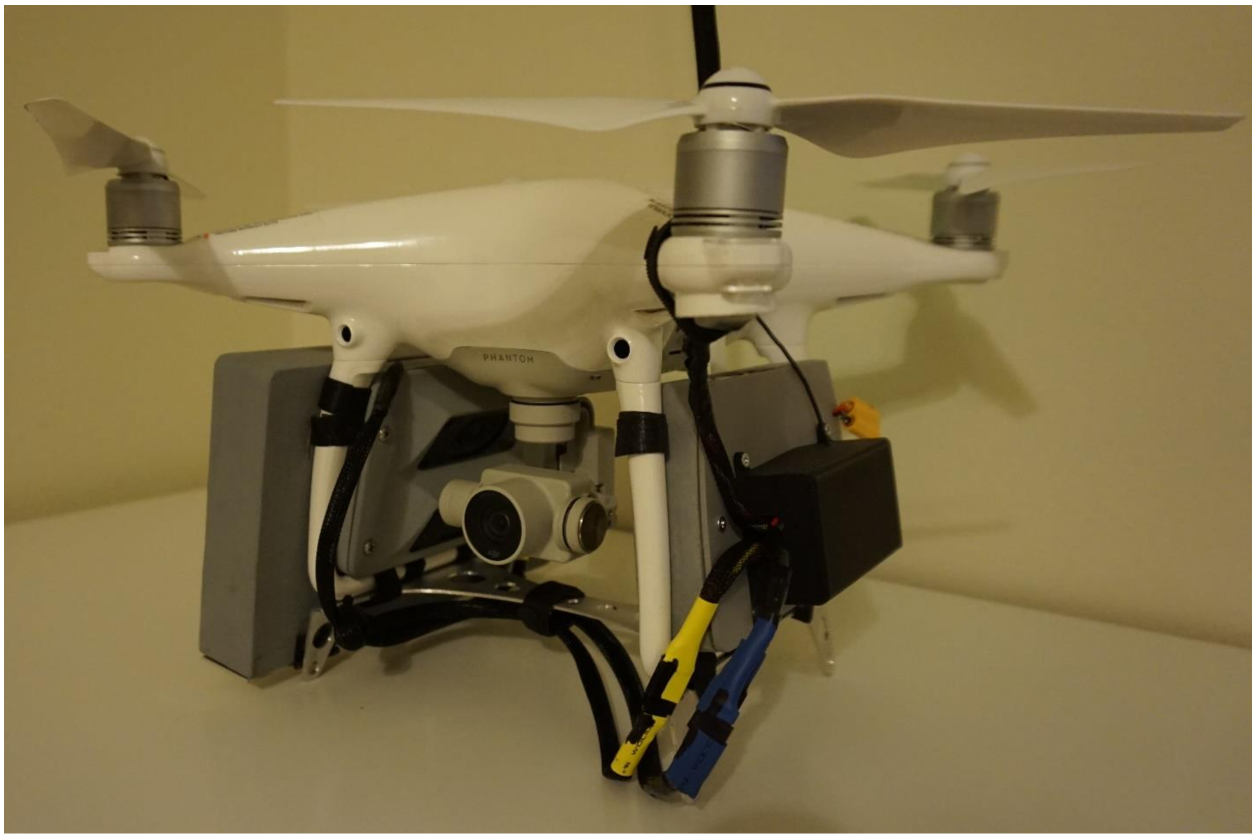
SPACE COAST UNMANNED

Airspew System User Manual



AUGUST 21, 2017

Matthew Colvin, James Tolson

Table of Contents

Table of Figures

List of Tables

Overview

The Space Coast Unmanned (SCU) AirSpew system consists of the following modules:

- Left Saddle (Figure 1)
  - Raspberry Pi
  - FM Transmitter
  - Audio Amplifier
  - Voltage Regulator
  - Battery for SCU AirSpew system

- Right Saddle
  - Pamphlet holder/dropper assembly (Figure 2)
  - Audio speakers (Figure 3)
- Landing gear extension (Figure 4)

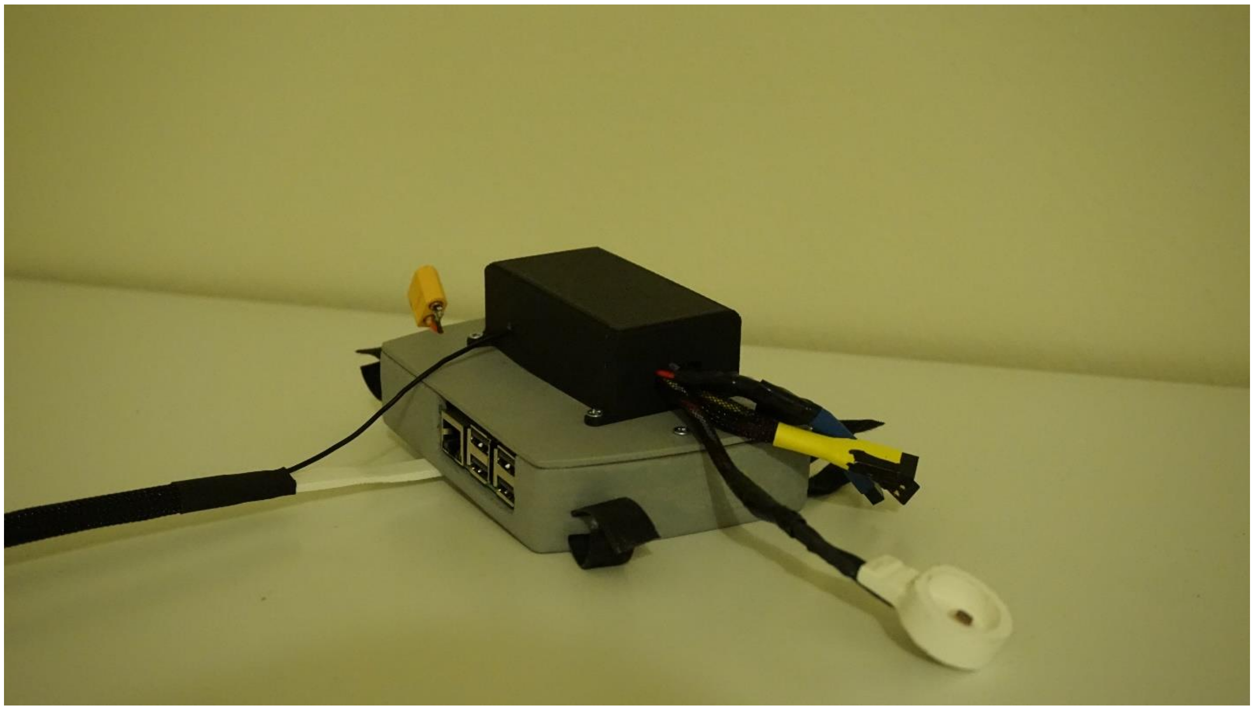


Figure 1. Left saddle module

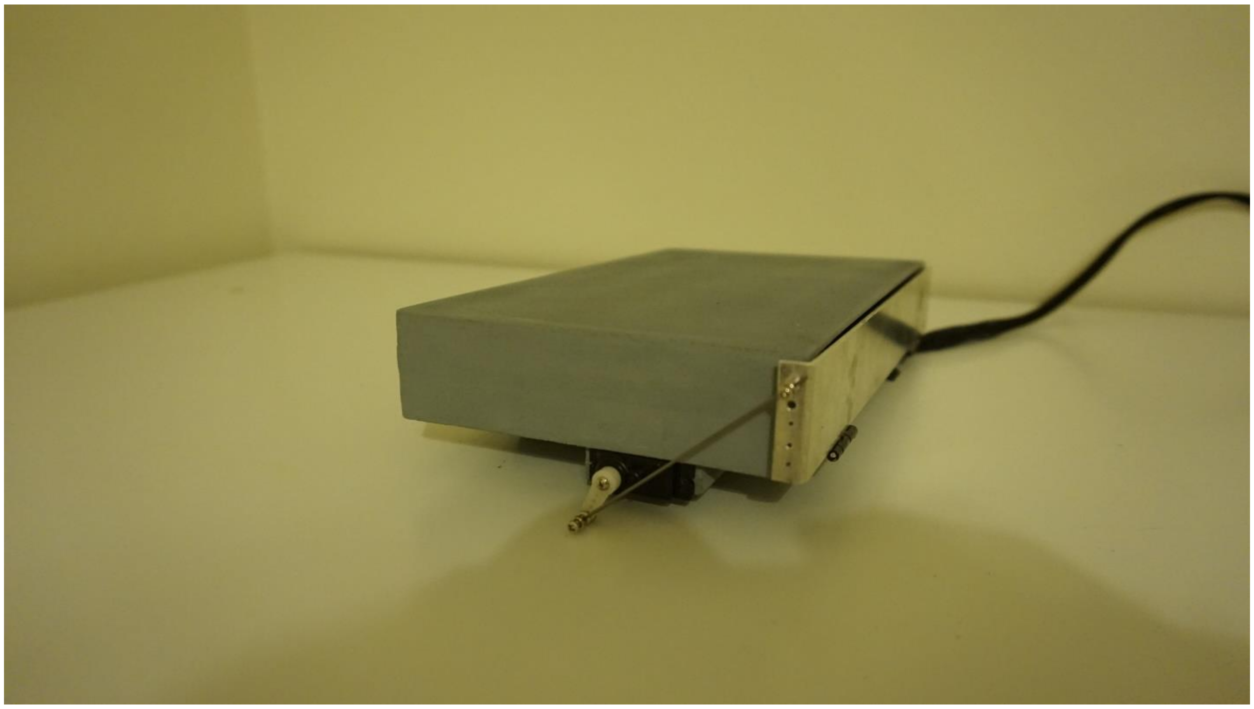


Figure 2. Pamphlet dropper module

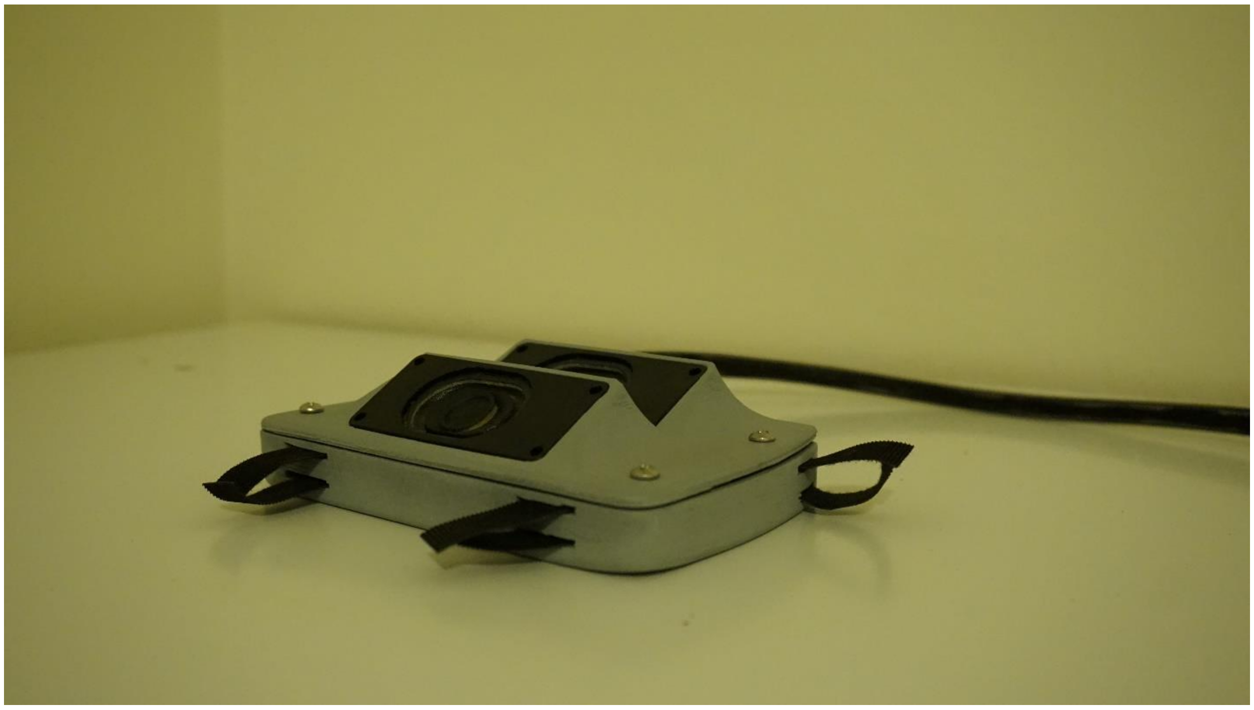


Figure 3. Audio speaker module

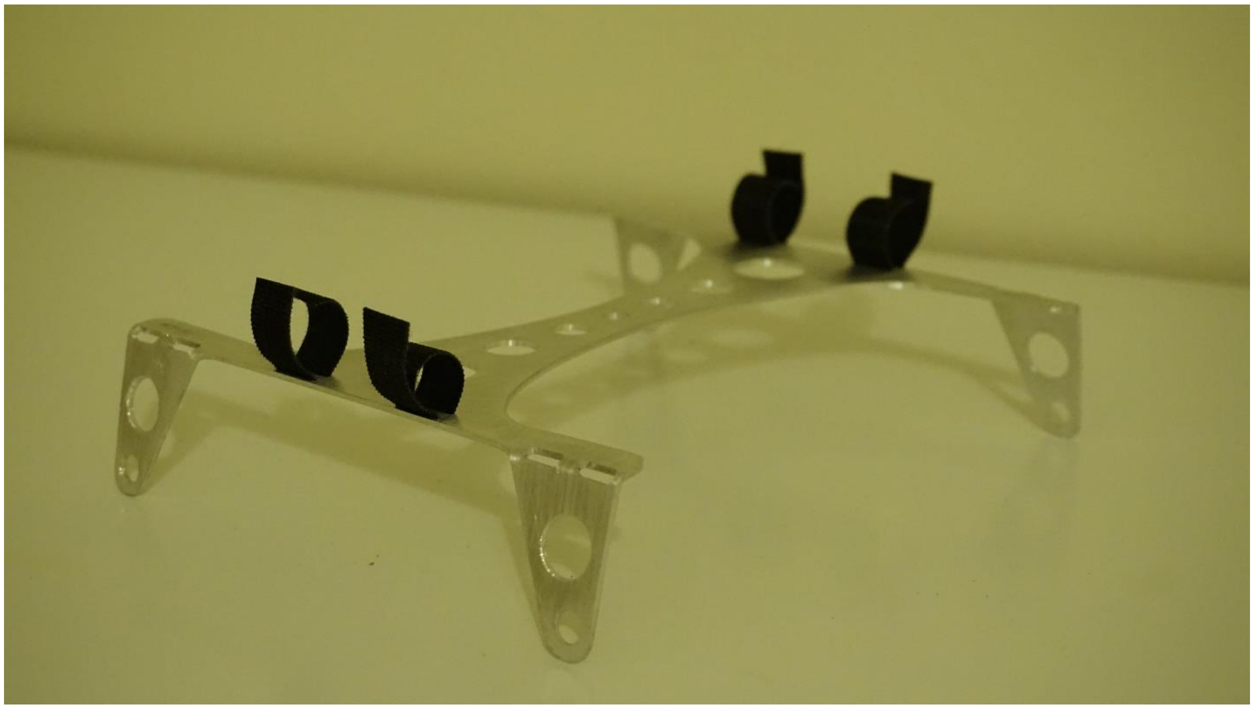


Figure 4. Landing Gear Extension

The SCU Airspew system has been configured to automatically load all Airspew functionality on boot. Boot process takes roughly 10 seconds. No user actions are required for configuration of a mission other than powering the system. The raspberry pi has been configured with a static IP address of 169.254.35.156. The username is *pi* and the password is *airspew*. The pi automatically executes programs specified in the file "~/.config/lxsession/LXDE-pi/autostart". No changes are required to use the SCU Airspew system, if necessary in the future this file can be modified using a text editor.

System Usage

Assembly

Attach the landing gear extension using the supplied Velcro as shown in Figure 5.

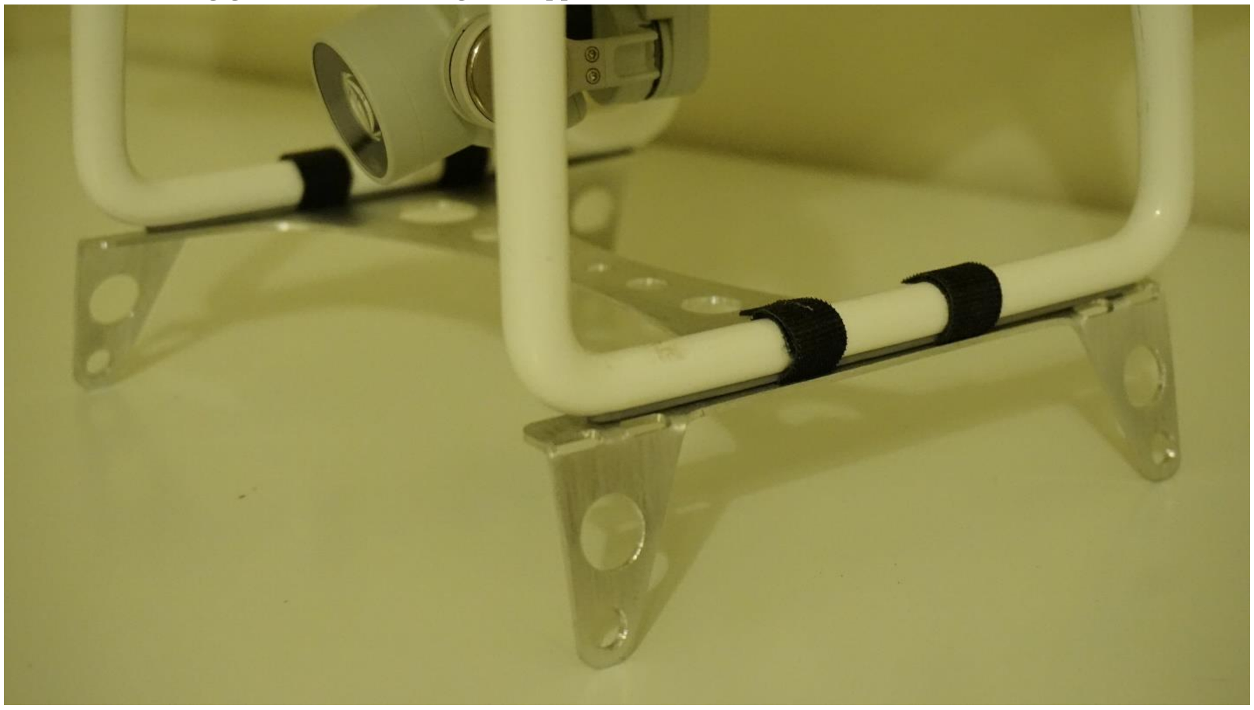


Figure 5. Landing gear extension installed

Remove the clear LED cover on the front left light by prying gently. Insert the supplied light sensor unit, and use Velcro to secure sensor into the socket as shown in Figure 6.

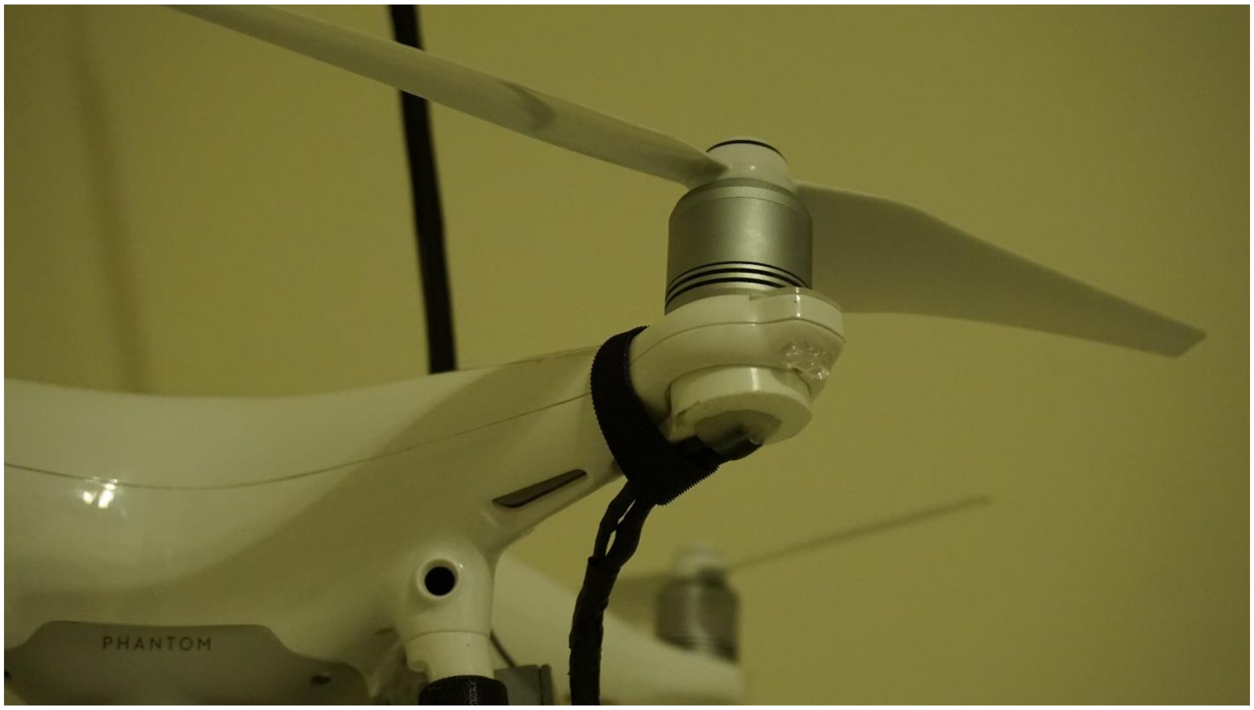


Figure 6. Light sensor installed

Attach the left saddle to the left-side landing gear using the supplied Velcro as shown in Figure 7.

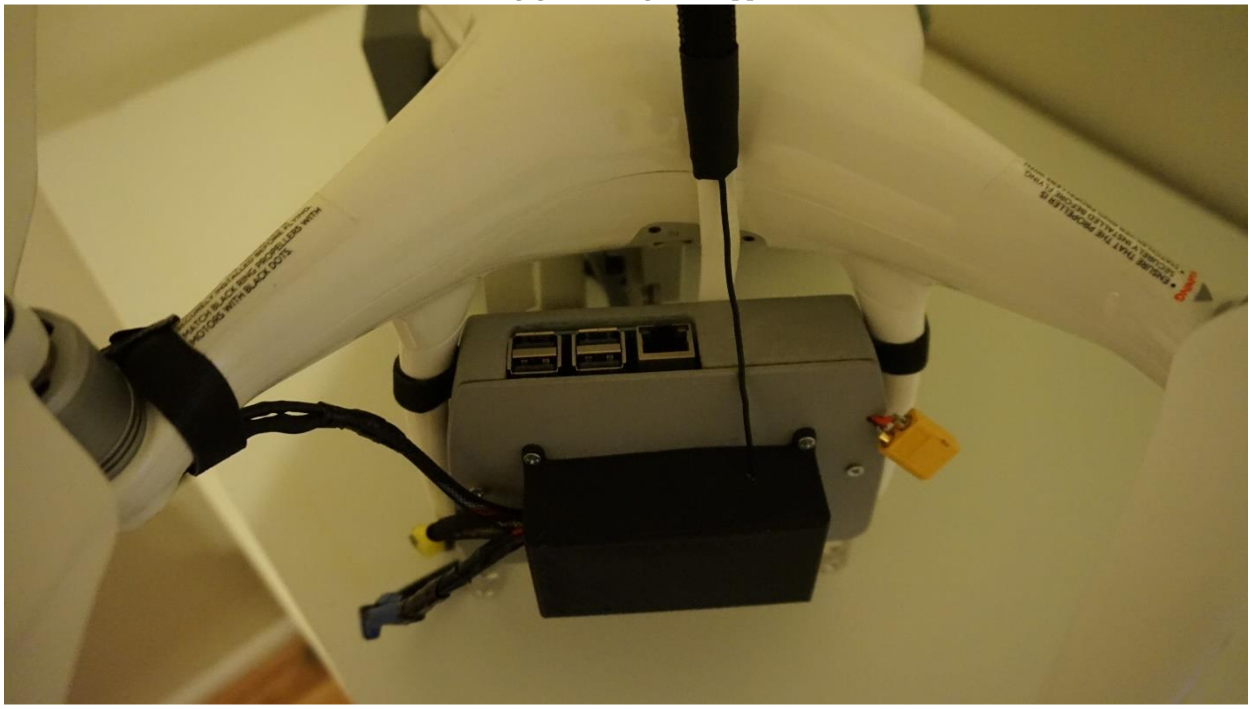


Figure 7. Left saddle module installed

Figure 8. Left saddle closeup

The left saddle can be omitted if FM transmit only functionality is desired. There are three additional configurations. To mount both the audio speakers and the pamphlet dropper, attach the right saddle to the right-side landing gear using the supplied Velcro as shown in Figure 9.

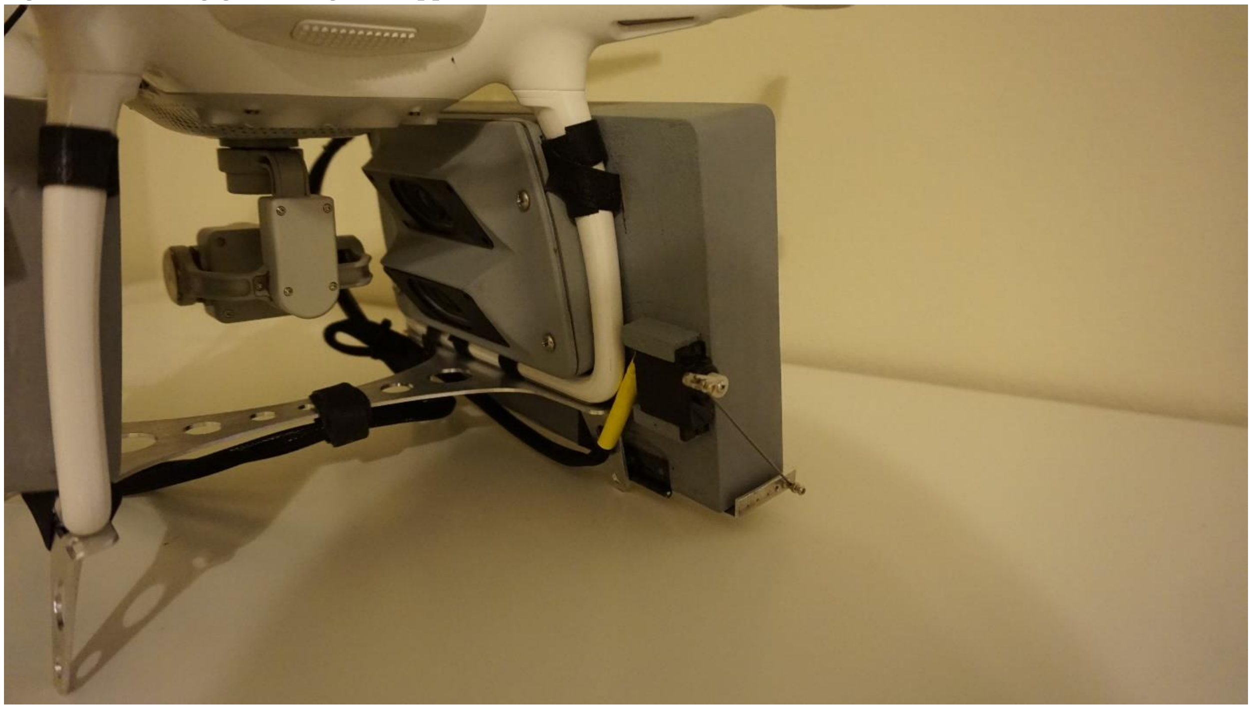


Figure 9. Audio speakers and pamphlet dropper installed

To install only the pamphlet dropper, simply leave the audio module uninstalled. To install only the audio speakers, the speakers can be reversed to point towards the exterior of the Phantom as shown in Figure 10.

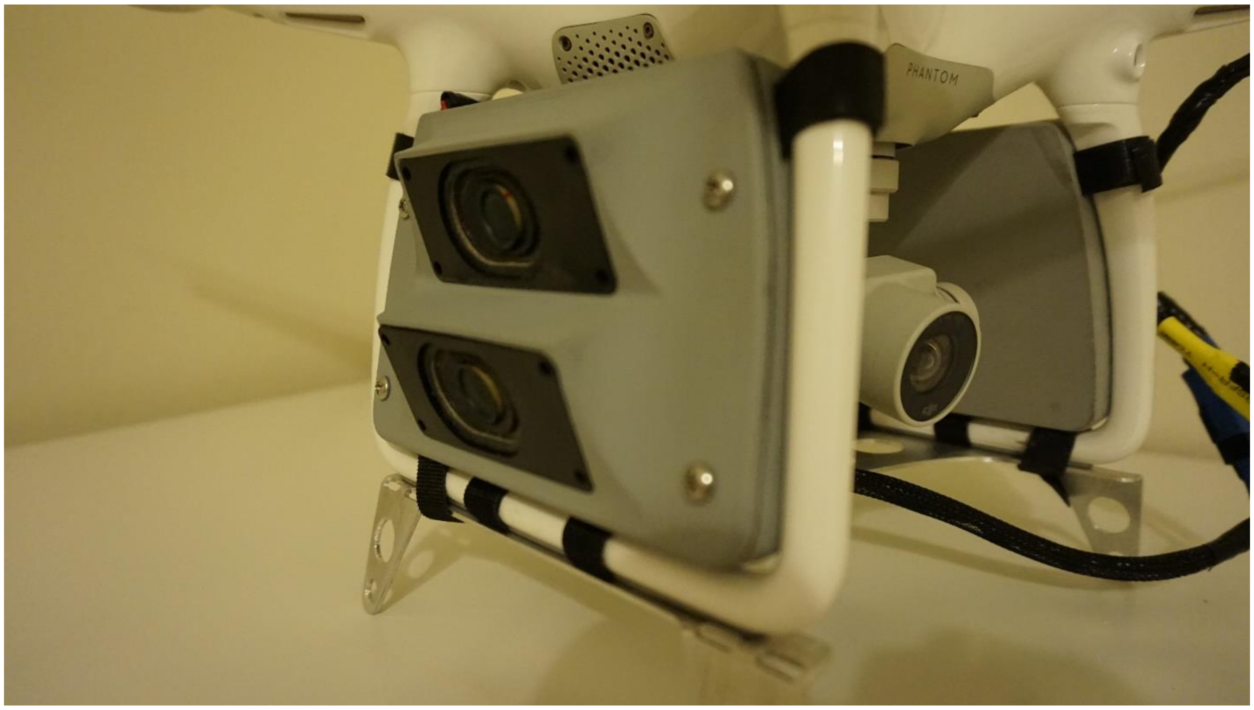


Figure 10. Audio speakers reversed without pamphlet dropper

Connect the wiring from the right saddle to the left saddle as shown in Figure 11. Note 1: the black lines indicate polarity and should match. Note 2: Be sure the wiring does not interfere with the pamphlet dropper door.

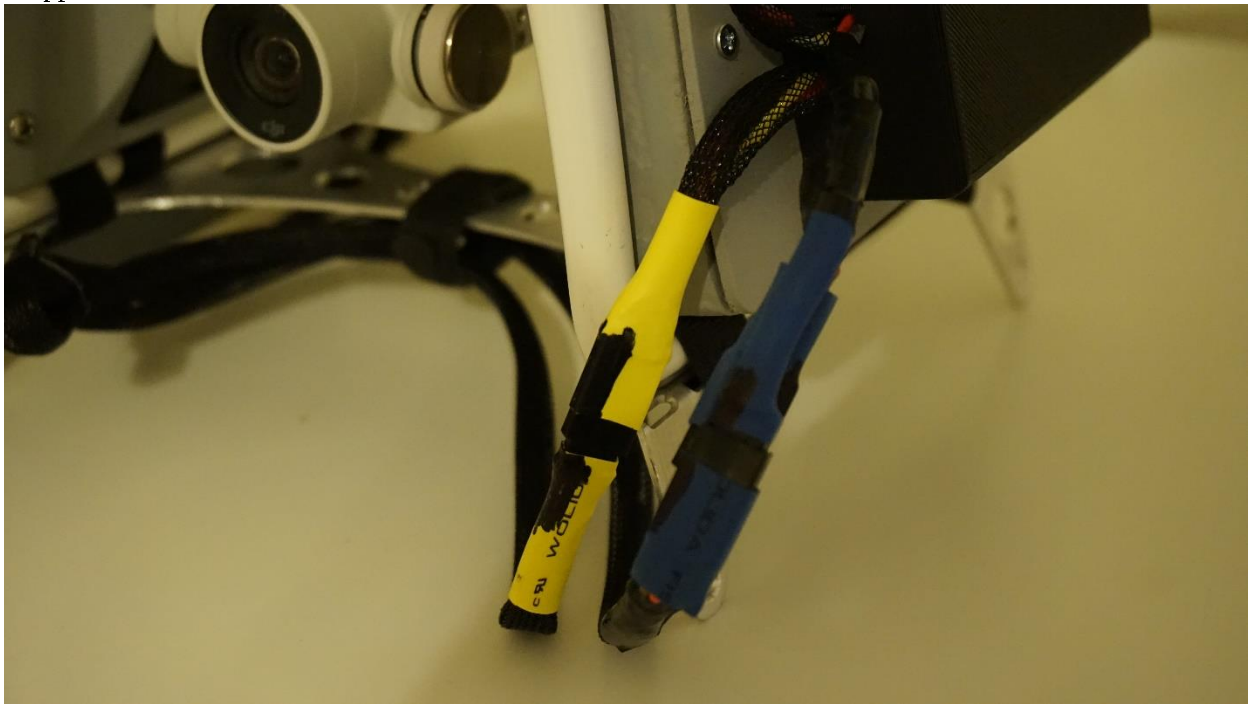


Figure 11. Module wiring connections

Operation

Power on the Phantom 4 without powering the SCU Airspew system. After Phantom boot has completed, test forward LED operation. If LEDs have not been configured, use the DJI app to configure the LEDs to be controlled via button.

Once LEDs are confirmed operational, power on SCU Airspew system. After the boot timing, the system is ready for use. It is recommended to test the

FM Transmitter Settings

By default, the FM transmitter is playing on 103.8 MHz. This can be customized by editing the Python script variable named `FMSTATION` (code line number 388).

The Si4713 output power can be changed by editing the Python script variable `POWER (code line number 389).`

External Resources Used

Adafruit Si4713 breakout module: https://www.adafruit.com/product/1958

Adafruit TPA2016 breakout module: https://www.adafruit.com/product/1712

An adaptation of Adafruit's Si4713 arduino code was created for Python on a Raspberry Pi. This code was modified by Space Coast Unmanned for use in the Airspew challenge. The original source code can be found here: https://github.com/daniel-j/Adafruit-Si4713-RPi

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

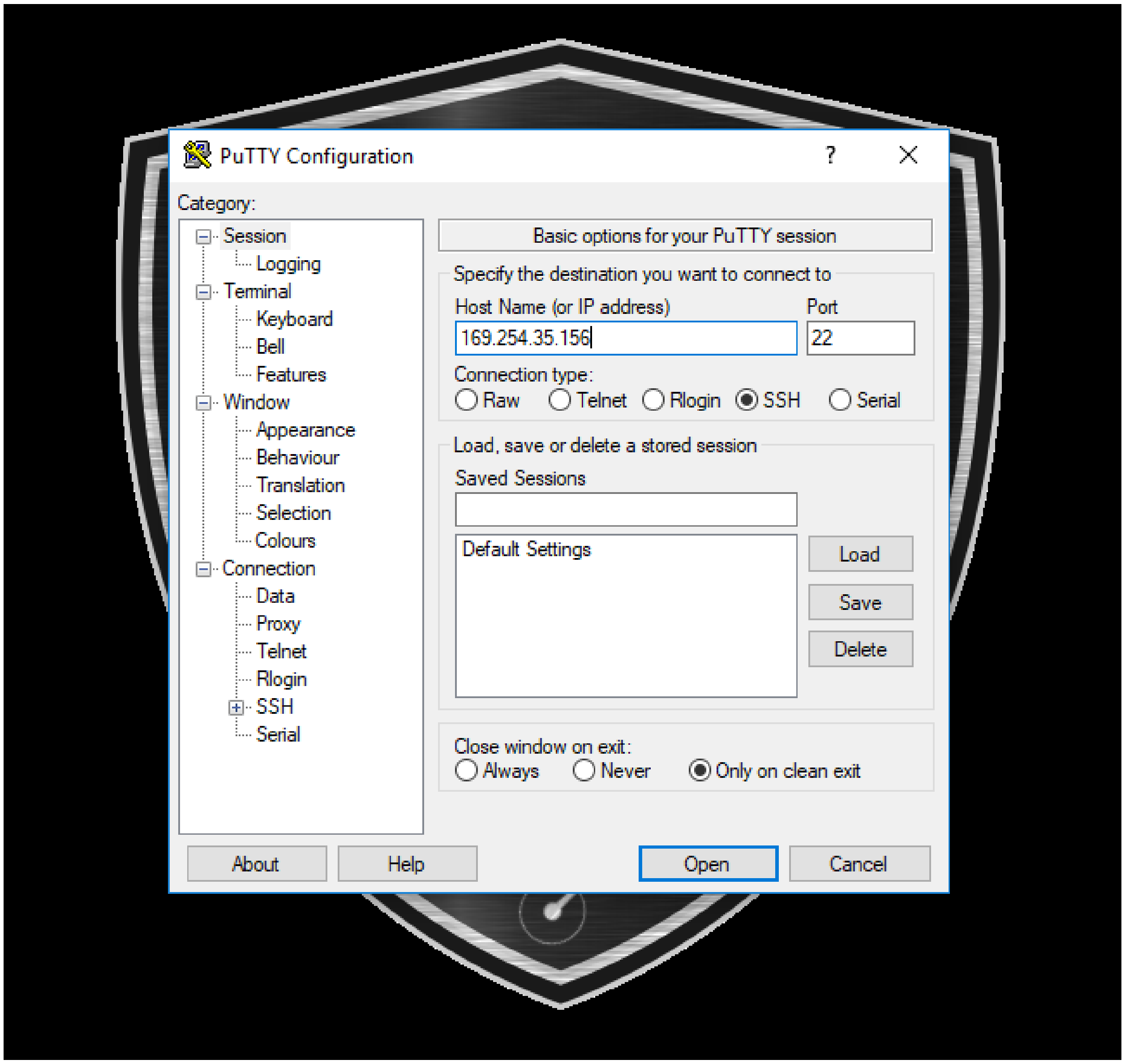
Putty: http://www.putty.org/

RealVNC Viewer: https://www.realvnc.com/en/connect/download/viewer/

Remote Access for Raspberry Pi

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: 169.254.35.156. Ensure the SSH radio button is selected and the port is 22 as shown in Figure 12. When prompted for user id input: “pi” without quotation marks. When prompted for the password input: “airspew” without the quotation marks and press enter. You will now be connected and can perform any SSH commands desired.


PuTTY Configuration

Category:

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 169.254.35.156

Port: 22

Connection type: Raw / Telnet / Rlogin / [✓ SSH] / Serial

Load, save or delete a stored session

Saved Sessions

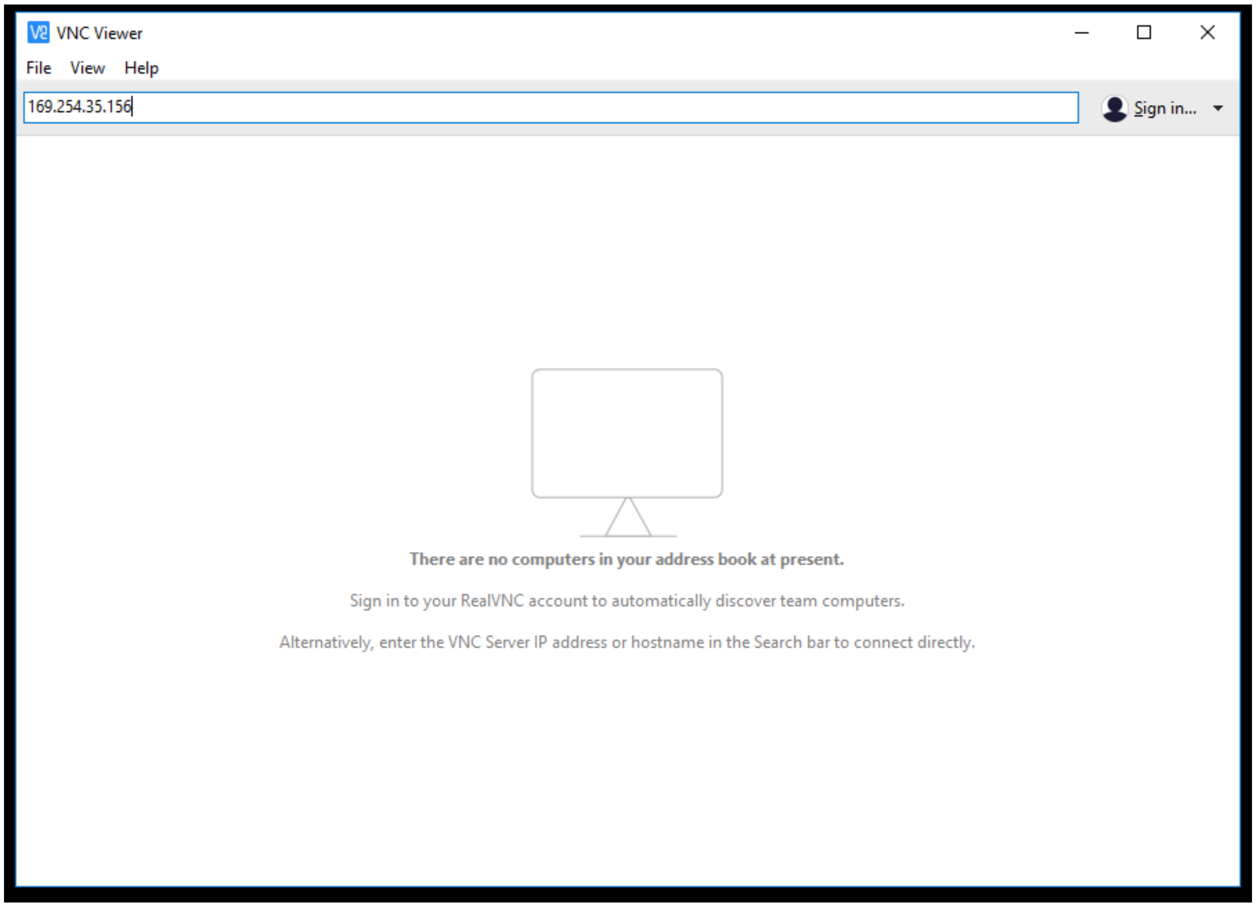
Default Settings

Close window on exit: Always / Never / [✓ Only on clean exit]

Figure 12. Putty connection configuration

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 169.254.35.156 as shown in Figure 13. Input “pi” as the username and “airspew” as the password when prompted.



There are no computers in your address book at present.

Sign in to your RealVNC account to automatically discover team computers.

Alternatively, enter the VNC Server IP address or hostname in the Search bar to connect directly.

Figure 13. VNC Viewer Connection Configuration

Adding Audio Files to Play

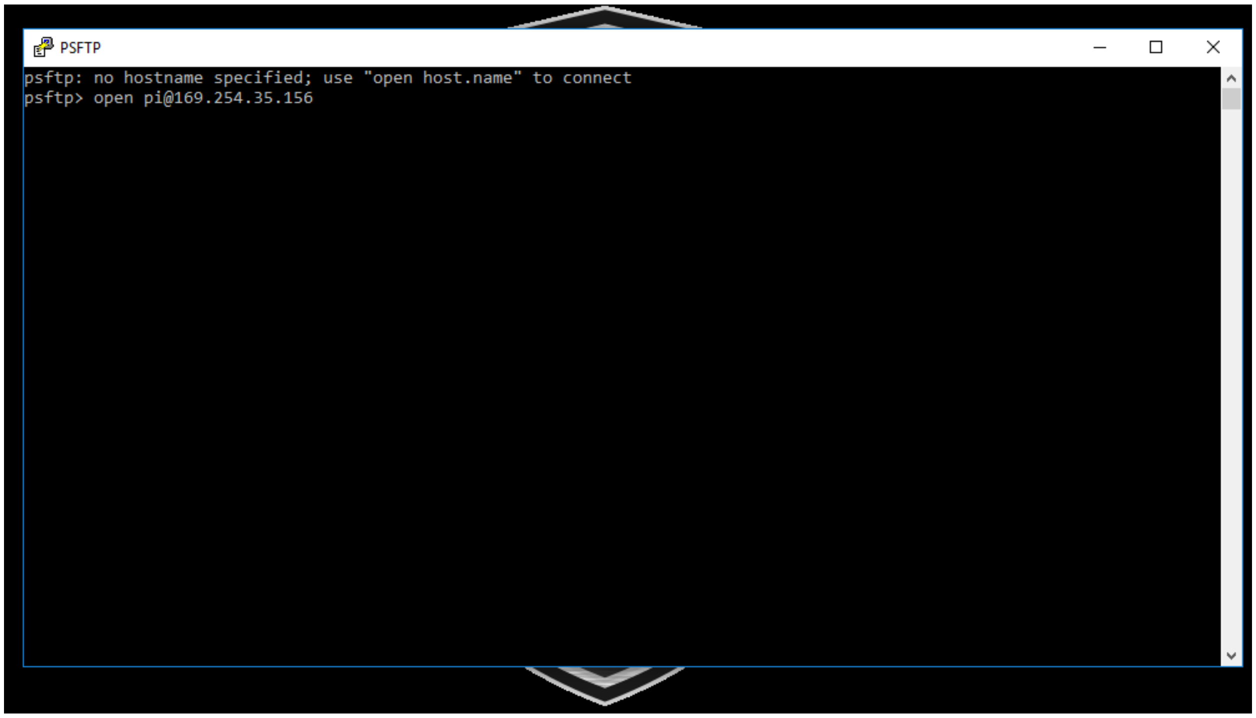
SSH and VNC do not offer a direct file transfer method. The easiest method is to load the music onto a USB drive, and use VNC to copy/paste it into the Music folder on the Pi.

The putty install does include a program called PSFTP which allows file transfer. Ensure the Raspberry Pi is connected to the user machine via Ethernet cable. All music files in the folder called “Music” on the Raspberry Pi desktop will be played in a loop. MP3 and WAV files have been tested to work.

Open PSFTP. Type

“open pi@169.254.35.156”

Without the quotation marks as shown in Figure 14.


PSFTP

```
psftp: no hostname specified; use "open host.name" to connect
psftp> open pi@169.254.35.156
```

Figure 14. PSFTP connection

When prompted for password input: “airspew” without the quotation marks. Once connected a new prompt line will appear. To place a file, using the following command: “put /localmachine/directory/localfile.ext /home/pi/Desktop/Music/”. Replace the local machine directory with the location of the file you wish to place on the Raspberry Pi. For example, if the file song.mp3 was in C:\Users\Airspew\Music, the command would be

“put C:\Users\Airspew\Music\song.mp3 /home/pi/Desktop/Music”

NOTE: Directory names on the Raspberry Pi are case sensitive, “Music” is not the same as “music”. Once finished transferring files, input “exit” and press enter and PSFTP will close. Verify the files are in the desired location using SSH or VNC.

Additional Information

GitHub Repository

Schematics, user manual, presentation, and more can be found at SCU’s GitHub repository here: https://github.com/Space-Coast-Unmanned/AirSpew

Module Wiring

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

Si4713 Pin	Connection Pin
SDA	Raspberry Pi 3
SCL	Raspberry Pi 5

RST	Raspberry Pi 7
Vin	5V Supply
GND	Ground

Table 1. Si4713 Pinout

Audio Amp Pin	**Connection Pin**
Vdd	5V Supply
GND	Ground
L+	3.5mm signal
L-	3.5mm shield
R+	3.5mm signal
R-	3.5mm shield

Table 2. Audio Amplifier Pinout

Light Sensor Pin	**Connection Pin**
Vin	5V Supply
GND	Ground
Signal	Raspberry Pi 11

Table 3. Light Sensor Pinout

Servo Pin	**Connection Pin**
Vin	5V Supply
GND	Ground
Signal	Raspberry Pi 12

Table 4. Pamphlet Dropper Servo Pinout

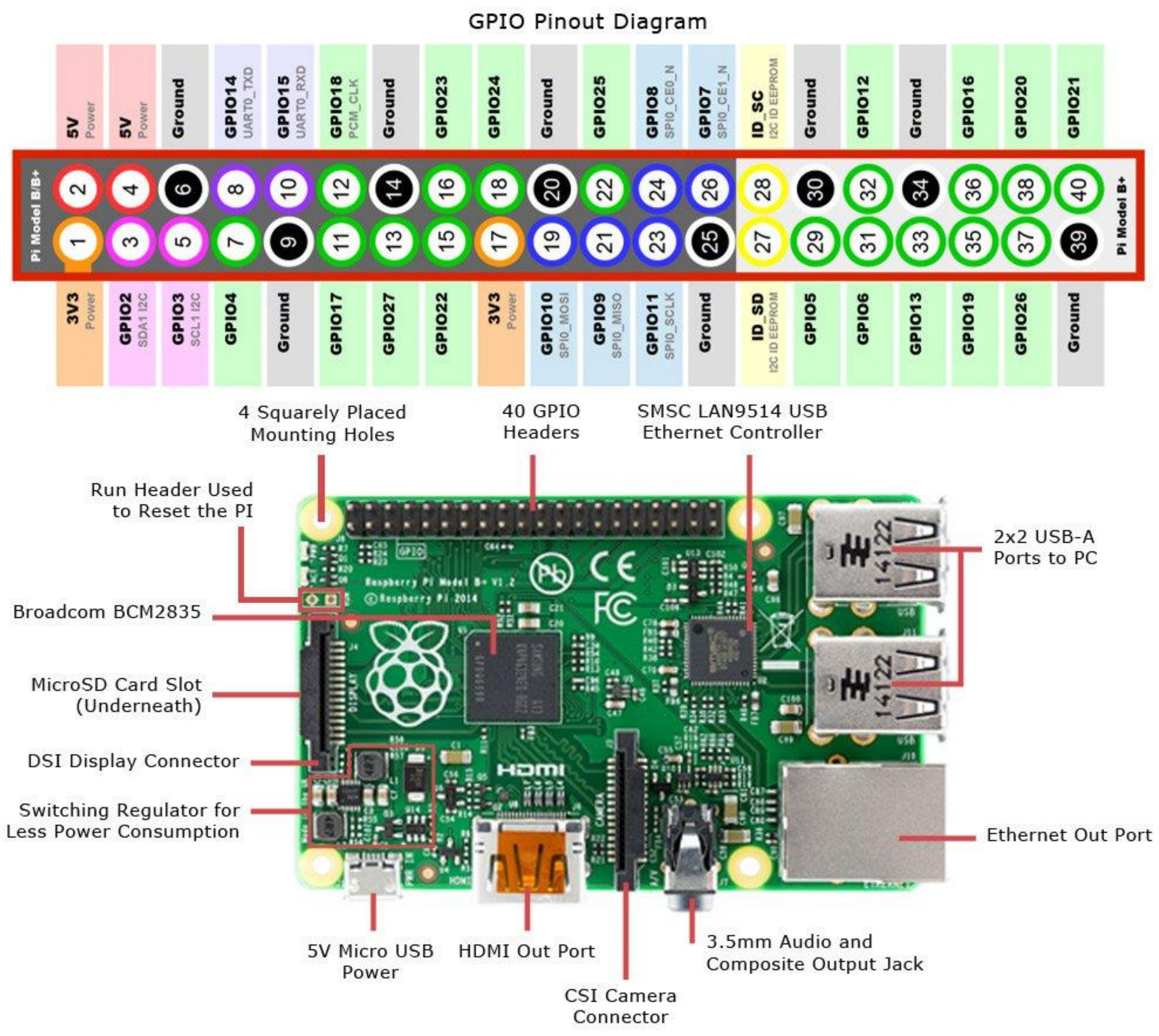


Figure 15. Raspberry Pi Pinout

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

RF Amplifier Schematic

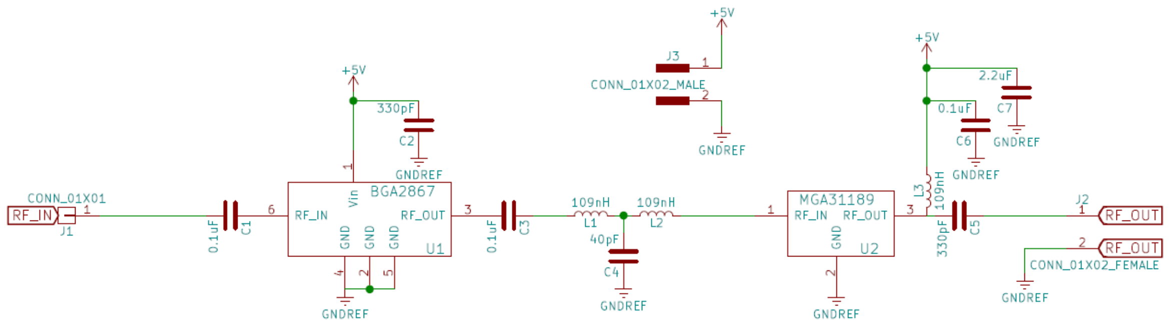


Figure 16. RF Amplifier Schematic

Switching Voltage Regulator Schematic

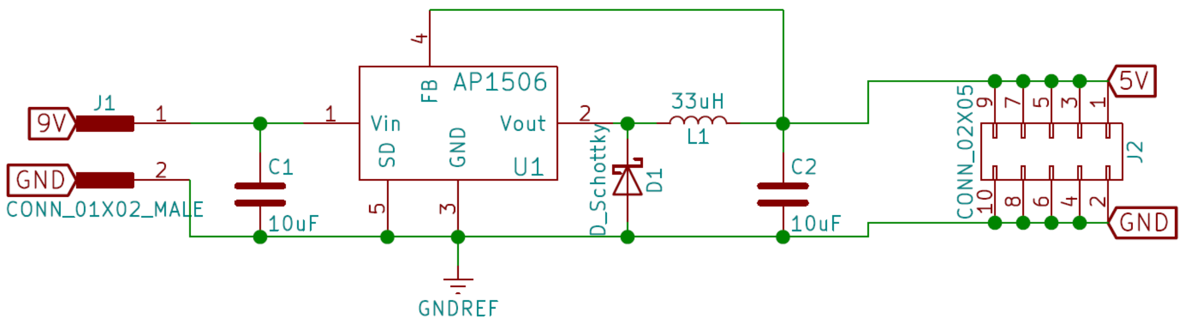


Figure 17. Switching Voltage Regulator Schematic

Python Code

Below is a capture of the Python code that is running on the Raspberry Pi.

```python
#!/usr/bin/python
import RPi.GPIO as GPIO
import time
from time import sleep
import os
import signal
import subprocess

#Code to setup transmitter starts below
#SI4713 code was made available by Adafruit

from Adafruit_I2C import Adafruit_I2C

class Adafruit_Si4713(Adafruit_I2C):

    SI4710_ADDR0 = 0x11   # if SEN is low
    SI4710_ADDR1 = 0x63   # if SEN is high, default!
    SI4710_STATUS_CTS = 0x80

    ## Commands ##
    SI4710_CMD_POWER_UP        = 0x01
    SI4710_CMD_GET_REV         = 0x10
    SI4710_CMD_POWER_DOWN      = 0x11
    SI4710_CMD_SET_PROPERTY    = 0x12
    SI4710_CMD_GET_PROPERTY    = 0x13
    SI4710_CMD_GET_INT_STATUS  = 0x14
    SI4710_CMD_PATCH_ARGS      = 0x15
    SI4710_CMD_PATCH_DATA      = 0x16
    SI4710_CMD_TX_TUNE_FREQ    = 0x30
    SI4710_CMD_TX_TUNE_POWER   = 0x31
    SI4710_CMD_TX_TUNE_MEASURE = 0x32
    SI4710_CMD_TX_TUNE_STATUS  = 0x33
    SI4710_CMD_TX_ASQ_STATUS   = 0x34
    SI4710_CMD_TX_RDS_BUFF     = 0x35
    SI4710_CMD_TX_RDS_PS       = 0x36
    SI4710_CMD_TX_AGC_OVERRIDE = 0x48
    SI4710_CMD_GPO_CTL         = 0x80
    SI4710_CMD_GPO_SET         = 0x81

    ## Parameters ##
    SI4713_PROP_GPO_IEN = 0x0001
    SI4713_PROP_DIGITAL_INPUT_FORMAT = 0x0101
    SI4713_PROP_DIGITAL_INPUT_SAMPLE_RATE = 0x0103
    SI4713_PROP_REFCLK_FREQ = 0x0201
    SI4713_PROP_REFCLK_PRESCALE = 0x0202
    SI4713_PROP_TX_COMPONENT_ENABLE = 0x2100
    SI4713_PROP_TX_AUDIO_DEVIATION = 0x2101
    SI4713_PROP_TX_PILOT_DEVIATION = 0x2102
    SI4713_PROP_TX_RDS_DEVIATION = 0x2103
    SI4713_PROP_TX_LINE_LEVEL_INPUT_LEVEL = 0x2104
    SI4713_PROP_TX_LINE_INPUT_MUTE = 0x2105
    SI4713_PROP_TX_PREEMPHASIS = 0x2106
    SI4713_PROP_TX_PILOT_FREQUENCY = 0x2107
    SI4713_PROP_TX_ACOMP_ENABLE = 0x2200
    SI4713_PROP_TX_ACOMP_THRESHOLD = 0x2201
    SI4713_PROP_TX_ATTACK_TIME = 0x2202
    SI4713_PROP_TX_RELEASE_TIME = 0x2203
    SI4713_PROP_TX_ACOMP_GAIN = 0x2204
    SI4713_PROP_TX_LIMITER_RELEASE_TIME = 0x2205
    SI4713_PROP_TX_ASQ_INTERRUPT_SOURCE = 0x2300
    SI4713_PROP_TX_ASQ_LEVEL_LOW = 0x2301
    SI4713_PROP_TX_ASQ_DURATION_LOW = 0x2302
    SI4713_PROP_TX_ASQ_LEVEL_HIGH = 0x2303
    SI4713_PROP_TX_ASQ_DURATION_HIGH = 0x2304

    SI4713_PROP_TX_RDS_INTERRUPT_SOURCE = 0x2C00
    SI4713_PROP_TX_RDS_PI = 0x2C01
    SI4713_PROP_TX_RDS_PS_MIX = 0x2C02
    SI4713_PROP_TX_RDS_PS_MISC = 0x2C03
    SI4713_PROP_TX_RDS_PS_REPEAT_COUNT = 0x2C04
    SI4713_PROP_TX_RDS_MESSAGE_COUNT = 0x2C05
    SI4713_PROP_TX_RDS_PS_AF = 0x2C06
    SI4713_PROP_TX_RDS_FIFO_SIZE = 0x2C07

    def __init__(self, resetpin = 4, addr = None, busnum = -1, debug = False):
        self._rst = resetpin

        if addr is None:
            addr = self.SI4710_ADDR1

        self._addr = addr
        self._busnum = busnum
        self._debug = debug

        self.currFreq = 0
        self.currdBuV = 0
        self.currAntCap = 0
        self.currNoiseLevel = 0
        self.currASQ = 0
        self.currInLevel = 0

        # for restoring state
        self._freq = None
        self._power = None
        self._rdsStation = None
        self._rdsBuffer = None


    def begin(self):

                self.reset()

                self.bus = Adafruit_I2C(self._addr, self._busnum, self._debug)

                self.powerUp()

                if self.getRev() is not 13:
                    return False

                return True

            def restart(self):
                sleep(5)
                if self.begin():

                    # restore values
                    if self._power:
                        self.setTXpower(self._power)

                    if self._freq:
                        self.tuneFM(self._freq)

                    if self._rdsStation or self._rdsBuffer:
                        self.beginRDS()
                        if self._rdsStation:
                            self.setRDSstation(self._rdsStation)
                        if self._rdsBuffer:
                            self.setRDSbuffer(self._rdsBuffer)

                    self.readASQ()
                    self.readTuneStatus()

                else:
                    sleep(5)
                    self.restart()


            def setProperty(self, prop, val):
                args = [0, prop >> 8, prop & 0xFF, val >> 8, val & 0xFF]

                res = self.bus.writeList(self.SI4710_CMD_SET_PROPERTY, args)

                sleep(0.05)
                return res

            def sendCommand(self, cmd, args):
                res = self.bus.writeList(cmd, args)

                sleep(0.05)
                #self.waitStatus()
                return res

            def getStatus(self):
                return self.bus.readU8(self.SI4710_CMD_GET_INT_STATUS)

            def waitStatus(self, waitForByte=0):
                timeout = 50 # about 2.5 seconds
                response = 0
                while True:
                    response = self.getStatus()
                    if response is -1:
                        self.restart()
                        return
                    if waitForByte == 0 and response != 0:
                        break
                    elif response == waitForByte:
                        break
                    sleep(0.05)
                    timeout -= 1
                    if timeout == 0:
                        break

                if timeout > 0:
                    return True
                else:
                    return False

            def reset(self):
                if self._rst > 0:
                    GPIO.setwarnings(False)
                    GPIO.setmode(GPIO.BCM)

                    GPIO.setup(self._rst, GPIO.OUT)

                    # toggle pin
                    GPIO.output(self._rst, GPIO.HIGH)
                    sleep(0.1)
                    GPIO.output(self._rst, GPIO.LOW)
                    sleep(0.1)
                    GPIO.output(self._rst, GPIO.HIGH)
                    sleep(0.2) # give it some time to come back

                    GPIO.cleanup(self._rst)

            def powerUp(self):

                self.sendCommand(self.SI4710_CMD_POWER_UP, [0x12, 0x50])

                self.setProperty(self.SI4713_PROP_REFCLK_FREQ, 32768) # crystal is 32.768
                self.setProperty(self.SI4713_PROP_TX_PREEMPHASIS, 1) # 50uS pre-
    emph (europe std)
```

```
        self.setProperty(self.SI4713_PROP_TX_ACOMP_ENABLE, 0x03) # turn on limiter and AGC
        #self.setProperty(self.SI4713_PROP_TX_ACOMP_GAIN, 10) # max gain?

        # aggressive compression
        self.setProperty(self.SI4713_PROP_TX_ACOMP_THRESHOLD, 0x10000-15) # -15 dBFS
        self.setProperty(self.SI4713_PROP_TX_ATTACK_TIME, 0) # 0.5 ms
        self.setProperty(self.SI4713_PROP_TX_RELEASE_TIME, 4) # 1000 ms
        self.setProperty(self.SI4713_PROP_TX_ACOMP_GAIN, 5) # dB

    def powerDown(self):
            #self.sendCommand(self.SI4710_CMD_POWER_DOWN)
            self.SI4710_CMD_POWER_DOWN

    def getRev(self):
        if self.sendCommand(self.SI4710_CMD_GET_REV, [0x00]) is -1:
            self.restart()
            return

        response = self.bus.readList(0x00, 9)
        if response is -1:
            self.restart()
            return

        pn = response[1]
        fw = response[2] << 8 | response[3]
        patch = response[4] << 8 | response[5]
        cmp = response[6] << 8 | response[7]
        chiprev = response[8]

        return pn


    def tuneFM(self, freq):
        self._freq = freq
        res = self.sendCommand(self.SI4710_CMD_TX_TUNE_FREQ, [0, freq >> 8, freq & 0xff])
        if res is -1:
            self.restart()
        else:
            self.waitStatus(0x81)

    def setTXpower(self, power, antcap = 0):
        self._power = power
        res = self.sendCommand(self.SI4710_CMD_TX_TUNE_POWER, [0, 0, power, antcap])
        if res is -1:
            self.restart()

    def readASQ(self):
        res = self.sendCommand(self.SI4710_CMD_TX_ASQ_STATUS, [0x1])
        if res is -1:
            self.restart()
            return

        response = self.bus.readList(0x00, 5)
        if response is -1:
            self.restart()
            return

        self.currASQ = response[1]
        self.currInLevel = response[4]

        if self.currInLevel > 127:
            self.currInLevel -= 256

        #print "ASQ:", hex(self.currASQ), "- InLevel:", self.currInLevel, "dBfs"

    def readTuneStatus(self):
        res = self.sendCommand(self.SI4710_CMD_TX_TUNE_STATUS, [0x1])
        if res == -1:
            self.restart()
            return

        response = self.bus.readList(0x00, 8)
        if response == -1:
            self.restart()
            return

        self.currFreq = response[2] << 8 | response[3]
        self.currdBuV = response[5]
        self.currAntCap = response[6] # antenna capacitance (0-191)
        self.currNoiseLevel = response[7]

        #print "Power:", self.currdBuV, "dBuV - ANTcap:", self.currAntCap, "- Noise level:", self.currNoiseLevel

    def readTuneMeasure(self, freq):
        # check freq is multiple of 50khz
        if freq % 5 != 0:
            freq -= (freq % 5)

        res = self.sendCommand(self.SI4710_CMD_TX_TUNE_MEASURE, [0, freq >> 8, freq & 0xff, 0])
        if res == -1:
            self.restart()
        else:
            self.waitStatus(0x81)

    def beginRDS(self):
        # 66.25KHz (default is 68.25)
        self.setProperty(self.SI4713_PROP_TX_AUDIO_DEVIATION, 6625)
        # 2KHz (default)
        self.setProperty(self.SI4713_PROP_TX_RDS_DEVIATION, 200)

        # RDS IRQ
        self.setProperty(self.SI4713_PROP_TX_RDS_INTERRUPT_SOURCE, 0x0001)
        # program identifier
        self.setProperty(self.SI4713_PROP_TX_RDS_PI, 0x40A7)
        # 50% mix (default)
        self.setProperty(self.SI4713_PROP_TX_RDS_PS_MIX, 0x03)
        # RDSD0 & RDSMS (default)
        self.setProperty(self.SI4713_PROP_TX_RDS_PS_MISC, 0x1808)
        # 3 repeats (default)
        self.setProperty(self.SI4713_PROP_TX_RDS_PS_REPEAT_COUNT, 3)

        self.setProperty(self.SI4713_PROP_TX_RDS_MESSAGE_COUNT, 1)
        self.setProperty(self.SI4713_PROP_TX_RDS_PS_AF, 0xE0E0) # no AF
        self.setProperty(self.SI4713_PROP_TX_RDS_FIFO_SIZE, 0)

        self.setProperty(self.SI4713_PROP_TX_COMPONENT_ENABLE, 0x0007)

    def setRDSstation(self, s):
        self._rdsStation = s
        # empty station name to start from
        stationName = [' ',' ',' ',' ', ' ',' ',' ',' ', ' ',' ',' ',' ', ' ',' ',' ',' ']

        # calc number of slots needed
        slots = (len(s)+3) / 4

        i = 0
        for char in s:
            stationName[i] = char
            i += 1

        # cycle through slots
        for i in range(4):
            res = self.sendCommand(self.SI4710_CMD_TX_RDS_PS, [i, ord(stationName[i*4]), ord(stationName[(i*4)+1]), ord(stationName[(i*4)+2]), ord(stationName[(i*4)+3]), 0])
            if res == -1:
                self.restart()
                return

    def setRDSbuffer(self, s):
        self._rdsBuffer = s
        bufferArray = [' ',' ',' ',' ', ' ',' ',' ',' ', ' ',' ',' ',' ', ' ',' ',' ',' ', ' ',' ',' ',' ', ' ',' ',' ',' ', ' ',' ',' ',' ', ' ',' ',' ',' ']

        # calc number of slots needed
        slots = (len(s)+3) / 4

        i = 0
        for char in s:
            bufferArray[i] = char
            i += 1

        # cycle through slots
        for i in range(5):
            if i == 0:
                secondByte = 0x06
            else:
                secondByte = 0x04

            res = self.sendCommand(self.SI4710_CMD_TX_RDS_BUFF, [secondByte, 0x20, i, ord(bufferArray[i*4]), ord(bufferArray[(i*4)+1]), ord(bufferArray[(i*4)+2]), ord(bufferArray[(i*4)+3]), 0])
            if res == -1:
                self.restart()
                return

    def setGPIOctrl(self, x):
        self.sendCommand(self.SI4710_CMD_GPO_CTL, [x])

    def setGPIO(self, x):
                self.sendCommand(self.SI4710_CMD_GPO_SET, [x])

#--------------------Begin unique code----------------------#
GPIO.setwarnings(False)#Removes warnings from GPIO channel

GPIO.setmode(GPIO.BCM)#Sets pinout to BCM
LightSensorChannel = 17 #Input channel for the light sensor
ONDUTYCYCLE = 5 #Defines servo duty cycle when On
OFFDUTYCYCLE = 80 #Defines servo duty cycle when OFf
```

```
        ServoChannel = 18 #Output channel for servo control signal

        GPIO.setup(LightSensorChannel, GPIO.IN)#Configures GPIO as Input
        GPIO.setup(ServoChannel, GPIO.OUT)#Condifugred GPIO as Output

        pwm = GPIO.PWM(ServoChannel, 50) #Defines a PWM signal and sets frequency to 50Hz
        pwm.start(OFFDUTYCYCLE) #Sets initial duty cycle

        startTime = time.time() #Grabs initial start time
        print("Servo Control Started")

        if __name__ == '__main__':

            FMSTATION = 9200 #Sets the fm transmit frequency. 9200 = 92.0MHz 10210 = 102.1Mhz
            POWER = 115 #Sets output power of SI4713 in dBuV, 88-115 acceptable
            #With two stage amplifier, 88 results in 14dbm output, 98 is 24dbm.
            #Output amplifier is rated to 24dbm without distortion, higher values
            #can lead to damage

            processid = 0 #Initializes a variable to store PID of a process
            powerstate = 0 #Initializes a variable to determine power state

            radio = Adafruit_Si4713() #Defines the fm transmitter

            while True: #Infinite loop
                #currentTime = time.time()
                sleep(0.02) #Keep code running no more than 50 times a second
                if (GPIO.input(LightSensorChannel) == True): #Checks if lights are on
                            #print("Processes activated")
                                pwm.ChangeDutyCycle(ONDUTYCYCLE) #Sets servo to on duty cycle value
                                if (powerstate == 0): #If this is the first loop since power has been activated, do:
                                        print("Powering on accessories") #Debug output for terminal
                                        process = subprocess.Popen("/home/pi/Airspew/playfiles.sh", stdout=subprocess.PIPE, shell=True, preexec_fn=os.setsid)
                                        #Above line spawns a new process that starts playing audio
                                        powerstate = 1 #Change power state variable so next loop knows
                                        processid = os.getpgid(process.pid) #Stores the PID of the audio process
                    #os.system("bash /home/pi/Airspew/playfiles.sh")

                                        if not radio.begin(): #Checks the fm transmitter connection
                                                print("error! couldn't begin!")
                                                pass
                                        else: #If transmitter initializes, set the power, station, and transmit RDS
                                                print("turning on fm tx")
                                                radio.readTuneMeasure(FMSTATION)
                                                radio.readTuneStatus()

                                                radio.setTXpower(POWER)
                                                radio.tuneFM(FMSTATION)

                                                radio.beginRDS()

                                                radio.setRDSstation("SCU      ")
                            radio.setRDSbuffer("Airspew!")

            else:
                    #print("Processes inactive") #Debug output for terminal
                if (powerstate == 1): #If this is first loop since deactivation, do:
                        powerstate = 0 #Change power variable
                        print("killing servo and radio")
                        pwm.ChangeDutyCycle(OFFDUTYCYCLE) #turn off servo
                        radio.powerDown() #kill transmitter
                        if (processid != 0): #If a process was spawned, do:
                            print("killing player")
                            os.killpg(processid, signal.SIGTERM)#kill audio process
                        sleep(0.02) #Sleep to give input pin a chance to settle
```