Igloo 23 VTOL Demonstrator

Project Summary & Lessons Learned

Michael Gunderman

Preface

This document is partially to show what I've learned, but it's mostly to pass on that knowledge to the UAV/RC community. The majority of this is talking about the project steps and methodologies, **but the "Tips & Lessons Learned" section has most of the condensed information that I think would be useful to someone building their own VTOL aircraft**, especially with the Pixhawk.

This project was an extension of the solar VTOL aircraft project I worked on for one of my two senior design projects at Miami University. For that project, we built and modified a Gemini V2 aircraft with solar charge capability, vertical-takeoff-or-landing, glowstick drop, and semi-autonomous flight. Due to understaffing, we barely got the aircraft to the beginning of the flight test program. It hovered and flew in forward flight, but many questions were left unanswered.

This is the original aircraft, built from a Bearospace Gemini V2 aircraft for a senior design project. I built the Igloo 23 demonstrator because I only had a 5S battery and wanted to experiment with pixhawk VTOL on a cheap platform.

Moving forward, I wanted to take a closer look and see what I could find out. I decided on a multi-phase approach. Phase I was to do some basic trade studies and research the potential performance of such a system. Phase II was the creation of the Igloo 23 VTOL demonstrator. This "phase II" is the subject of this document. With this, I wanted to gain experience running a VTOL flight-test program using the pixhawk on a low-risk platform and also test new designs for vertical-takeoff-motor pivoting.

Project Summary

First it should be understood that this aircraft was intentionally suboptimal. Nothing about it was high performance *or* high quality. Also, while I have used it a few times in the past, this was by far my most intimate experience with the Pixhawk 4 (Px4) flight controller. In the course of the project, I crashed the airplane ~4 times. **Most of these crashes were caused by improper tuning or configuration of the pixhawk parameters.** I have outlined these mistakes in the "Tips & Lessons Learned" section at the end. On this aircraft, I flew the same VTOL motors from the original aircraft. Because it's a much lighter aircraft (~8 lbs vs ~18 lbs), I used 5S batteries instead of 6S. I did not change the propellers to account for the voltage change. I made fiberglass parts, integrated them with 3d-printed motor mounts, made and tuned a custom pixhawk airframe controls profile, and put together some custom instrumentation. Eventually the aircraft flew pretty well, completing full VTOL transitions and limited autonomous flight. The project ended when a bad RC connection, combined with bad return mode settings, caused the aircraft to loiter over one spot until the battery died. By this point, I had achieved most of my goals with the project, and the cheap Flite-Test-foamboard airframe had deteriorated to the point that was barely capable of flying.

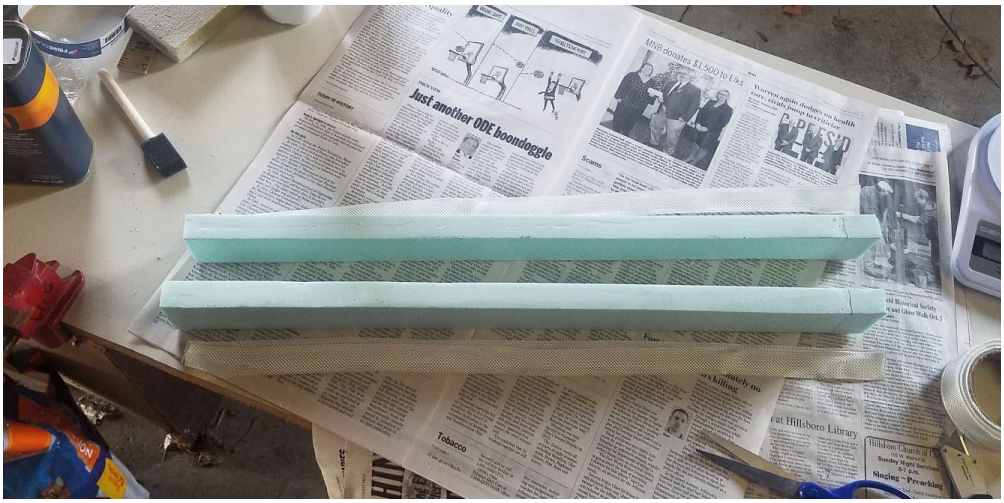
Project Phases

Before starting construction, I needed to validate the aircraft's size and setup. It was based on the "Igloo 23" foamboard airframe, which I had flown before in a lighter configuration. After a quick bench test with the additional weight of the motors required for vertical takeoff, I decided the wing needed some reinforcement because there were early signs of buckling/compression damage on the top of the wing. Suspending the aircraft by its wingtips (simulating the loads of a high-g load case), the wing structure seemed close to its limit. I did not examine the negative-G load case because I figured this aircraft would spend virtually all the time in positive-g flight. My initial estimation of the aircraft's mass suggested it would have a reasonable wing loading.

After validating the configuration, I started prototyping the VTOL arms. The arms on the original aircraft were made of plywood and balsa, and had a very crude system for mounting the

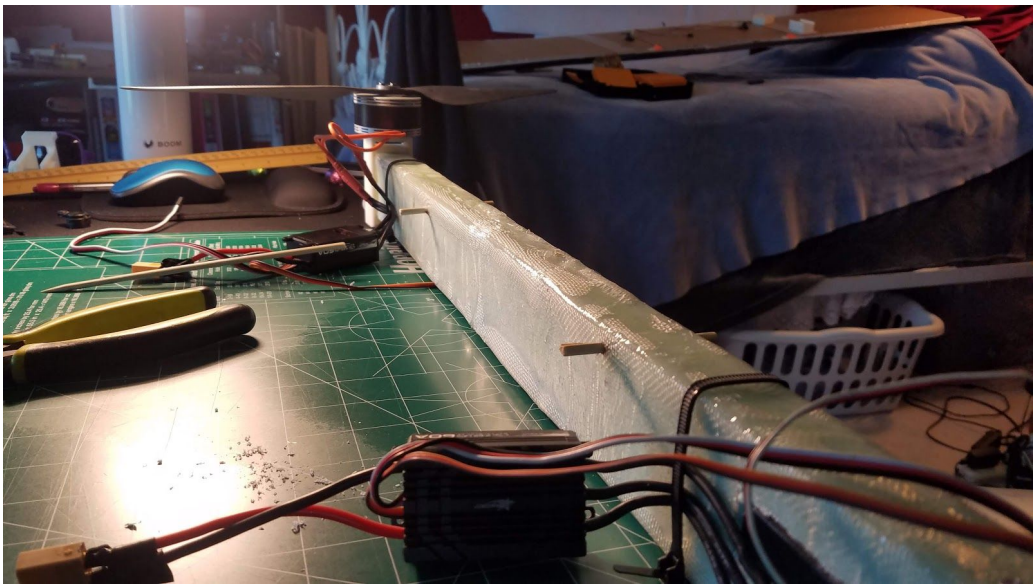
motors. During the hover testing of that much larger aircraft, the motor mounts broke. I wanted to see if I could come up with a more elegant solution. There were a few major challenges. For reasons I won't discuss here, I went with a fiberglass structure and 3d-printed motor mountings. I would have preferred aluminum mountings, but I didn't have the necessary tools. My initial attempts at making fiberglass were unsuccessful because of incompatibility between polyurethane foam and polyester resin. After switching to epoxy resin and refining my process, I put together two working structural arms.

These parts were set up with 2 layers of 10 oz fiberglass tape on both the top and bottom of the foam core. Then I placed the part on its side and applied 6 oz plain cloth on the sides. This cloth reached up and covered the "top" and "bottom" surfaces as well. To maintain shape during the curing process, it turns out, you can use kitchen plastic wrap (cellophane). This precludes you from wiping off much of the excess resin, but the resultant form of the fiberglass layering is improved. After applying glass to both sides (to act as a "shear web"), the net result was 4 layers of glass on top/bottom and 1 layer on each side. If this were an I-beam, these would be the shear flange and shear web, respectively.



In testing, one of these arms supported 35 lbf from each motor mount (this was actually a failed attempt at a distructive test). Under 6S power, my motors are capable of ~10 lbf of thrust each, so this test displayed very conservative performance. It's not clear how close to failure this part was, but the data will be useful for future designs. Under load, my estimations show that it was supporting ~3000 psi of normal mechanical stress. According to the internet, a typical fiberglass layup could support 9000+ psi. The expected failure mode here is either buckling under compression (due to the normal stress resulting from bending) or a compressive failure at the point supporting the beam under loading (in this case, the 70 lbs at the center). In actuality, the failure mode would probably be a combination of the two. As for the risk of buckling, I have

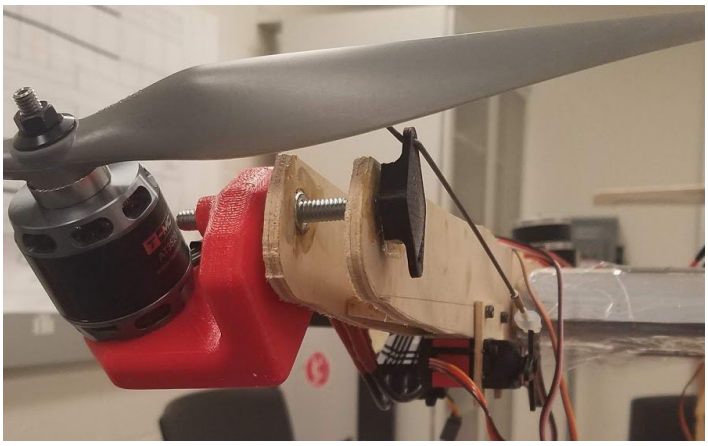
read that store-bought polystyrene foam is hardly stiff enough to add much structural benefit as a composite core material.



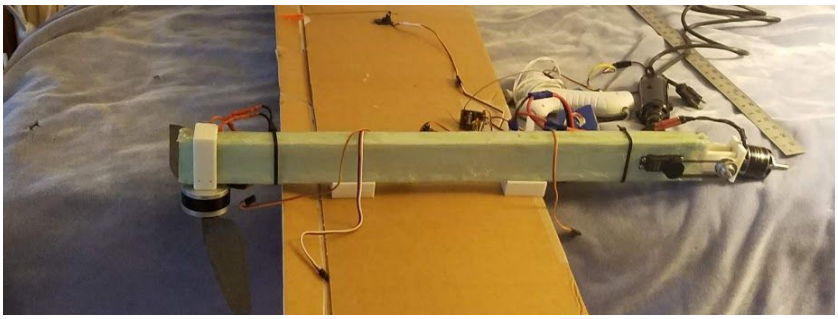
Around this time, I also worked on the mechanism for tilting the forward motors. In some ways this was difficult, but mostly due to my lack of access to useful tooling. Instead of having proper aluminum components, I had to use a 3d-printed structure. Instead of a proper shaft, I used a #4 (I believe) screw. I used GT2 pulleys from robotshop.com. The whole thing was crude but effective. The final result looked something like this:

The printed mounting connected to a printed test jig

Sometimes the belt would slip, but it was mostly during crashes or hard landings.



The original mechanism on the original aircraft. The servo was connected to the pivot axis with a single piano wire connection. This geometry works, but has a fair amount of slop and an inherent nonlinearity.



On the whole, the belt drive works better than the piano-wire connection I used on the original aircraft. There is decreased slop, and the system does not suffer from the same geometric nonlinearities. In retrospect, I doubt there is any advantage to using a belt-drive vs a direct drive system. I would have to figure out how to do that, but there's probably room for improvement here. As a side note, I am intrigued by the use of the aft motor as a counterbalance against the pivoting front motor. This adds weight in some ways, but it would allow a much smaller pivot actuator. I suspect this is used on the stunning WingCopter VTOL drone.

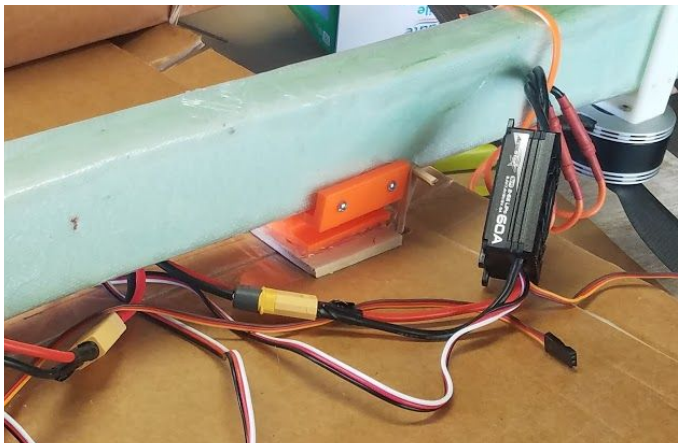
Printed components introduce lots of structural uncertainty, but I was able to mitigate some of this via load testing and good old-fashioned conservative design. Load-testing in the forward direction was relatively trivial, but testing in the hover position required putting together a complete motor arm and then loading both ends.

Load testing the forward motor pivot mount. A couple weights are suspended in the bags, simulating the thrust of the motor in the forward direction.

These fiberglass arms were originally designed for the dimensions of the Gemini v2 aircraft, which has a smaller wing chord than the igloo 23. This meant that on the igloo aircraft, a greater portion of the aft props would be suspended over the wing. This also meant that maintaining a physical separation between the wing and aft props was a challenge. The ailerons' presence makes it worse. To clear the ailerons, I also used a 3d-printed spacer between the wing's bottom surface and the top of the fiberglass arm. To keep the propellers clear of the wing, I redesigned the aft motor mounts to be taller. Then I mounted them at a slight back-lean angle so the forward prop tip clears the top of wing. This all increased drag, but that was of little consequence on this low-performance demonstrator.

In early testing, I started out using only the rubber bands to connect the wing and fuselage. However, there was too much motion and flexibility between the arms and the main

wing. This seemed to cause structural instability and increased the odds of the propellers striking another part of the airframe. To mitigate this, I used hot-glue to connect the 3d-printed wing-arm spacers to the wing. Then I screwed the arms to the spacers. This constrained movement of the arms. The rubber bands helped hold everything together and decrease the load on the hot glue.



It certainly was not a robust setup, but it was good enough for this suboptimal demonstrator. The one advantage was it did a great job of absorbing impacts. It's a good idea to decide what part will break first because if you don't, then the physics of the impact will. Then it could be hard to repair. When this aircraft crashed, it would sometimes impact nose-first. The 3d-printed motor mounts would sometimes break, and so would the wing-arm connection. Some new hot glue, rubber bands, and 3d-printing were usually all that were required to fix it.

Once the aircraft was mostly hardware-ready, it became a problem of addressing the software component. The original aircraft's (gemini v2) software would be sufficient, but I had to get a copy of it from my old group-mate from college. That took well over a week, so I spent some time seeing if I could make my own version of it. Here's what I learned:

Firstly, the explanation on the px4 documentation of how to actually build the px4 flight stack is not well written. It's broken into multiple places, and some of the pages are helpful while others aren't. Here's what you need to know:

To create a custom airframe. We had some difficulty with this, so I'll spend some time on this. Airframes are defined by configuration and mixer files. This is talked about on this page: https://dev.px4.io/v1.9.0/en/airframes/adding_a_new_frame.html I'll try to add some additional commentary to clarify the process for someone who's never seen this before. Firstly, the mixer files and configuration files are essentially just .txt files, but the mixer files use the .mix ending and configuration files do not have a file type designation. The Config file allows you to set the fundamental parameters for the aircraft, while the mixer file tells the flight computer how to actuate the outputs (controls) to make it do what you want. If you're making a custom (especially VTOL) airframe, I would strongly recommend that you check out my parameter lessons learned

in the tips & lessons learned section. Mixer files are considerably more complicated. As explained in the mixer documentation page on the internet, the mixer file contains several blocks of numbers. Each block represents 1 or more PWM channel outputs. The numbers inside this block characterize the source and makeup of the output signal(s). Here I have provided an example from the online documentation of controls block, but I have added some additional explanation.

```
M: 2
O:      10000  10000      0 -10000  10000
S: 0 0  -6000  -6000      0 -10000  10000
S: 0 1   6500   6500      0 -10000  10000
```

Where each number from left to right means:

- **M:** Indicates two scalers for two control inputs. It indicates the number of control inputs the mixer will receive. For example, this is useful in the creation of an “elevon” control where two inputs, the aileron and elevator, contribute to one output (the elevon).
- **O:** Indicates the output scaling (*1 in negative, *1 in positive), offset (zero here), and output range (-1..+1 here).

If you want to invert your PWM signal, the signs of the output scalings have to be changed:

```
O:      -10000  -10000      0 -10000  10000
```

This line can (and should) be omitted completely if it specifies the default scaling:

```
O:      10000  10000   0 -10000  10000
```

- **S:** Indicates the first input scalar for this channel: It takes input from control group #0 (This is Flight Control--See below) and input 0 (roll). It scales the roll control input * 0.6 and reverts the sign (-0.6 becomes -6000 in scaled units). It applies no position offset (0) and outputs to the full range (-1..+1)
- **S:** Indicates the second input scalar: It takes input from control group #0 (Flight Control) and the second input (pitch). It scales the pitch control input * 0.65. It applies no offset (0) and outputs to the full range (-1..+1)

More info on Control Groups can be found at the following link: https://dev.px4.io/v1.9.0/en/concept/mixing.html This is an important part of setting up these control blocks correctly. The distinction between control groups 0 and 1 is a little vague to me, but the listing is as follows:

Control Group #0 (Flight Control)

- 0: roll (-1..1)
- 1: pitch (-1..1)
- 2: yaw (-1..1)
- 3: throttle (0..1 normal range, -1..1 for variable pitch / thrust reversers)
- 4: flaps (-1..1)

- 5: spoilers (-1..1)
- 6: airbrakes (-1..1)
- 7: landing gear (-1..1)

Control Group #1 (Flight Control VTOL/Alternate)

- 0: roll ALT (-1..1)
- 1: pitch ALT (-1..1)
- 2: yaw ALT (-1..1)
- 3: throttle ALT (0..1 normal range, -1..1 for variable pitch / thrust reversers)
- 4: reserved / aux0
- 5: reserved / aux1
- 6: reserved / aux2
- 7: reserved / aux3

Control Group #2 (Gimbal)

- 0: gimbal roll
- 1: gimbal pitch
- 2: gimbal yaw
- 3: gimbal shutter
- 4: reserved
- 5: reserved
- 6: reserved
- 7: reserved (parachute, -1..1)

Control Group #3 (Manual Passthrough)

- 0: RC roll
- 1: RC pitch
- 2: RC yaw
- 3: RC throttle
- 4: RC mode switch
- 5: RC aux1
- 6: RC aux2
- 7: RC aux3

Control Group #6 (First Payload)

- 0: function 0 (default: parachute)
- 1: function 1
- 2: function 2
- 3: function 3
- 4: function 4
- 5: function 5
- 6: function 6
- 7: function 7

The example mixer file for the aircraft(s) we built is provided below:

```
# Mus Quadcopter X-Configuration Mixer

# Motors
R: 4x 10000 10000 10000 0

Tilt mechanism servo mixer
--------------------------
#Left
M: 1
S: 1 4      0  20000 -10000 -10000  10000

CH6: Aileron mixer
------------
M: 2
S: 1 0   10000   10000      0  -10000  10000
S: 0 0   10000   10000      0  -10000  10000

CH7: Elevator mixer
------------
M: 2
S: 1 1  -10000  -10000      0  -10000  10000
S: 0 1  -10000  -10000      0  -10000  10000

CH8: Rudder mixer
------------
M: 2
S: 1 2   10000  10000       0  -10000  10000
S: 0 2   10000  10000       0  -10000  10000
```

Note that we assigned two inputs to the aerodynamic flight controls so that they would function in both fixed-wing and mulicopter mode. Also this mixer file actually specifies the inputs for 8 output channels, not 4. Just below the title, there is a section called “motors,” that has the following information:

4x 10000 10000 10000 0

This enables use of the quad-copter controller when the aircraft is in multicopter mode, and automatically assigns the first 4 channels to the motors. You can also specify a 3y configuration. In fact, that is what the mixer file used for the eflite convergence (which is the platform we used as a starting place for our airframe). I’m not sure what other options are available, but I am sure it’s in the px4 documentation somewhere. Otherwise, you can look at the other (existing) aircraft mixer files on the px4 Github. Assuming you set the parameters correctly, much of the VTOL transitioning is handled by the px4 software in the background.

After taking care of putting together your custom configuration and mixer files, you must drop them into a copy of the flight stack. To do this, you can clone the px4 github repository. The "Building the Code" documentation for px4 has a pretty good explanation on how to do that. Then you have to figure out how to actually build the code. It seems there's a ton of ways to do this, and they're mostly not well-explained…. Especially the methods that happen in Windows. I'll attempt to explain how I did this:

I spent a lot of time messing around with installing linux bash and attempting to do it that way, but finally I had luck with the Cygwin Toolchain. This is actually intended for Windows, and seems to work pretty well. So download and install the Cygwin Toolchain. You can find it at: https://dev.px4.io/v1.9.0/en/setup/dev_env_windows_cygwin.html Then (if you haven't already) install github and clone the px4 repository. Then clone it to your local machine.

After going through the steps to install the toolchain, go to the toolchain's directory. It defaults to **C:\PX4** Then run the file **run-console.bat** to start the bash console file. If you haven't cloned the repository yet, you can do that now. The necessary commands to do that are on the website I linked above. Next you want to navigate to the appropriate firmware directory and then build the flight stack. To build the stack, you need to enter the appropriate command for you hardware. This ensures that the toolchain builds a version of the flight stack compatible with your hardware. The commands for various boards are listed at this site: https://dev.px4.io/v1.9.0/en/setup/building_px4.html For example, I was building software for a Pixhawk 4, so I used the command: `make px4_fmu-v5_default`

I didn't go into much detail on the software build process, frankly, because it's been a few months and I don't remember very well. The website does an okay (but not great) job of explaining the process. I hope this is all helpful! Don't miss my overall tips & lessons learned:

Tips & Lessons Learned - More information on Parameters can be found at the Px4 site for parameter reference: https://docs.px4.io/v1.9.0/en/advanced_config/parameter_reference.html

1. Polyester resin and polyurethane foam are not compatible.
2. The original control gains were very high, at least for my purposes. I'm not sure if they were the px4 default values or if they were the values from the convergence profile we originally modified for this aircraft.
3. High control gains lead can lead to oscillations, but very high gains can lead to divergent oscillations. This led to several crashes. I never got my hover parameters perfect, but they weren't bad. They were as follows:
    a. **MC_PITCH_P = 4**
    b. **MC_PITCHRATE_P = .12**
    c. **MC_PITCHRATE_I = .05**
    d. **MC_ROLL_P = 3.5**
    e. **MC_ROLLRATE_P = .12**
    f. **MC_ROLLRATE_I = .05**
    g. **MC_YAW_P = 2.2**
4. Another thing to watch out for is the max rollrate settings. These seem to be linked to the control inputs. Decreasing them decreases your control sensitivity. With the stock values (220 deg/s), the controller would command huge amounts of control. I think it made it hard for both the flight controller and myself to properly control the aircraft. I changed the values like so:
    a. **MC_PITCHRATE_MAX = 100 (deg/s)**
    b. **MC_ROLLRATE_MAX = 100 (deg/s)**
    c. **MC_YAWRATE_MAX = 70 (deg/s)**
5. VTOL-specific parameters:
    a. Unless your aircraft is aerodynamically neutral (it probably isn't because it's VTOL), I would enable the weathervane parameter (Set **WV_EN** = **1**). This will keep the aircraft from fighting hard against its natural tendency to point into the wind. This over-controlling can lead to instability and loss-of-control.
    b. **VT_MOT_COUNT = 4** This depends on the configuration of your aircraft, but if you don't set it right, your aircraft will not function properly as a VTOL aircraft.
    c. **VT_TYPE = 1** This determines the type of VTOL aircraft. 1 means it's a tiltrotor aircraft.
    d. **VT_FW_MOT_OFFID = 24** This parameter is weird. These are the channel output numbers of the motors that shut off during fixed wing flight, if applicable. This uses the convention where the outputs are labled from 1-8, instead of 0-7. So on this aircraft, outputs 2 and 4 go to the two aft motors. As another example, on a

generic separate-lift-thrust quadplane VTOL aircraft, this parameter would be set to **1234**

   e. **VT_F_TR_OL_TM = 5** This is the open-loop transition time. That is, if you have the airspeed sensor disabled (see lesson-learned #6), this is how long the aircraft stays in transition mode before going into full fixed-wing mode. In some ways, higher numbers are better. The default value is 6 seconds, and that's pretty good.
   f. If you are using an airspeed sensor (and it's well-calibrated), check **VT_ARSP_TRANS.** I didn't use this parameter, so I won't give any advice on it.

6. Pitot tube (airspeed sensor) is not necessary if you set the following parameters:
   a. **FW_ARSP_MODE** to **1** Unless you set this to 1, a VTOL aircraft without an airspeed sensor will not complete a transition. It'll just cruise around happily in mid-transition…. Or it'll fall out of the sky. Depending on how it's set up.
   b. **CBRK_AIRSPD_CHK** to 162128. Not entirely sure if this is necessary, but I used it.
7. Position hold control gains:
   a. I set **MPC_XY_P = .3** This value could probably be higher, but I didn't have much time to tune it well. I set it like this before I had the controller really dialed because the aggressive maneuvers that came with higher values were causing control issues.
   b. **MPC_Z_P = .3** I'm not sure on this value either. I didn't spend much time dialing it in.
8. You CAN build the px4 flight stack in windows with the Cygwin toolchain and command prompt. I think that's how I did it, but I need to confirm.
9. Return Mode. Be intentional about setting your return and failsafe (such as return-to-launch) modes. It's also not a bad idea to test the Return mode. I found this out the hard way, and it cost me my only battery. Set the aircraft to hover for a finite amount of time before landing.
   a. Set **RTL_LAND_DELAY** = some positive, finite value. **15** or **30** sec would do. If you set it to -1, the aircraft will return and hover at RTL_RETURN_ALT until it regains signal or destroys your battery in an unsafe fashion. I wouldn't recommend that.
   b. **RTL_RETURN_ALT = 50** to **100 (m)** This is at your discretion… depending on whether you have a bunch of trees and other tall things to fly over.
   c. **RTL_DESCEND_ALT = 20 (m)** To clarify, this is the altitude the aircraft loiters at for RTL_LAND_DELAY amount of time after returning to the home/launch position.
10. When the pixhawk lands an aircraft in horizontal flight mode, it loiters over some position and then starts to descend towards a predetermined point. You can adjust the final approach in multiple ways. There's the loiter altitude, loiter radius, glide slope

angle, distance, and maybe others. It is worth noting that variation in true altitude can cause variations in where exactly the aircraft touches down. There might be a way of adjusting this, but I'm not sure exactly what that is.

11. Doing a range check is a pretty good idea. I knew this, but for once I "payed" for not doing it. It turns out that my Spektrum Satellite receiver doesn't work properly.
12. Be aware of receiving fasteners in composite materials…. Unless it's the fiberglass is very thick. Then it would be a good idea to tap the hole properly.
13. Make sure you have a lot of tension in the pivot belt… this is liable to be the slip point in such a mechanism. I got my system to the point where it typically wouldn't slip in flight, but it would slip in a crash or hard landing.
14. Plastic wrap is useful for holding shape on a wet layup. If you let the epoxy resin cure completely, the plastic wrap will release easily from the cured part.
15. There is an unusual failure mode with the pixhawk where you can use a custom airframe profile but then "update" your firmware. If you do this via the typical method of updating your firmware with the latest public copy of px4 (that qgroundcontrol downloads), then the vehicle will no longer have your custom airframe data on hand (the data that comes from the configuration and mixer files). That is, the controller will reference the **SYS_AUTOSTART** parameter and look for an airframe that is not there anymore. I had this happen because I did not realize that the custom airframe configuration was embedded in the firmware. It was not obvious what was going on, but Qgroundcontrol no longer recognized my transmitter inputs. It gave the following error when attempting to calibrate the radio: "Detected 0 radio channels. To operate px4, you need at least 5 channels." There's quite a few things that can cause this error, but I think this was a rare edge case cause that wasn't well documented. If you want to switch to a stock airframe profile, that will fix this issue. You can do this in the qgroundcontrol airframe menu or via changing the **SYS_AUTOSTART** parameter. However, if you stick with your custom airframe profile, you will have to install a (old or new) firmware copy that has that custom airframe imbedded in it.
16. For an unknown reason, sometimes the pixhawk's reading of level horizon would wander. Once or twice I took off and it took a large amount of pitch input to maintain level flight. I would just recommend that you make sure that it is correct before takeoff.
17. It turns out that the external power distribution board that comes with the HolyBro Pixhawk 4 has places for you to connect your motors. Using this, it can measure the amount of current, and therefore power, consumed. I did not realize this initially and bypassed it with a separate wiring harness. This stinks from a data-collection perspective, but I wasn't sure whether the board could support enough current for this aircraft anyway. I would love to find more data on this board.
18. This is a bit obvious, but make sure you have a battery that can really support the current draw you need. VTOL airplanes use a lot of power, and my cheap battery could barely

handle it. The voltage would drop terribly under operation, and I only drew ~1600 mah from a 5000 mah battery and achieved a ~3 minute hover flight times as a result.

19. If you're building a VTOL aircraft from scratch or bashing one together, be aware of the torque-through-the-wing problem. On a plane like mine (or most quad-vtols), you get substantial torque between the two sets of motors when the aircraft tries to yaw. On this aircraft, that was supported by the wing's structure. Combined with a few crashes, this structural integrity slowly deteriorated because this wing wasn't built to withstand these loads.
20. Until the aircraft's hover control is very well-dialed in, beware of going from any manual mode to position mode. This is also true of going from acro mode to stabilized mode. Sometimes this can lead to very sudden control inputs as the controller switches modes and tries to regain control. If your controller is not fairly well tuned, this can destabilize it.