Random notes

Baud rate for DJI GPS is 9600, chip has SDA and SCL so may have compass chip onboard. Compass chip says 7140 010 but it’s probably proprietary DJI crap. Backup is compass sensor in front of Mavic. GPS ANTENNA BAD, chip seems ok. 🡪 Bought new GPS antenna from digikey, will attempt replacement shortly

Network comms

Use p2p approach since LTE router filters all ports, bookmarked p2p chat python application 🡪 steal code

Networking parameters 🡪 port is gonna be 55555 with addy ludicroustech.ca

p2p\_server.py will run on a RPi hosted at ludicroustech.ca:55555 and it allows two peers to connect to each other by sending their addresses to themselves

Data is sent as lat/lng/alt from cell phone to server. Future comms may include telemetry but everything is one way for now.

Software

3 basic modes, travel, takeoff, and land. Travel mode needs to get location and orientation of aircraft using GPS and compass after which it should be able to compute a vector in order to reach the destination coordinates. Using this vector, travel mode will apply the necessary input to the flight controller. Use a speed function to regulate how fast the drone is going (magnitude of the vector): speed function should allow for quadcopter to speed up smoothly instead of 0-100 and it should slow down quadcopter when near the destination. Takeoff mode will be simple, spool up and go straight up to predefined altitude otherwise default to a specific value. Landing should use ultrasonic sensor to make sure it slows down significantly before touching down. Ultrasonic sensor will be linked to Arduino so RPi and Arduino need to have a special mode that ensures rapid communication when landing.

Output map is gonna be AETR + Arm, data in SBUS packet spans 22 bytes that contain data for 16 channels (each channel is 11 bits to allow for a resolution of 1024)

SBUS header is 0x0F and footer is 0x00. Byte[23] contains both channel 17 and 18 data as well as failsafe and lost frame data. I have set all of these values to 0 using a 0x00 byte, hoping this works but more testing is needed.

Files:

* Main.py
* Flight.py
* Misc.py
  + Basic functions like log and whatnot
* Network.py
  + Network code to connect with peer server
  + Contains vars of target coords, altitude, and flight mode

Connection revisions

RPi UART RX will be for GPS and TX will be for SBUS out. Arduino Nano will have relay that can switch between FC sbus and RPi sbus based on input it receives from the receiver by analyzing sbus. This is how I can ensure a failsafe and have the drone fallback to an internal RX communicating with a remote control. RPi’s USB port will go to an FTDI thereby opening a second UART bus that will be used for communication with onboard Arduinos. RPi will connect to internet through WiFi to make things easier wiring-wise.

Seeduino Pinout

1. Left Arm Light (gray)
2. WS2812b LEDs (green)
3. Relay signal (white)
4. ECHO pin (red)
5. TRIG pin (blue)
6. N/A
7. UART Tx (black)
8. UART Rx (yellow) for i6X
9. N/A
10. N/A
11. Right Arm Light (gray)

Arduino Nano Pinout

1. UART Rx (yellow) for i6X (channel 7 dictates control and 8 dictates LEDs)
2. UART Tx (black)
3. Left Arm Light (gray)
4. WS2812b LEDs (green)
5. Relay signal (white)
6. ECHO pin (red)
7. TRIG pin (blue)
8. N/A
9. N/A
10. N/A
11. N/A
12. N/A
13. N/A
14. Right Arm Light (gray)

<https://basejunction.wordpress.com/2015/08/23/en-flysky-i6-14-channels-part1/>

<http://blog.dsp.id.au/posts/2017/10/22/flysky-ibus-protocol/>

Status Pins (21,20,19) 🡪 19 is last bit of 3-bit combo

* Pi offline 000 – all red
* No internet 001 – front flashing red
* Cannot takeoff (No GPS, mag, etc) 010 – front flashing yellow
* Ready to go 011 – back flashing quick green
* Automatic control mode normal speed 100 - hazard
* Automatic control mode fast speed 101 – police
* Beacon 110 – all LEDS white
* Cancer 111 – strobe all white
* User control does not have any way to be controlled by pi since pi doesn’t know when it is in control – green strobe no back