The Trailing Edge

October 2025

Weapons Systems for Your Homebuilt

Before the turn of the century, back in the 1980s, my first US Air Force assignment was at Eglin AFB. I was a young test engineer working in Aircraft/Stores Compatibility. We ran test missions to determine if weapons could be carried safely in specified configurations aboard combat aircraft. We also tested if those weapons could be safely dropped or launched without striking the aircraft, and what the ballistics were of the falling weapon. I ran missions with F-4, F-111, F-15, F-16, and A-10 aircraft.

Each of these aircraft had a stick grip with various buttons and switches on it. Under the pilot's index finger was the trigger, which was used to shoot the gun or a missile. None of my missions involved using the trigger. At the top of the grip was a hat switch which activated the pitch and roll trim. Next to the hat switch, convenient to the pilot's thumb, was the bomb release button (commonly called the pickle button, which apparently refers back to claims that the Norden bombsight could drop a bomb in a pickle barrel). The radio push-to-talk (PTT) switch was located on the throttle to be activated by the left thumb or index finger.

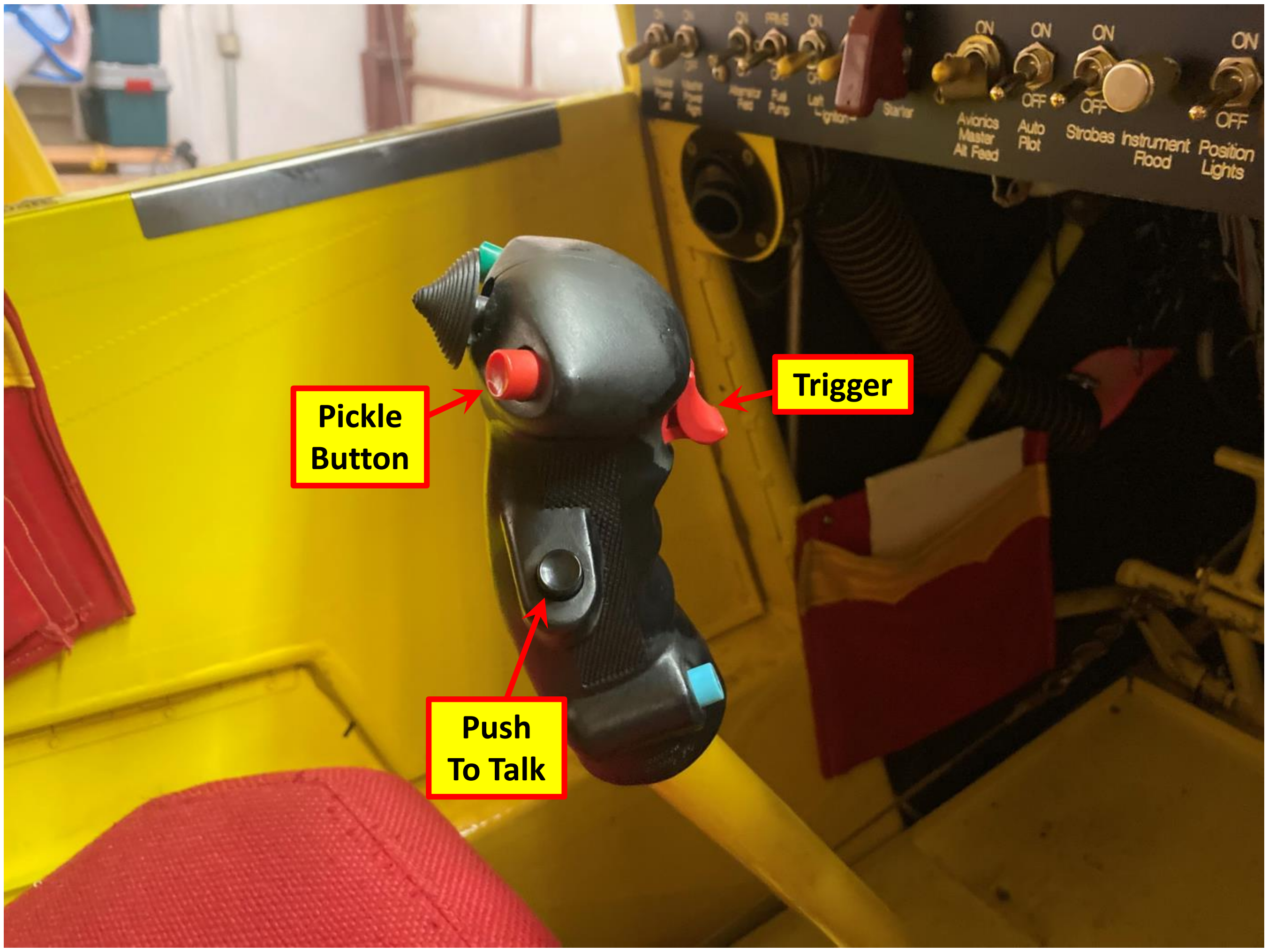
Most importantly, each aircraft had a "Master Arm" switch. If the Master Arm switch was in "SAFE" (the normal position), you could push the pickle button or pull the trigger and nothing would happen. If the Master Arm switch was in "SIM" (for "simulate"), the Head Up Display (HUD) or gunsight would show all of the symbology necessary to attack a target, but pushing the pickle button or pulling the trigger would have no effect. However, if the Master Arm switch was in "ARM" then the weapons system would function as designed, releasing the bomb or firing the gun or missile.

As you can imagine, one of my most important tasks during a test mission was to establish the current position of the Master Arm switch. During the practice passes (called "dry passes"), the Master Arm switch needed to be in "SIM". For an actual weapons release ("hot pass"), the Master Arm switch needed to be in "ARM". At any other time, the Master Arm switch needed to be in "SAFE". Clearly, this was very important, and part of my job was to back up the pilots by listening for the position of the Master Arm switch or querying them for its position. This was critical, not only to not waste any munitions, but to keep everyone safe.

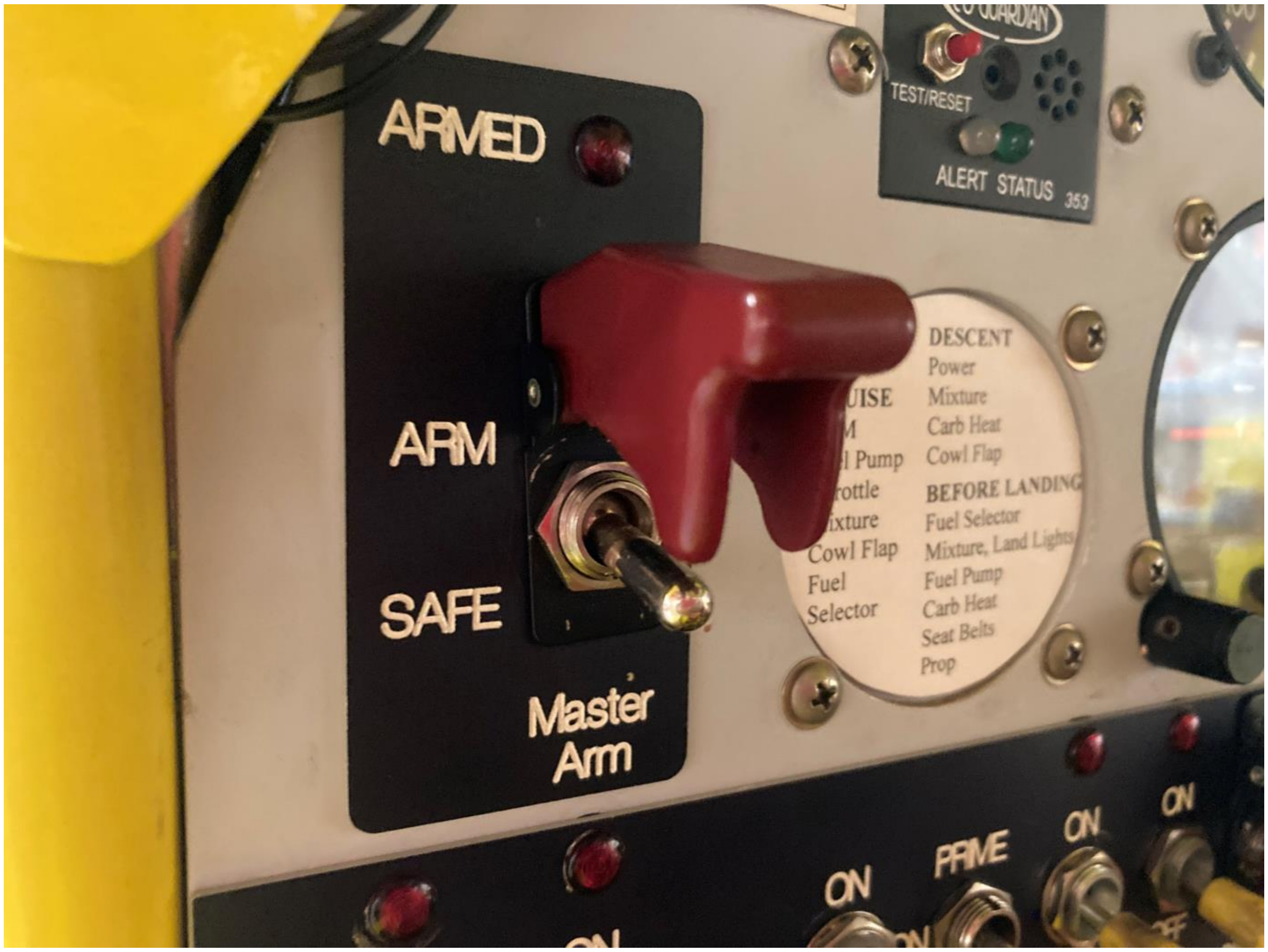
Planning My Stick Grip and Panel

At some point after I started building my Bearhawk in 1996, I needed to start thinking about how I was going to set up my cockpit. For the control stick grips, I decided to go with the Infinity Aerospace stick grip, which was a copy of the F-4 stick grip, reduced slightly in size to fit most people's hand better, including mine. The left stick has a left-handed grip, and the right stick has a right-handed grip. I highly recommend these stick grips.

The Infinity Aerospace stick grip is festooned with lots of buttons and switches. Though I would eventually find a meaningful purpose for each of the switches, initially all I required was a radio PTT switch. Pitch trim in the Bearhawk is a manual wheel in the cockpit ceiling. While many people have done it, because of my experience detailed above, I could not bring myself to use the trigger or the pickle button as the PTT switch. I didn't want to feel like I was firing a gun or dropping a bomb each time I talked to tower or anyone else. Fortunately, the F-4 stick grip and the Infinity Aerospace stick grip have an additional push button on the side of the stick grip. In the F-4, this switch was used to release the aerial refueling boom. I chose to use this push button as my PTT switch.



As an homage to my experience in testing airborne weapons, I decided that my panel should have the all-important Master Arm switch. This would be a red guarded switch, properly labeled, but its only real function would be to turn on a red LED to show that the "system" was "armed". It would be just for fun and something that might generate questions at the fly-in. But then….



My Brain Suddenly Started Running Without My Permission…

I started thinking that I have a Master Arm switch, a trigger, a pickle button, and a music input on the GMA 340 Audio Panel. Why not wire them together such that pulling the trigger would play the sound of a machine gun in the headset? And pushing the pickle button would play the sound of a bomb falling and exploding in the headset? The possibilities were just too good to pass up!

Imagine that guy who cuts you off in the pattern, or worse yet, cuts in front of you over RIPON on that long approach into AirVenture. You reach up, flip the Master Arm switch to "ARM", draw a bead on his tail and pull the trigger. BRRRRPPT!! You've addressed your frustration in an imaginary way and nobody really gets hurt. (This may or may not have actually occurred in the past.)

Alternatively, it's Walter Mitty day, and you are in your imaginary fighter-bomber, flying low level into enemy territory (complying with all 14CFR requirements for minimum altitudes and distance from obstacles, of course!) to interdict the supply shipment train carrying ball bearings and aircraft engines to the secret aircraft factory. You spot the train and sneak up behind it. Master Arm to "ARM". As your pipper passes over the ammunition car, you press the pickle button. The bomb releases, screams down toward the train, and explodes, destroying the ammunition car and derailing the remainder of the train into a giant tangled mess. You pull up, do a victory roll (assuming your homebuilt is capable of that) and dash away from the enemy fighters to get back safely to your home base.

After all, I'm told that the original designation for the design that would become the Bearhawk was RB-4. Most people assumed this was from Robert Barrows design number 4, but who's to say in our Walter Mitty world it wasn't Reconnaissance Bomber 4?

Sound Files

For this to work, I needed to find some suitable sound files. Searching the internet, I found a file listed as an M60A machine gun (m60a.wav). It was firing at 10 rounds per second, or 600 rounds per minute. According to Wikipedia, an AN/M2 Browning fires at 750-850 rounds per minute, which would be around 12-14 rounds per second. Close enough. It was only one gun firing instead of six, but how many do you think a Bearhawk could carry? I trimmed the sound file to just under one second (ten rounds). This kept the file small so that it would load quickly. I made sure it was exactly 10 rounds, such that when played in a loop it was indistinguishable when the file started over.

A real bomb falling through the air does not make any significant sound. For the effect to work, I went looking for a sound clip like Hollywood would use of a bomb whistling through the air, followed by an explosion. I found one (bombdrop.wav) with a 3.85 second time of fall, followed by a 1.15 second explosion. The pedant in me calculated what the equivalent release height would be for these numbers, which yielded an unrealistically low 215 feet. At that altitude, you might be within the frag pattern of your own bomb, especially if you hit the enemy ammo dump. On the other hand, a more realistic release altitude of 5000 feet would give a time of fall of 18 seconds, which is way too long to wait for the joke. As it is, this sound file seems to be just the right length.

Original Implementation

My original implementation used two sound modules from Radio Shack. I recorded the sound files on to the modules. When power was applied to the sound module, it would play back the sound, looping continuously until the power was removed. Output was through a 3.5mm audio jack into an audio mixer, which combined the weapons sounds and cockpit music into the music input of the GMA 340 audio panel. The audio mixer was powered by 12 volt power from the main power bus, and automatically powered up when the Master Power switches were turned on.

The trigger switch applied power to the machine gun module. The sound played for as long as the trigger was pulled. Generally, the joke ran out before the ammunition did.

The pickle button applied power to the bomb module. In the F-4, the pickle button only had to be pressed momentarily to initiate the bomb release. In this implementation, the pickle button had to be held down until the bomb finished exploding. If the pickle button continued to be held down, another bomb would release. It was a compromise, but it worked.

Eventually, something went wrong with the bomb sound module and the bombs failed to fall. With the demise of Radio Shack, there were no replacement sound modules available. Thus, an alternative solution was needed.

Enter the Raspberry Pi

The Raspberry Pi is a low cost, single board computer that was created to be affordable to promote teaching basic computer science in schools. Besides just running programs, a significant feature of this computer is a general purpose input-output (GPIO) bus, which allows the use of switches to affect the program execution. The software was written in Python. Being able to write custom software offered the ability to fix the problem with the pickle button functionality, specifically changing it so only a momentary press was required to start the bomb sound playing.

In operation, turning on the aircraft Master Power sends power to the Raspberry Pi through a micro-USB cable. Applying power wakes up the Raspberry Pi and starts the boot up process, which finishes with loading and executing the program. The boot up process takes about a minute, which is unnoticeable when starting the airplane for flight. However, you will need to wait for a minute if firing up the system to demonstrate the "weapons systems" to someone on the ground.

The software loads the required libraries, and then enters an infinite loop, constantly checking for a switch press from either the trigger or the pickle button. The trigger and pickle button are wired through the Master Arm switch, such that the weapons systems can only be "activated" with the Master Arm switch in the "ARM" position. A second pole on the Master Arm switch lights an LED to show the system is "armed".

Pulling the trigger connects pin 38 of the P1 connector through the trigger and Master Arm switch to ground on pin 39. When the program determines that the trigger has been pulled, the program enters a branch that loads and plays the sound file m60a.wav continuously until the trigger is released.

Pressing the pickle button connects pin 40 of the P1 connector through the pickle button and Master Arm switch to ground on pin 39. When the program determines that the pickle button has been pressed, the program enters a branch that loads and plays the sound file bombdrop.wav once from start to finish. The sound will continue to play to completion even if the pickle button is released.

In the initial version of the software, only one detection of grounding pin 40 was required to start the bomb drop sound. This worked fine in the clean power of the development environment (i.e. on my desk), but the electromagnetic environment of the aircraft is much less clean. Stray electrical noise would cause pin 40 of the P1 connector to momentarily appear grounded, even though the Master Arm switch was in "SAFE", and this would be enough to be detected by the software. This resulted in random uncommanded bomb releases all around the local flying area. The Skypark Association at the airport where I am based was getting tired of repairing imaginary holes in the runway, not to mention the complaints from all the imaginary destroyed cars in the grocery store parking lot.

Because the Raspberry Pi was installed in the airplane and not easy to remove, I first tried a hardware solution to the electrical noise. I connected a 200 pF capacitor between P1 pin 40 and pin 39. While this did reduce the number of uncommanded releases, it did not eliminate them. The solution was to remove the Raspberry Pi at the next condition inspection and take it back to the software development lab. There, the software was updated to "debounce" the pickle button, a common requirement for switches installed in vehicles or aircraft to address this very problem. With this change, the first detection of a grounding on P1 pin 40 starts a delay loop of 5000 passes, which nominally lasts for half a second. On each pass through the loop, the software checks if P1 pin 40 is still grounded. If P1 pin 40 is not grounded, the debounce loop exits and returns to the main polling loop. If the debounce loop completes 5000 iterations, then it is highly probable that the pickle button was actually pressed and the bomb drop sound plays.

If you are interested in acquiring copies of the Python code and the sound files, send an email to my boss, **Evil Editor Zurg**, at eez@pobox.com and he will see that they are sent to you.

Hardware

I am by no means an expert on Raspberry Pi computers. In fact, so far, I have learned just enough to make this project work. There are certainly other configurations that will work, but I will describe what I did. I bought my parts from Amazon, though I am sure there are other sources. Prices quoted were current in March 2024.

For the board I used an RS Components Raspberry PI 3 B+ ($51.99). This board was mounted in an iUniker Raspberry Pi 3 Model B+ Transparent Case ($6.99). This box conveniently protects the computer board while allowing access to all of the ports. The box also comes with two heat sinks, one for the CPU and one for the Ethernet controller. These heat sinks are not required, but since they were included, I installed them.

The operating system and file storage are on a micro SD card. One popular operating system installation is NOOBS (New Out Of Box Software). I went the easy route and purchased a Raspberry Pi 32GB Preloaded (NOOBS) Micro SD Card ($9.99).

To set up the Raspberry Pi, insert the NOOBs Micro SD card in the micro SD card slot. Connect a monitor to the HDMI port. Plug a keyboard and a mouse into the USB-A ports. Connect headphones or speakers to the 3.5mm audio jack. Switch wires can be connected to the P1 connector either with individual wires or a ribbon cable. The other ports are not needed for this application and may be left empty. Finally, apply power with a micro USB cable connected to a USB power adapter.

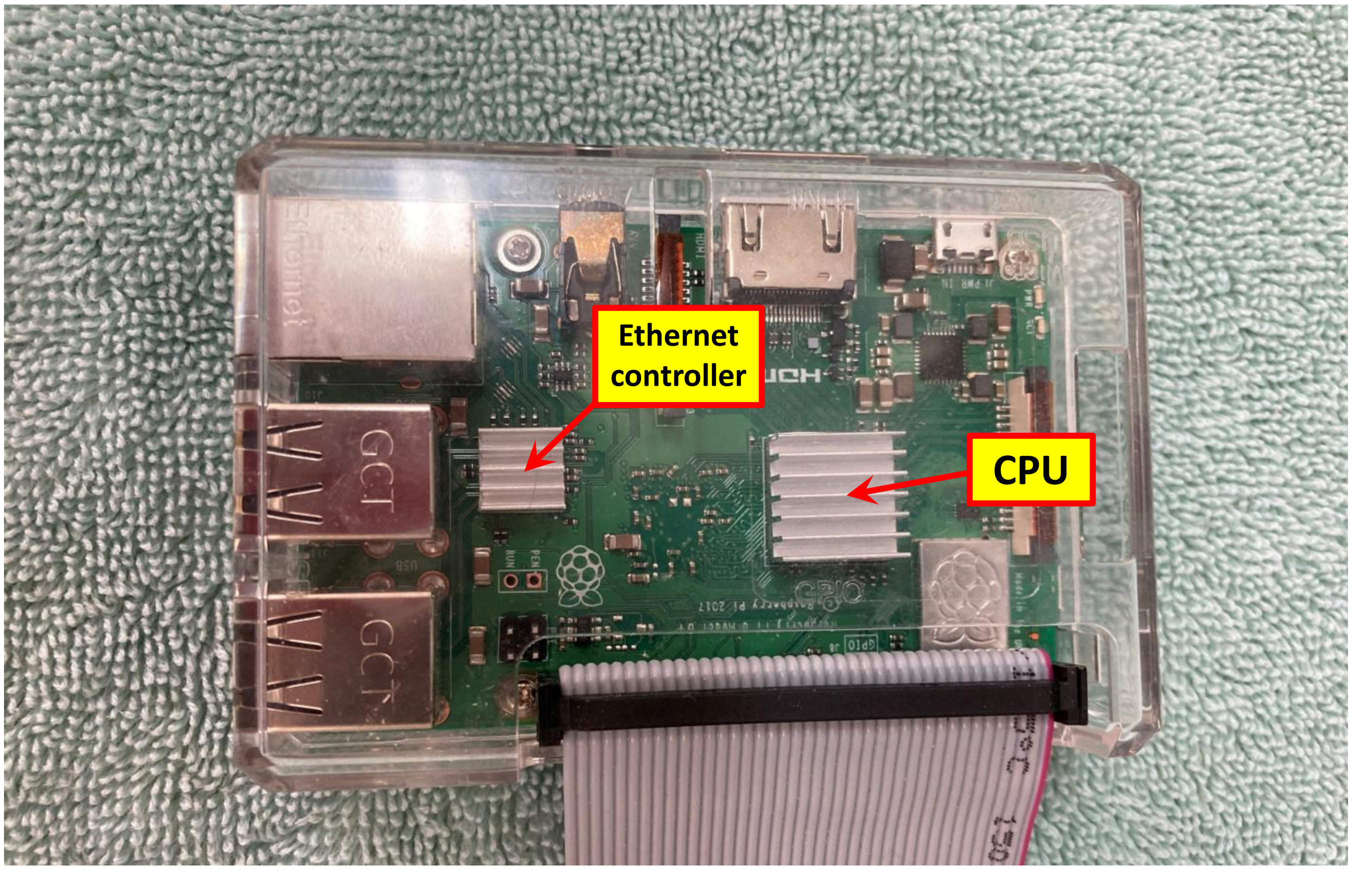
The Raspberry Pi will boot into a desktop GUI interface similar to Windows or Mac. To enter the program, open IDLE (Interactive DeveLopment Environment) on the desktop.

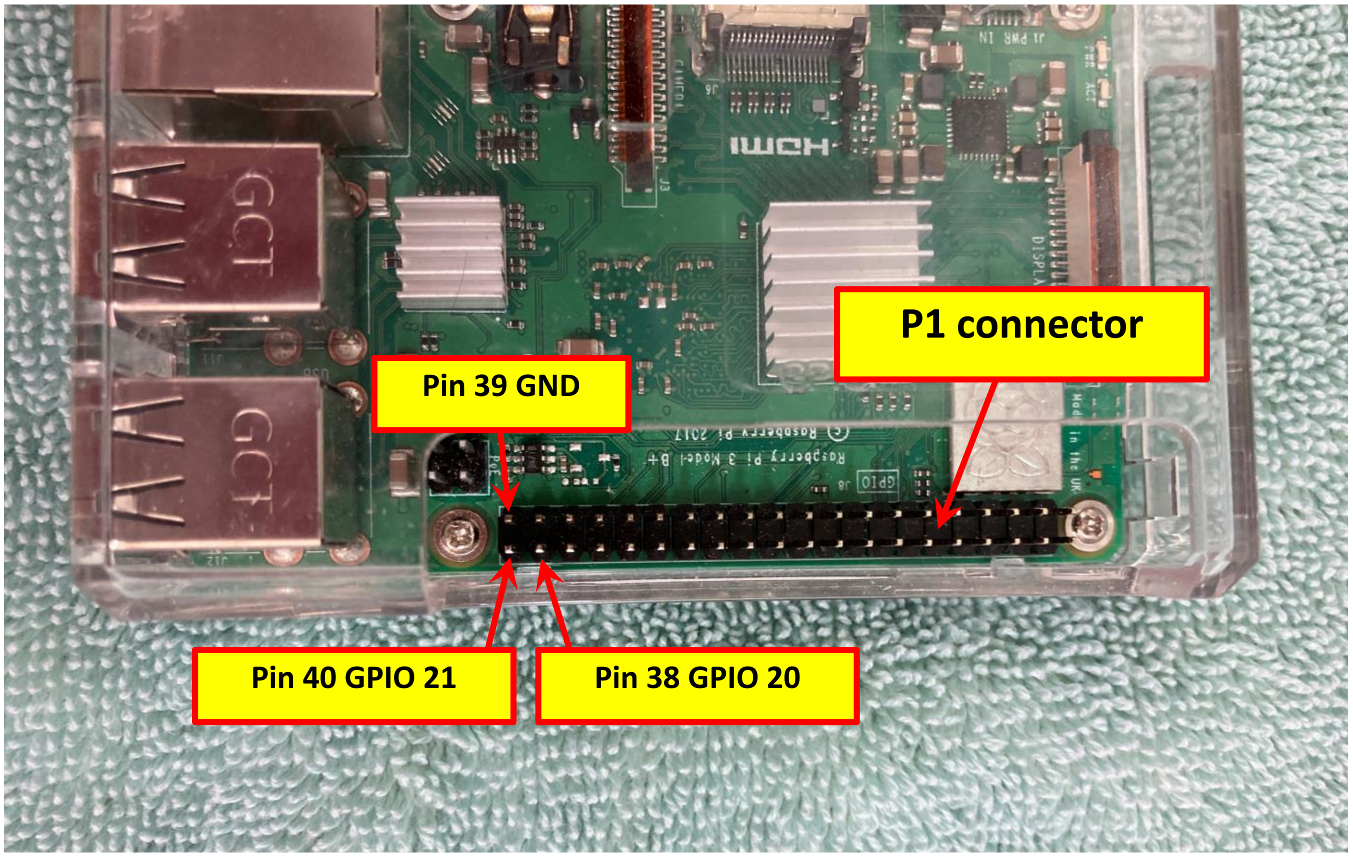
In the airplane, the Raspberry Pi is powered by a micro USB cable plugged into a USB power adapter plugged into a cigarette lighter socket connected to the main power bus. An audio cable is plugged into the 3.5mm audio jack and connected to the audio mixer, which is connected to the music input of the GMA 340 audio panel. The program is set up to autoexecute, so a keyboard and mouse are not required. The lack of a monitor has no effect on running the program.

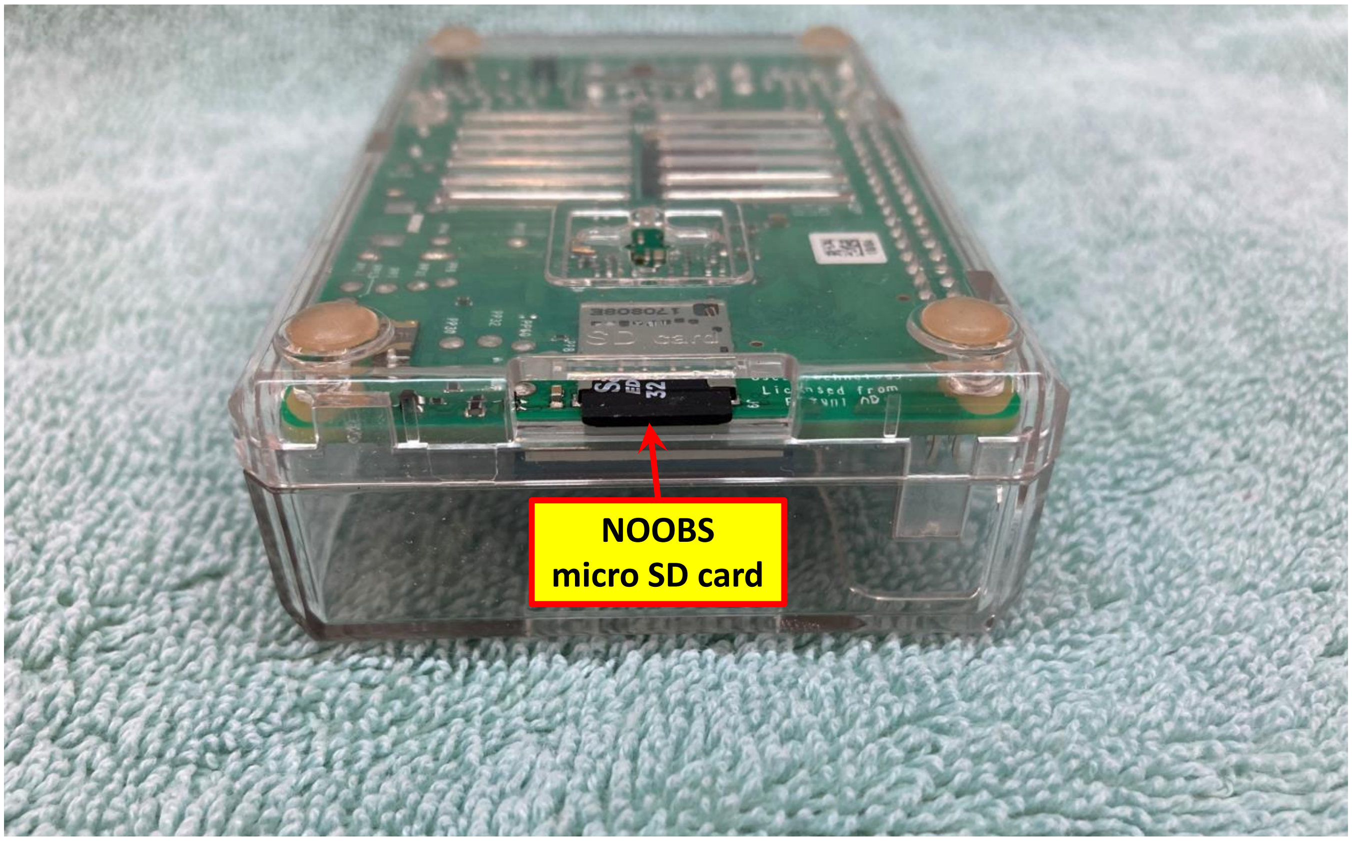
It's Walter Mitty Time!

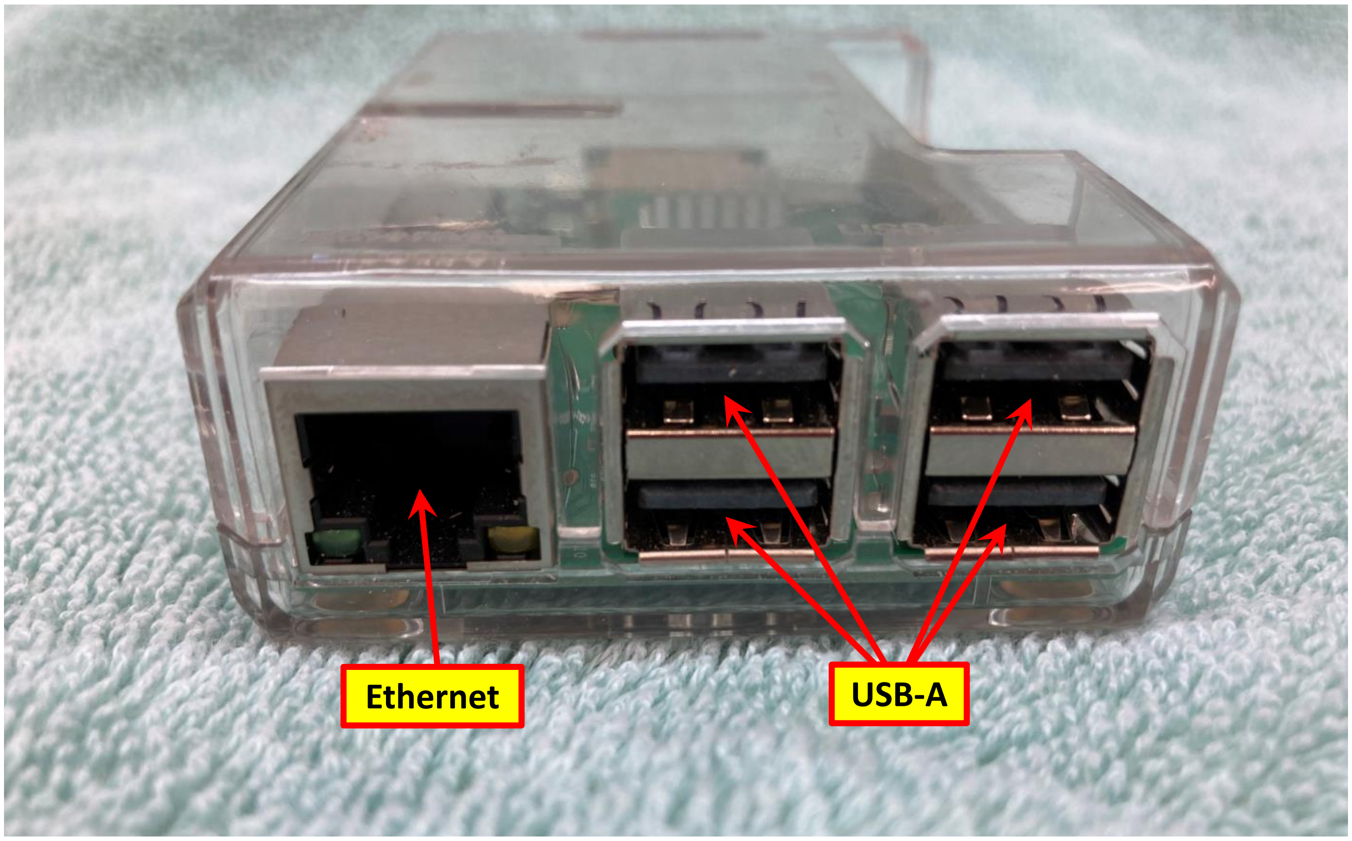
In the opening paragraph of James Thurber's *The Secret Life of Walter Mitty*, Walter Mitty imagines himself as the pilot of an eight-engined Navy hydroplane, struggling to get through the storm. With these weapons systems you can live out your fantasies of reporting back to General Doolittle that you single-handedly fought off the Luftwaffe and destroyed the target. If you're not into such fantasies, perhaps you're like me and enjoy "Things that make you go 'Huh?!'" Looky-loos at the local fly-in are bound to cock their head and ask you what the "Master Arm" switch is for. Passengers always laugh when I flip on the "Master Arm" and invite them to pull the trigger. Usually, they are so surprised that they immediately release the trigger, think for a second, and then pull it again and unload the "whole nine yards". Then I invite them to press the pickle button and wipe out the infidels in the fields below us. Even demonstrating the system on the ground through the headset evokes a similar response. Flying is supposed to be fun, and it can be even more fun when you allow yourself some pure and utter silliness!

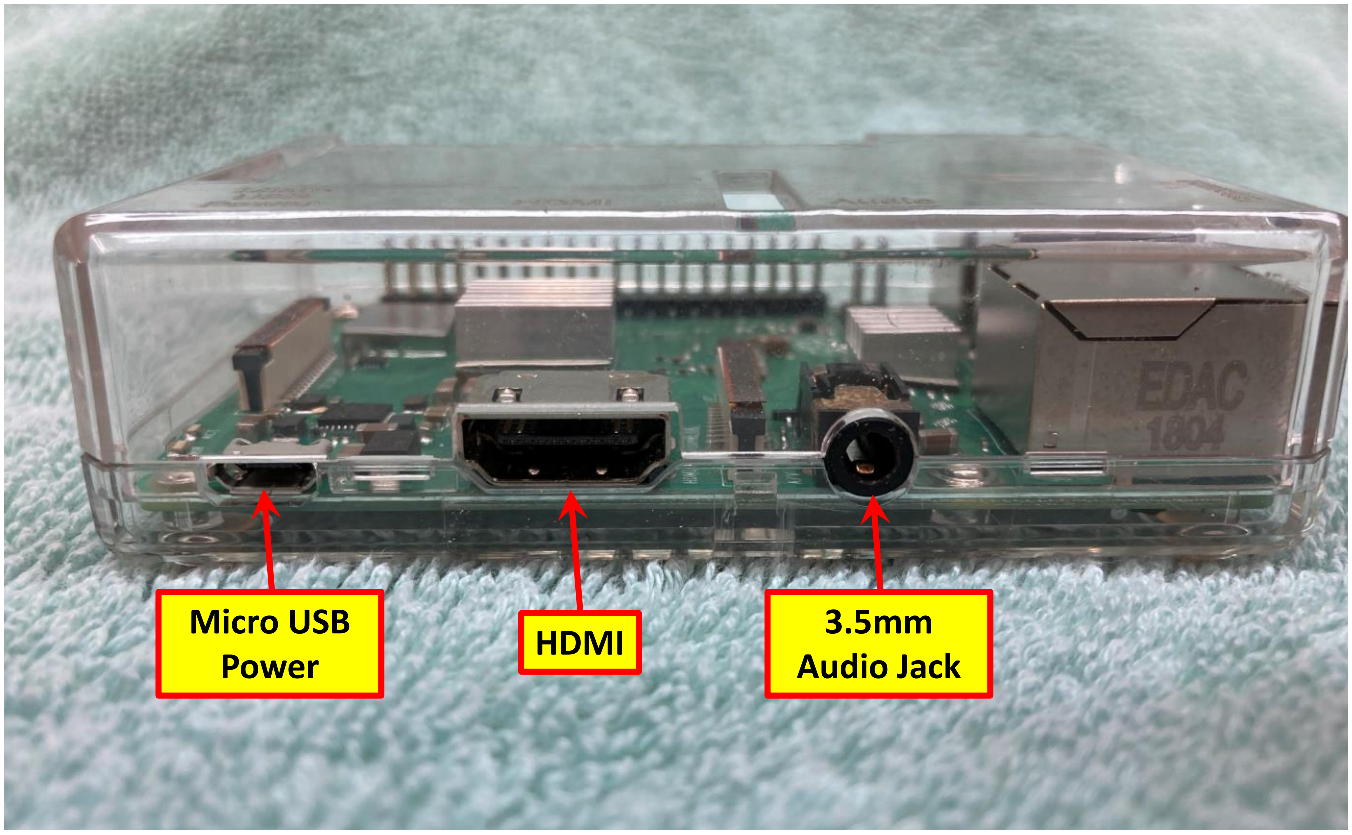
- Russ Erb

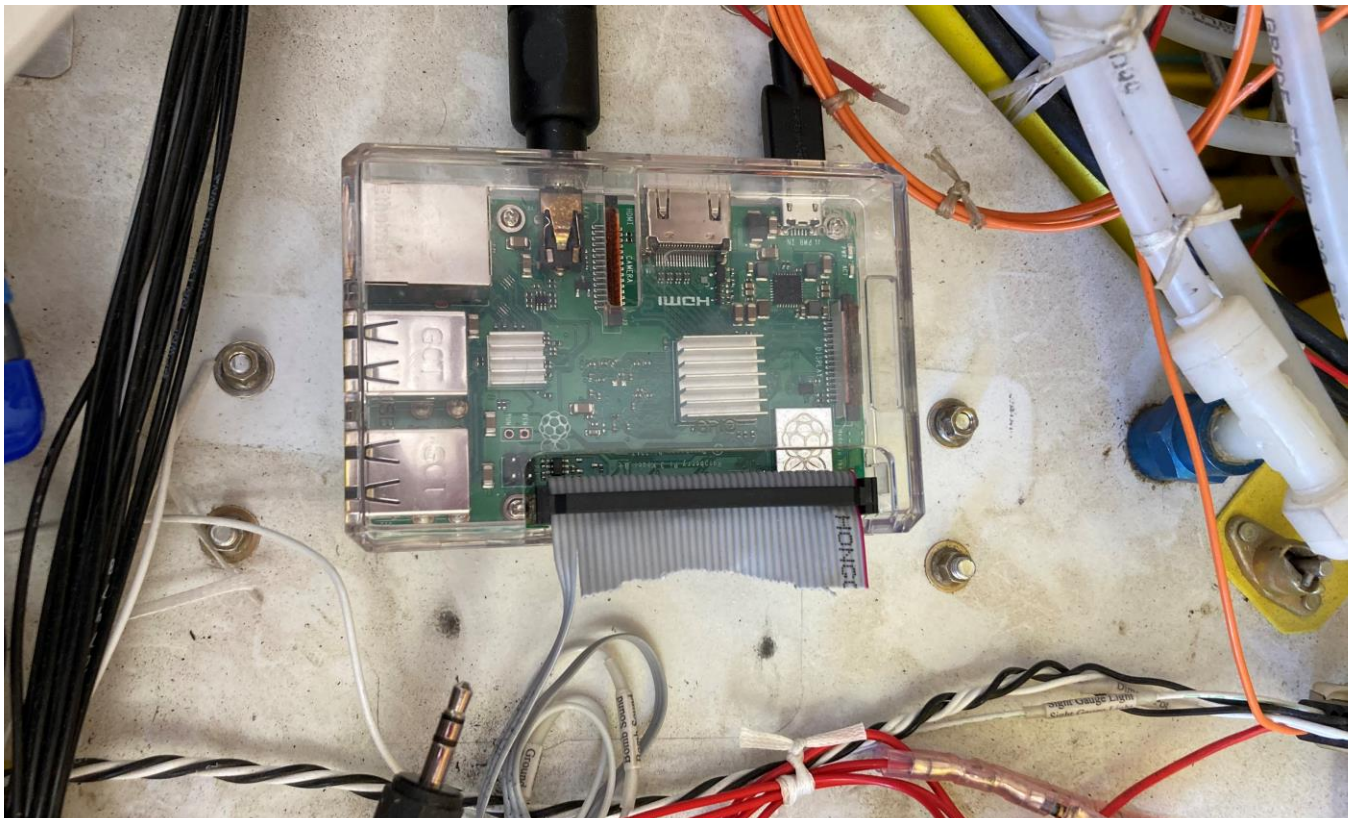

Ethernet controller
CPU


P1 connector
Pin 39 GND
Pin 40 GPIO 21
Pin 38 GPIO 20

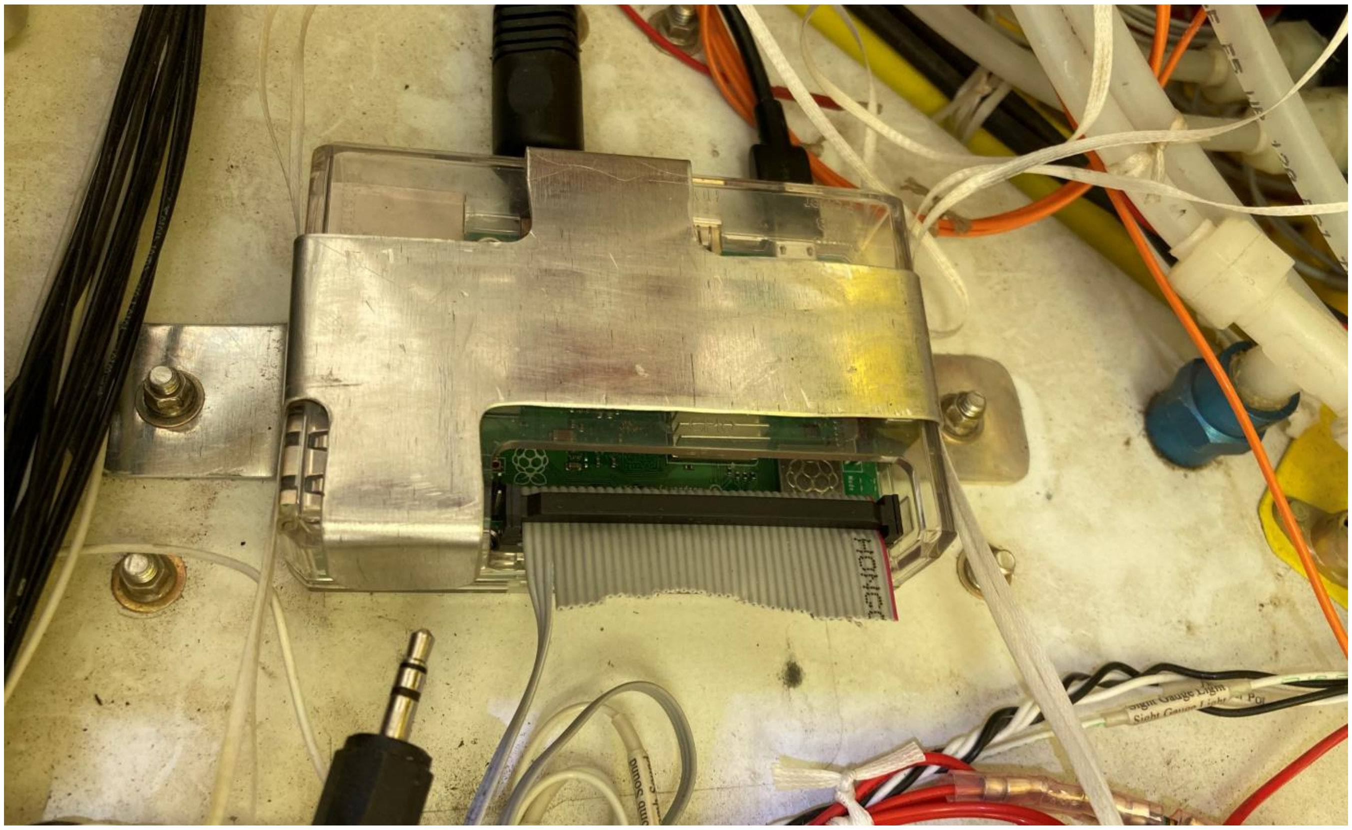

NOOBS
micro SD card

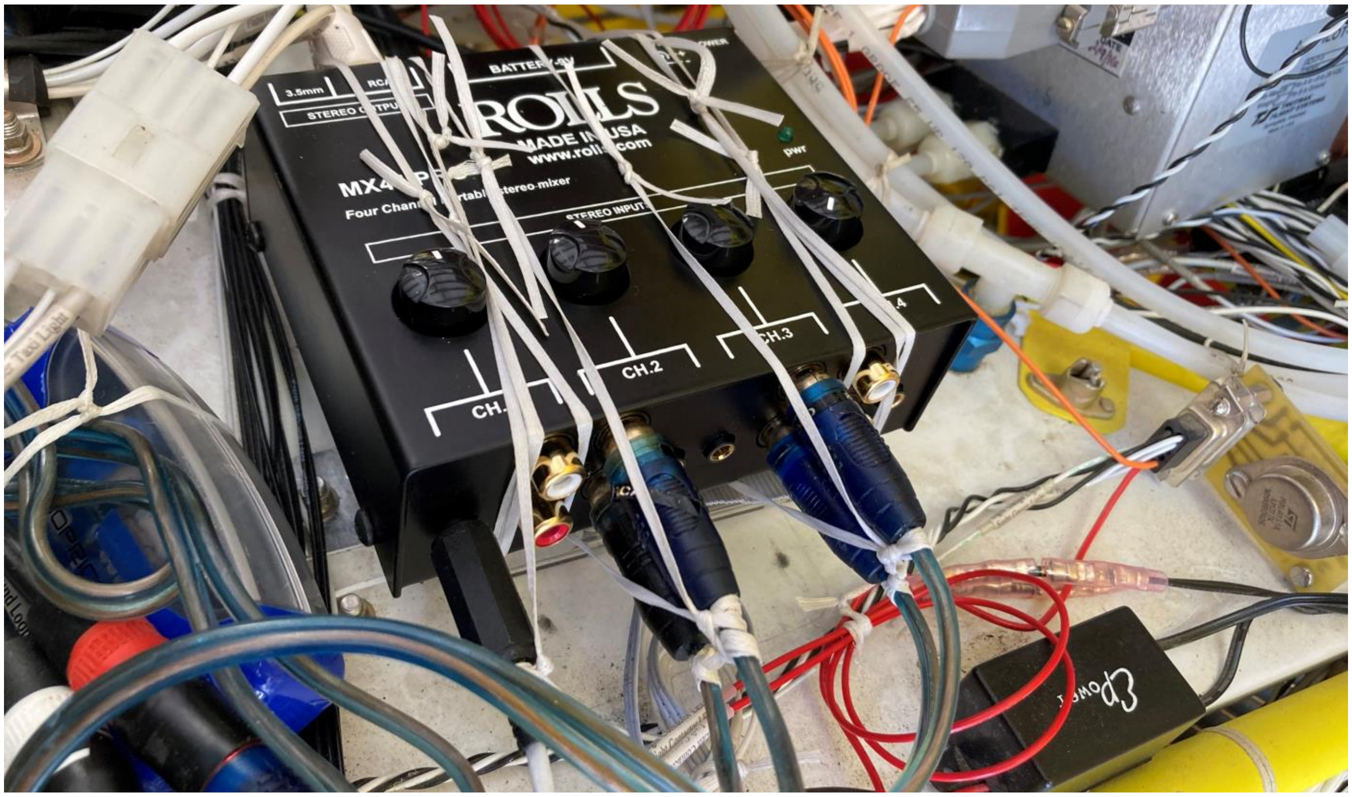

Ethernet
USB-A





Raspberry Pi installed in the aircraft. Unneeded wires of the ribbon cable have been cut away


Bent aluminum bracket holding Raspberry Pi in place



Rolls MX44-PRO audio mixer tied to Raspberry Pi bracket. Avionics lacing cord is used to retain all plugs.

bearhawk_sounds.py Listing

```
# In Python, all text following “#” is a comment and not executable code.  Indents matter.
# Bearhawk sounds guns and bombs on the Raspberry Pi
# Russ Erb, Bearhawk #164 N6786E April 2019
# Bomb sound delay loop added April 2021

import RPi.GPIO as GPIO                     # Load RPi.GPIO library.  This library contains the routines to set up
                                            # and access the GPIO pins on connector P1
import pygame                               # Load pygame library.  This library contains the routines needed to
                                            # load and play sound files

# These print statements show on a monitor to describe the operation of the program.  If no monitor is connected,
# as when installed in the airplane, these statements are sent to the bit bucket without affecting program
# operation.  These print statements can be removed without affecting program operation
print ("This program is intended to autoexecute and run without keyboard, mouse or monitor.")
print ("Sounds play through the 3.5mm headphone jack, not HDMI monitor (set in settings)")
print ("Ground GPIO pin 20 to play guns sound.  Guns play until pin 20 is ungrounded (trigger released)")
print ("Ground GPIO pin 21 momentarily to play bomb sound.  Bomb sound will play in its entirety")
print ("once after a momentary grounding of pin 21.  Pickle button and trigger will be ignored")
print ("while sound is playing.")
print ("Ctrl C to stop program")

GPIO.setmode(GPIO.BCM)                                 # Use Broadcom GPIO numbering

GPIO.setup(20, GPIO.IN, pull_up_down=GPIO.PUD_UP)      # Set pin 20 as input and use the internal pull up
                                                       # resistor to set pin 20 normal state to HIGH (i.e.
                                                       # TRUE).  Setting the pin to HIGH makes the
                                                       # disconnected state reliable.
                                                       # Allowing the pin to float would leave the pin in an
                                                       # uncertain and unpredictable state.  Connecting
                                                       # pin 20 to ground (P1 pin 39) changes the state to
                                                       # FALSE

GPIO.setup(21, GPIO.IN, pull_up_down=GPIO.PUD_UP)      # Same for pin 21

pygame.init()                                          # Initialize pygame library

SOUND_END = pygame.USEREVENT + 1                       # Assign a user defined event named SOUND_END
                                                       # to an unused event number
pygame.mixer.music.set_endevent(SOUND_END)             # Associate SOUND_END event with the end of the
                                                       # sound file
# End of setup routine

# Begin infinite loop to poll for trigger and pickle button position
while True:                                            # Loop will run infinitely

   if not GPIO.input(20):                              # Grounding GPIO pin 20 will change its state to
                                                       # FALSE. “not FALSE” is TRUE, so execution will enter
                                                       # this routine.  GPIO pin 20 is grounded through
                                                       # trigger switch and Master Arm switch.  This routine
                                                       #plays the gun sound.
```

```
        pygame.mixer.music.load('m60a.wav')             # load machine gun sound from file m60a.wav
        pygame.mixer.music.play(-1)                     # Play file m60a.wav.  With argument set to -1, music
                                                        # file will repeat indefinitely until commanded to stop
        while not GPIO.input(20):                       # As long as the trigger remains depressed, stay in this
                                                        # loop
            continue                                    # Dummy statement

        pygame.mixer.music.stop()                       # When the trigger is released,  the while loop will exit
                                                        # and this will command the music file to stop playing
                                                        # Execution returns to the main polling loop
    if not GPIO.input(21):                              # Grounding GPIO pin 21 will change its state to
                                                        # FALSE. “not FALSE” is TRUE, so execution will enter
                                                        # this routine.  GPIO pin 21 is grounded through the
                                                        # pickle button and Master Arm switch.  This routine
                                                        #plays the bomb sound.
        drop_bomb = True                                # Set flag for debounce routine
        for icnt in range(5000):                        # Delay loop to ignore spurious inputs.  Increase range
                                                        # number to increase delay.  5000 is a reasonable
                                                        # start. Test delay in terminal mode.  Response will be
                                                        # slower in the editing environment (IDLE).
                                                        # GPIO pin 21 must remain grounded throughout this
                                                        # for loop (about 0.5 second) for the bomb sound to
                                                        # play. This filters out spikes in the aircraft electrical
                                                        # system.
            print (icnt)                                # Print to nonexistant monitor to slow loop execution
                                                        # by a factor of about 100.  During debugging, number
                                                        # shows how many times the loop executed.
            if GPIO.input(21):                          # For each pass through the delay loop, check if pickle
                                                        # button is still depressed (FALSE). If the result is TRUE,
                                                        # then the pickle button has been released during the
                                                        # delay loop or the loop was entered because of
                                                        #spurious input.
                drop_bomb = False                       # Set flag to stop bomb sound from playing
                print (drop_bomb)                       # Shows that loop stopped while debugging
                break                                   # Stop execution of debounce for loop

        if drop_bomb:                                   # This check follows the debounce loop.  if drop_bomb
                                                        # is still TRUE, then the pickle button was actually
                                                        # pressed. If drop_bomb is FALSE, return to the main
                                                        # polling loop.
            pygame.mixer.music.load('bombdrop.wav')     # Load bomb sound from file bombdrop.wav
            pygame.mixer.music.play(0)                  # Play bomb sound. With argument set to 0, music file
                                                        # will play through only once
            bomb_playing = True                         # Set flag to enter while loop
            while bomb_playing:                         # This loop ties up execution while sound plays,
                                                        # ignoring any trigger or pickle button presses
                for event in pygame.event.get():        # Cycle through pygame events
                    if event.type == SOUND_END:         # Check if an event matches SOUND_END
                        bomb_playing = False            # Upon reaching the end of playing bombsound.wav,
                                                        # set bomb_playing to FALSE which will cause
                                                        # execution to break out of this while loop and return
                                                        #to main polling loop.
#end of main loop
```

Raspberry Pi Setup Notes

To set up the program to automatically execute (autoexec), navigate to the /home/pi directory. Press Ctrl-H to show hidden files. Open the file ".bashrc". At the end of ".bashrc", add the following statements (substitute your actual path, directory, and file name):

```
cd /home/pi/python/bearhawk_sounds
python Bearhawk_sounds.py
```

This will load and run the program.

To get sound out of the 3.5mm headphone jack instead of the HDMI port, open the terminal window. Type in:

```
sudo raspi-config
```

In the configuration program:

1. Select Option 7 Advanced Options and press Enter.
2. Select Option A4 Audio and press Enter.
3. Select Option 1 Force 3.5mm jack and press Enter.
4. Exit the configuration program.