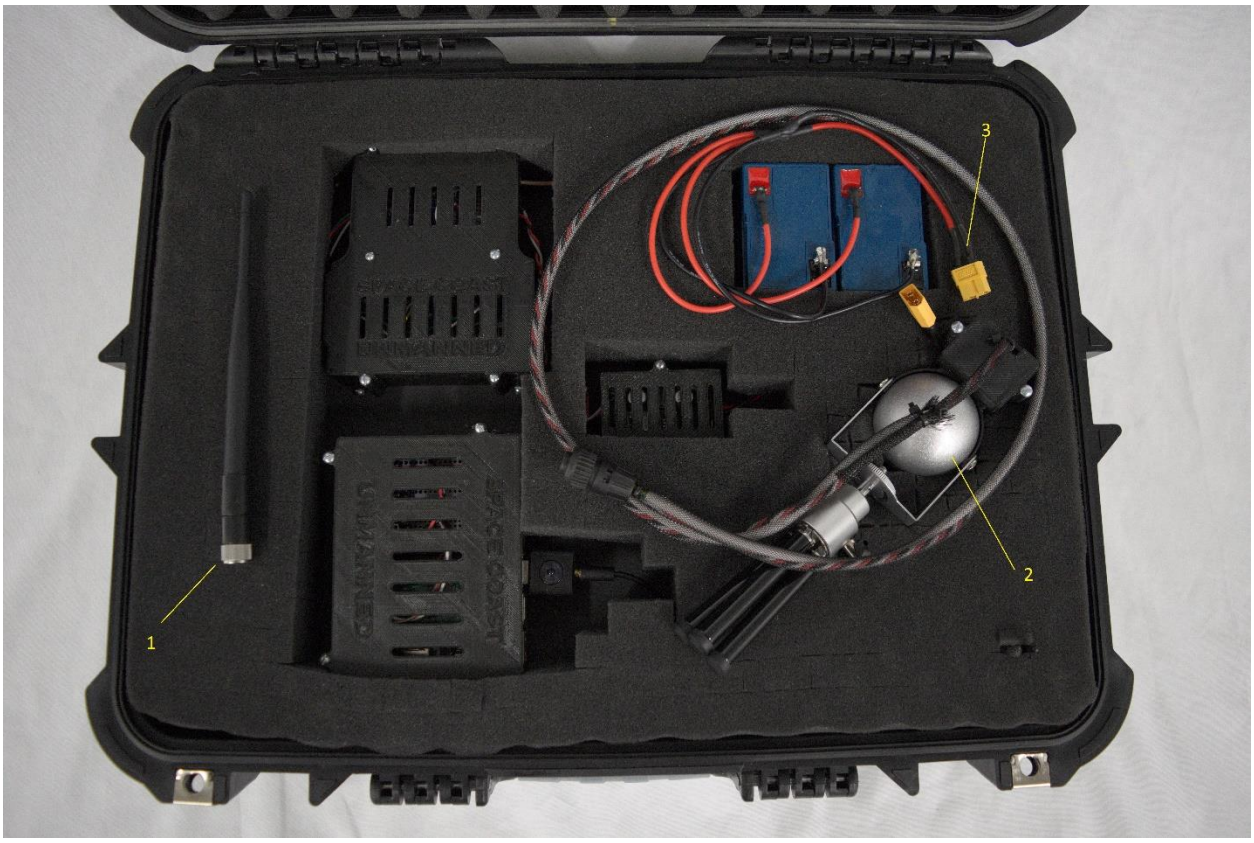

Figure 2. Overview of components for assembly.

Figure 3. Left side of case.

Figure 4. Right side of case.

6.2 Operation

It takes about 5 minutes for the Cerberus to become fully initialized. The majority of this timeline is due to the boot time of the Raspberry Pi, and initialization of the USB ports for the RTL-SDR. While the Cerberus is initializing, the receiver can be connected to the monitoring station. The Arduino IDE includes a serial monitor and can be used to monitor incoming data. Any serial client such as Putty can be used as well. In the future, a program can be written to monitor the data and output files such as .csv to allow monitoring multiple sensors without any user interaction.

6.3 Code Modification

To modify the Raspberry Pi, connect using ssh or VNC Viewer. The python scripts that are loaded during boot are found in /home/pi/FPK/ named “frs_detect_final.py” and “9dof_final.py”. Simply modifying these is sufficient, and the new version will be loaded after rebooting. To run additional scripts, modify the configuration file located at ~/.config/lxsession/LXDE-pi/autostart and add the new commands below the existing lines. To modify the Arduino, simply attach via USB and upload using the Arduino IDE.

7 Receiver Station

Figure 5. Receiver Station

7.1 Setup

The easiest method to ensure that the proper drivers are configured for the receiver is to download the Arduino IDE from https://www.arduino.cc/en/Main/Software. Once installed, attach the USB cable to the receiver and the monitoring computer. Open the Arduino IDE, click the tools tab, hover over the Port option, and select the port that indicates an Arduino Mega is attached. Once selected, click the tools tab again, and select Serial Monitor. This will bring up the monitor and start receiving output from the Cerberus remote unit.

7.2 Output

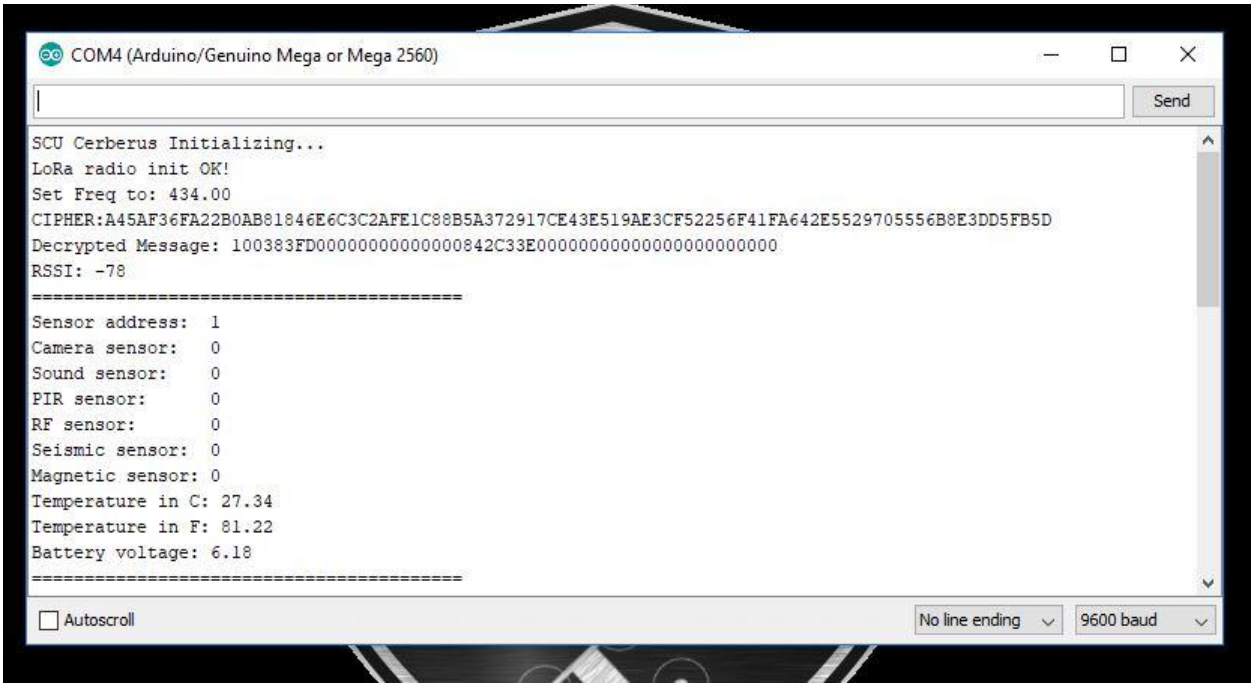

Figure 6. Receiver Output Captured via Serial Monitor

The output of the receiver shows status of all sensors, as well as the original Cipher, and Decrypted message in its raw format.

- Lines 1-3 are one time only, and show the initialization status of the Cerberus.
- Line 4-5 show the Cipher message received, and the resulting decrypted data.
- Line 6 shows the Received Signal Strength Indicator, higher numbers are stronger signals
- Line 8 shows the address of the sensor that is transmitting. This can be 0-255
- Line 9-14 shows the status of the individual sensors. A “0” means no activity, a “1” means activity has been detected from that sensor
- Line 15-16 show the temperature reported by the sensor in degrees
- Line 17 shows the voltage of the battery in Volts.

8 Summary

The Cerberus provides detection across an array of spectrums, including visible light, infrared, audio, and radio frequency. Tamper, seismic, and magnetic sensing is accomplished via a 9 degree of freedom sensor. Telemetry for temperature and battery voltage are provided along with all sensor statuses via an AES 128 encrypted wireless link at distances up to 2km line of sight. The Raspberry Pi and Arduino both have plenty of reserve capacity for additional functionality, and open source software libraries are written to give the end user maximum flexibility.

9 Remote Access for Raspberry Pi

9.1 Connect via SSH

SSH is available, but is not as user friendly as the Remote Desktop Option. Instructions for Remote Desktop are in the following section. To connect via SSH, ensure the Raspberry Pi is connected to the user machine via Ethernet cable. Open the Putty program. In the Host Name box enter: 169.254.240.117. Ensure the SSH radio button is selected and the port is 22 as shown in Figure 7. When prompted for user id input: “pi” without quotation marks. When prompted for the password input: “spacecoast” without the quotation marks and press enter. You will now be connected and can perform any SSH commands desired.

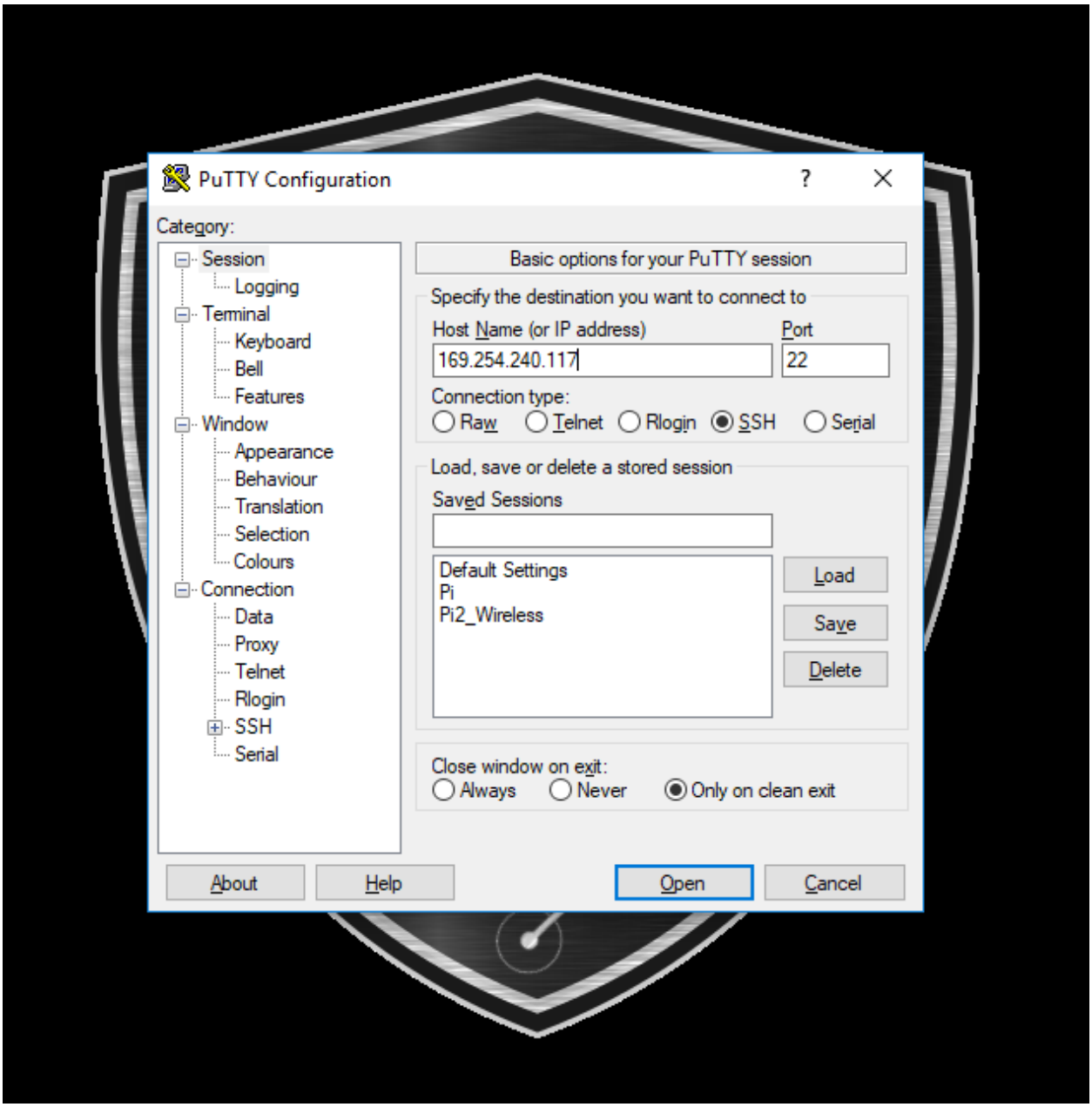

Figure 7. Putty connection configuration

9.2 Connect via VNC (Remote Desktop)

Ensure the Raspberry Pi is connected to the user machine via Ethernet cable. Open VNC viewer program. In the field for VNC address, enter 169.254.240.117 as shown in Figure 8. Input “pi” as the username and “spacecoast” as the password when prompted.

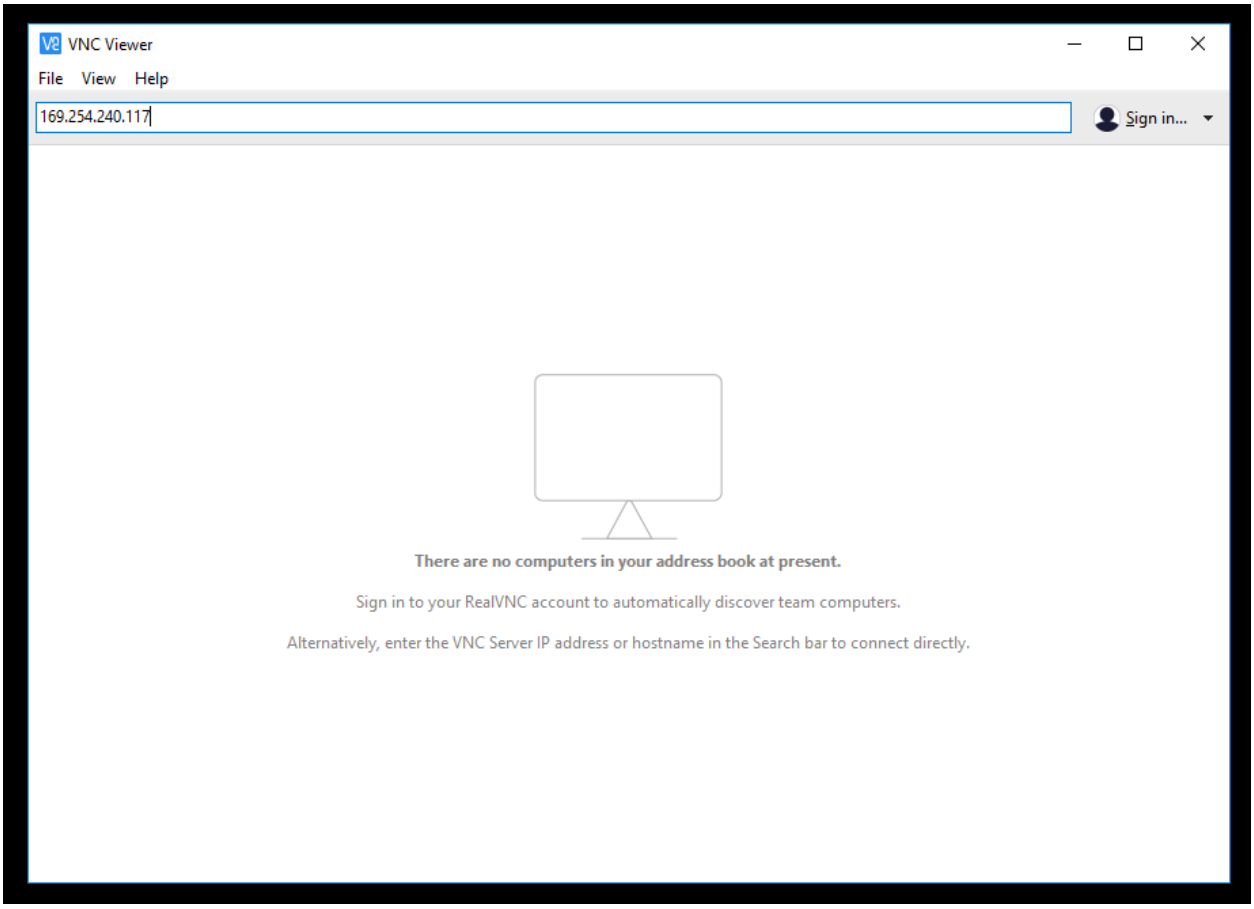

Figure 8. VNC Viewer Connection Configuration

10 External Resources Used

Multiple connection methods are available to the user to configure the Raspberry Pi:

- Putty: http://www.putty.org/
- RealVNC Viewer: https://www.realvnc.com/en/connect/download/viewer/

11 Software Libraries

- Radiohead library for RFM95: http://www.airspayce.com/mikem/arduino/RadioHead/
- Ghirlekar library for BNO055: https://github.com/ghirlekar/bno055-python-i2c
- Python wrapper for RTL-SDR: https://github.com/roger-/pyrtlsdr
- AES Encryption: https://github.com/spaniakos/AES
- Tutorials from Adafruit: https://www.adafruit.com/

12 Additional Information

12.1 GitHub Repository

The User Manual and presentation can be found in SCU's GitHub repository here:
https://github.com/Space-Coast-Unmanned/Cerberus

12.2 Module Wiring

The modules should not need to be rewired by the end user. They are included in this manual for information only.

Table 1. Pinout for Cerberus Modules

Component	Pin	Component	Pin	Notes
Arduino	2	RFM95 Transmitter	G0	
Arduino	52	RFM95 Transmitter	SCK	
Arduino	50	RFM95 Transmitter	MISO	
Arduino	51	RFM95 Transmitter	MOSI	Must be on hardware interrupt capable pin
Arduino	4	RFM95 Transmitter	CS	
Arduino	3	RFM95 Transmitter	RST	
Arduino	A16(69)	Voltage Divider 1	R1A	
Arduino	A2	LM35	Vout	
Arduino	10	Camera	Green	
Arduino	8	PIR Sensor	Vout	
Arduino	A1	Microphone	Out	
Arduino	A0	Voltage Divider 2	R1B/R2A	
Arduino	5V	Power Rail	5V	
Arduino	Gnd	Power Rail	Gnd	
Pi	3	BNO055	SDA	
Pi	5	BNO055	SCL	
Pi	12	BNO055	RST	
Pi	31	Arduino	46	RF Trigger
Pi	33	Arduino	47	Seismic Trigger
Pi	29	Arduino	48	Magnetic Trigger
Pi	4	Power Rail	5V	
Pi	6	Power Rail	Gnd	
Voltage Divider 1	R1B/R2A	Camera	White	R1 = 10k, R2 = 10k

Component	Pin	Component	Pin	Notes
Voltage Divider 1	R2B	Power Rail	Gnd	R1 = 10k, R2 = 10k
Voltage Divider 2	R1A	Battery	V+	R1 = 3.3k, R2 = 10k
Voltage Divider 2	R2B	Power Rail	Gnd	R1 = 3.3k, R2 = 10k
BNO055	Vin	Power Rail	5V	
BNO055	Gnd	Power Rail	Gnd	
LM35	Vin	Power Rail	5V	
LM35	Gnd	Power Rail	Gnd	
Microphone	Vdd	Power Rail	5V	
Microphone	Gnd	Power Rail	Gnd	
RFM95 Transmitter	Vin	Power Rail	5V	
RFM95 Transmitter	Gnd	Power Rail	Gnd	

12.3 Message Format

The message is automatically formatted for both transmission and receiving. The word is currently 47 bytes, with 9 bytes used. The unused bytes can be utilized for additional functionality, and if necessary the message size can be increased at a later date if necessary.

Table 2. Sensor Address Word

Byte 0 (Sensor Address, 0-255)		
Bit	**State**	**Item**
7	0: Not Active 1: Active	MSB
6	0: Not Active 1: Active	
5	0: Not Active 1: Active	
4	0: Not Active 1: Active	
3	0: Not Active 1: Active	

Byte 0 (Sensor Address, 0-255)		
Bit	**State**	**Item**
2	0: Not Active 1: Active	
1	0: Not Active 1: Active	
0	0: Not Active 1: Active	LSB

Table 3. Sensor Status Word

Byte 1 (Sensor Status)		
Bit	**State**	**Item**
7	0: Not Triggered 1: Triggered	
6	0: Not Triggered 1: Triggered	
5	0: Not Triggered 1: Triggered	Magnetic Sensor
4	0: Not Triggered 1: Triggered	Disturbance/Seismic Sensor
3	0: Not Triggered 1: Triggered	RF Sensor
2	0: Not Triggered 1: Triggered	PIR Sensor
1	0: Not Triggered 1: Triggered	Microphone
0	0: Not Triggered 1: Triggered	Camera

Table 4. Temperature Upper Half Word

Byte 2 (Temperature Upper Half)		
Bit	**State**	**Item**
7	0: Not Active 1: Active	Temp 15
6	0: Not Active 1: Active	Temp 14
5	0: Not Active 1: Active	Temp 13
4	0: Not Active 1: Active	Temp 12
3	0: Not Active 1: Active	Temp 11
2	0: Not Active 1: Active	Temp 10
1	0: Not Active 1: Active	Temp 9
0	0: Not Active 1: Active	Temp 8

Table 5. Temperature Lower Half Word

Byte 3 (Temperature Lower Half)		
Bit	**State**	**Item**
7	0: Not Active 1: Active	Temp 7
6	0: Not Active 1: Active	Temp 6

5	0: Not Active 1: Active	Temp 5
4	0: Not Active 1: Active	Temp 4
3	0: Not Active 1: Active	Temp 3
2	0: Not Active 1: Active	Temp 2
1	0: Not Active 1: Active	Temp 1
0	0: Not Active 1: Active	Temp 0

Table 6. Battery Upper Half Word

Byte 4 (Battery Upper Half)		
Bit	**State**	**Item**
7	0: Not Active 1: Active	Battery 15
6	0: Not Active 1: Active	Battery 14
5	0: Not Active 1: Active	Battery 13
4	0: Not Active 1: Active	Battery 12
3	0: Not Active 1: Active	Battery 11
2	0: Not Active 1: Active	Battery 10

1	0: Not Active 1: Active	Battery 9
0	0: Not Active 1: Active	Battery 8

Table 7. Battery Lower Half Word

Byte 5 (Battery Lower Half)		
Bit	**State**	**Item**
7	0: Not Active 1: Active	Battery 7
6	0: Not Active 1: Active	Battery 6
5	0: Not Active 1: Active	Battery 5
4	0: Not Active 1: Active	Battery 4
3	0: Not Active 1: Active	Battery 3
2	0: Not Active 1: Active	Battery 2
1	0: Not Active 1: Active	Battery 1
0	0: Not Active 1: Active	Battery 0

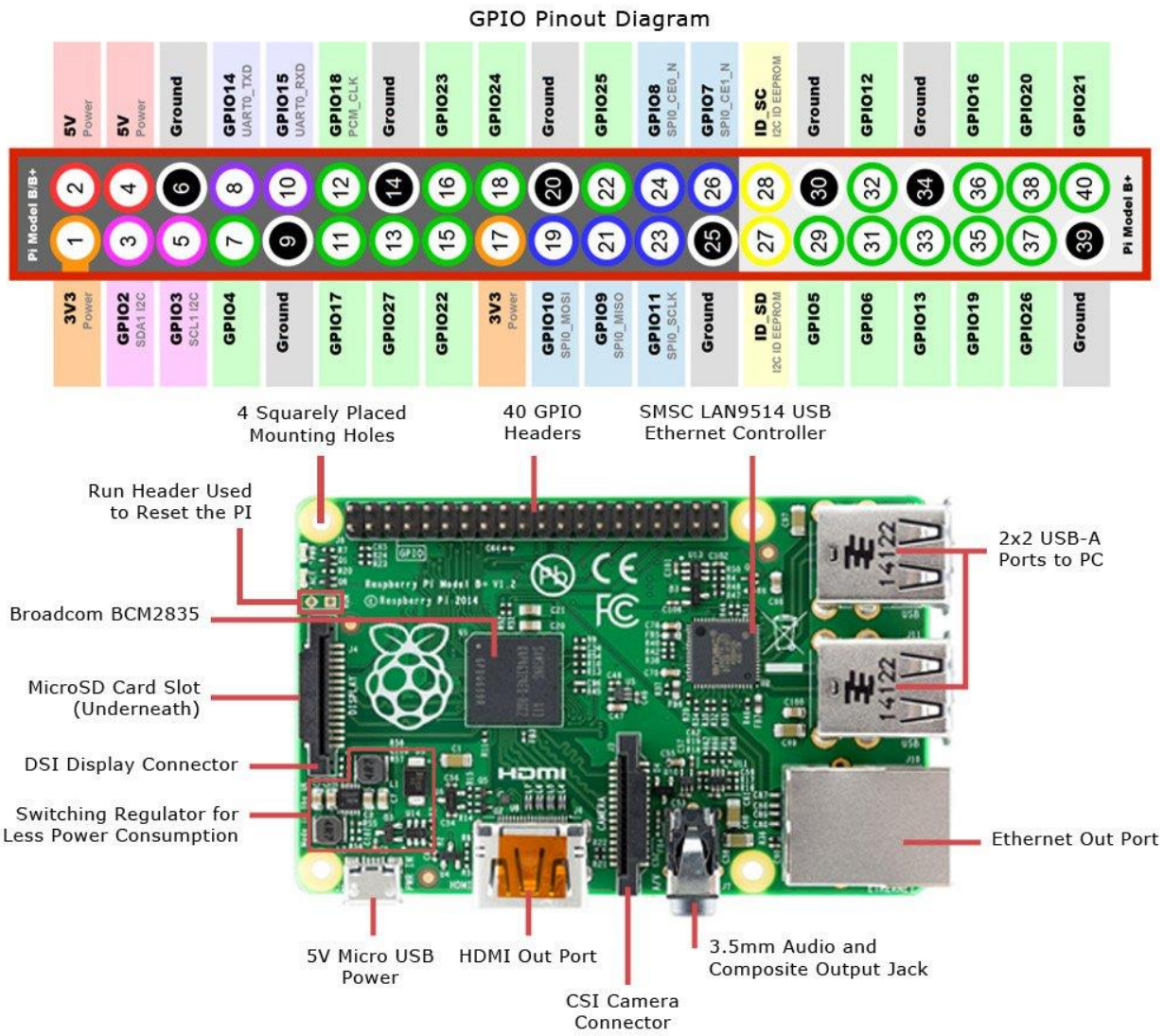

Figure 9. Raspberry Pi Pinout

Pinout picture used from:
http://www.jameco.com/Jameco/workshop/circuitnotes/raspberry_pi_circuit_note_fig2a.jpg