ESP8266 Multirotor Flight Control Board

Code Explantion and how to..

3rd April 2017

Hemanshu Kale
hemanshu.kale@gmail.com

INDIA

Table of Contents -

Electronics -

- 1 ESP8266
- 1* 6DOF sensor (ex. MPU6050) / any 9DOF sensor
- Header pins, dotted pcb and soldering equipment
- 1 USB-TTL converter for programming the esp
- Male header pins 2.5mm pitch
- Male header pins 2mm pitch – these can be soldered to ESP,
  (After soldering these to ESP, I forced-bent them into the 2.5mm breadboard)
- For programming, you can use female bergstrip or male header pins, whichever convenient to connect to FTDI or any USB-TTL converter
- Insulated wires – sometimes you might need to jump some connections over one another
- If you want Voltage monitoring functionality, by using ADC pin of ESP, then more resistors will be needed to step down the battery voltage to less than 1 V (I used 10K and 47K)

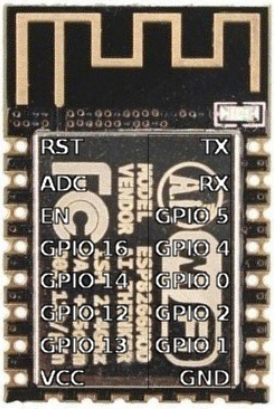
For coreless type-

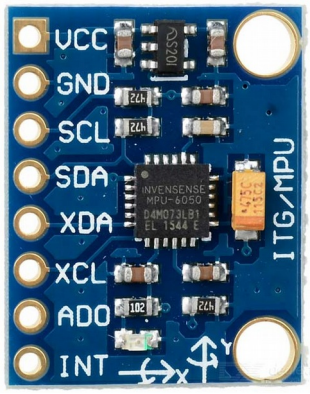
- 4 * coreless motors with props
- 4 * Motor driver – MOSFETs - any will do with specifications greater than 4.2V and 4A
- 4 * button cells or Level shifter if MOSFETs with max gate cutoff < 3.3 V are not available
- Zener diode as voltage regulator
- Around 4.3 ohm ohm resistance for above purpose

For brushless type-

- 4 * Brushless Motors
- 2 * ESCs
- 1 AMS 1117 3.3V Voltage regulator

Pinouts of Modules -





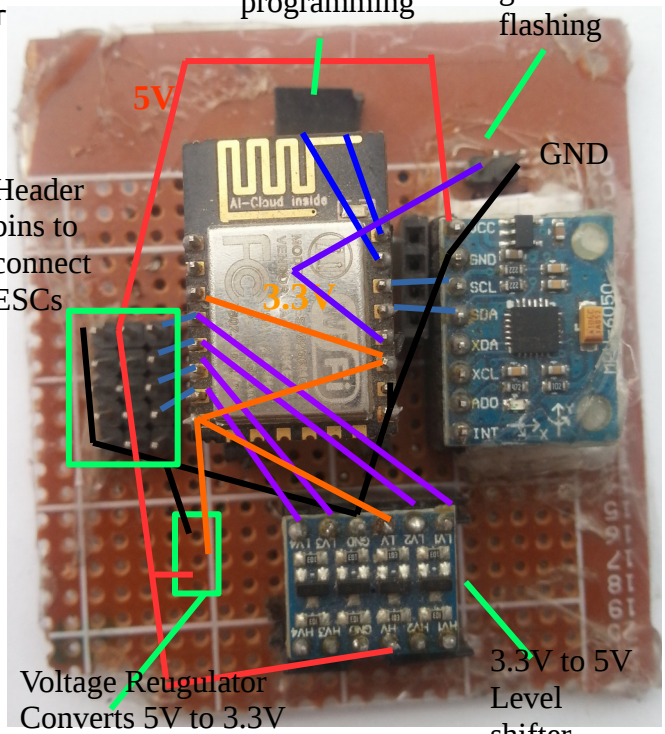
ESP GPIO 5 - SCL MPU
ESP GPIO 4 - SDA MPU
ESP GPIO 15 - Pulled down
ESP GPIO 2 - Pulled Up
ESP GPIO 0 - Pulled down only for flashing
ESP EN - Pulled Up
ESP ADC - 3.3V -- 47K -- ADC -- 10K -- GND

ESP VCC - 3.3V – from regulator
ESP GND - GND
MPU VCC - 3.3V – from regulator
MPU GND - GND
All GND are connected to GND of the LiPo

Structures I made ->

This is my first one, hence used berg strips for debugging purposes and it is not at all compact..
All the connections are shown..
This can be used for both the types of motors..
Connect the 4 ESCs to the male header pinouts on left side
or connect the MOSFETS/drivers to the output of level shifters below

AMS1117 3.3V is used to convert 5V of ESC into 3.3V for the ESP



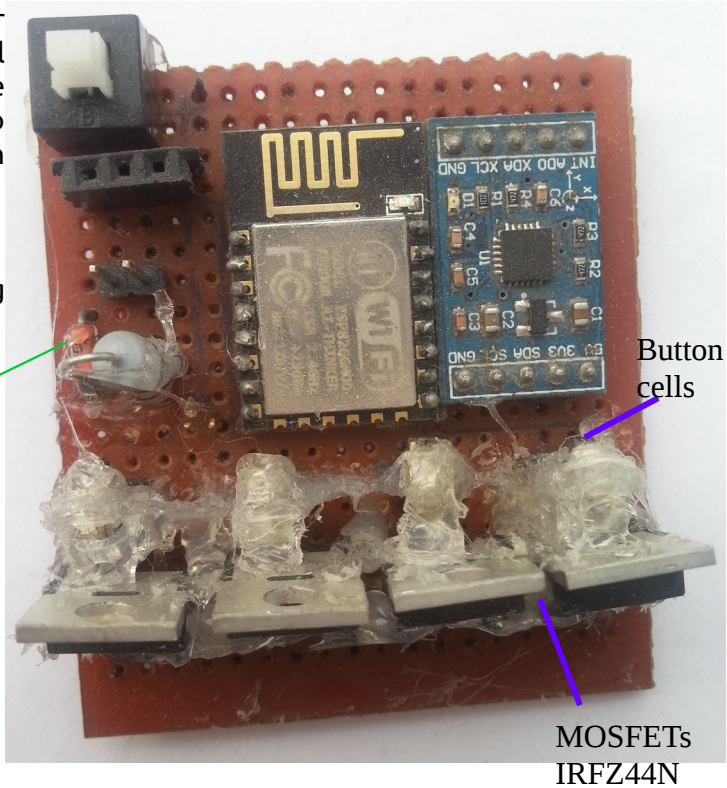
ESP GPIO 16 – (P1=M1) - LV1 (Level Shifter)
ESP GPIO 14 – (P2=M2) - LV2 (Level Shifter)
ESP GPIO 12 – (P3=M3) - LV3 (Level Shifter)
ESP GPIO 13 – (P4=M4) - LV4 (Level Shifter)

The 3pin connector of ESC can be directly plugged into the respective pins

This second structure is more optimised for coreless motors and relatively compact.. All conections are similar to the above structure except that gate of MOSFETs are connected to the GPIO of ESP with a 1.5V button cell in between to increase gate voltage.
Here GPIO 16 is not used and 3 is used instead

And the switch is for physical arming/disarming of the quad.

This is made to run with a single cell LiPo hence a Zener diode is used as Voltage regulator to get 3.3V



ESP GPIO 3 – 1.5V Cell – gate of MOSFET1
ESP GPIO 14 – 1.5V Cell – gate of MOSFET2
ESP GPIO 12 – 1.5V Cell – gate of MOSFET3
ESP GPIO 13 – 1.5V Cell – gate of MOSFET4

Corresponding drains of MOSFETS then will be attached to corresponding motors and the other terminal with VCC of LiPo battery

If you have a MOSFET / motor driver that completely turns on with a gate voltage of 3.3V, then directly connect the GPIO pins to the driver

About Code -

The 2 codes, of ESP and of processing are meant for each other...

Processing Code -

I have made this GUI to control the quad as easy as it is to play games-

- use 'W' and 'S' to pitch forward and backward respectively
- use 'A' and 'D' to roll left and right respectively
- use 'Q' and 'E' to yaw counter-clockwise and clockwise respectively
- use Space to jump (increase throttle here)
- use Shift to Crouch (decrease throttle here)
- use 'R' to increase rate of change of throttle by Space and Shift
- use 'T' to decrease throttle rapidly, in case if it is ever needed to
- use '`' to reconnect to ESP
- use 'F' to arm and disarm the quad..Red background color indicates armed and green indicates disarmed

  What this does –

  Changes the '60' between 'g' and 'h' in aAAb90c90d5e90f90g60h to '120' to arm and back to '60' to disarm. The esp code is configured to read any value above 90 in 6^{h} channel i.e. between 'g' and 'h' as armed

- use 'M' to change PID mode

  currently, there are 4 PID modes in the ESP sketch.. discussed in more detail in the ESP skech explanation. Keep pressing 'm' to change through the modes

  What this does –

  changes the first 'A' in aAAb90c90d5e90f90g60h to 'B' then to 'C' then to 'D' and finally back to 'A'

- use '/' to change the view mode...

  This toggles the mode in which different values from the ESP are shown like accel / gyro / motor values / none. Keep pressing '/' to change throught the modes.

  What this does -

  changes the second 'A' in aAAb90c90d5e90f90g60h to 'H' then to 'G' then to 'I' then to 'J' and finally back to 'A'

- use '0' to enter PID mode (currently only enters PID mode when disarmed, for safety purposes, but that can be changed)

  In PID mode -

  - press '0' if you don't want to save the new values in quad or in the client and both will keep using o
  - ld values when restarted

    What this does -

    Sends this string containing all the PID values - **pMq1r1s1t1u1v1w1x1y1z**

  - press '.' to save the values on client and on ESP side, values are stored in a text file in the sketch folder with name - "pid_values.txt" and loaded at each boot.

    What this does -

    changes the M in the string to N.. this tells the ESP to save the string in EEPROM

- use ‘p’ to load PID values from quad
  If you’re using the client for the first time on a new system, the values present in the text file may not be the same as those present in quad.
  Hence, when you press ‘p’, the values from quad are loaded and shown in the GUI
  Press ‘=’ If you to want to save these values into the sketch and subsequently the file
  Else, press ‘p’ again to discard and stop showing the values

- use 'H' to enter trim change mode -
  - press ‘H’ to cancel saving the mode
  - press ‘N’ to save trims into the disk – trims are saved in a file “trims.txt” in the sketch folder and loaded into sketch at each boot

  Even when your quad shoud ideally be hovering perfectly still when you give same inputs to all motors, it might drift to a side due to multiple factors like weight offset, winds, miniature manufacturing differences in motor/propellers or sensor offsets.
  So we need the ‘trim’ functionality which gives a small counteracting offset ,i.e. it changes the centre of yaw/pitch/roll control so that the quad will be perfectly centered

  Changing trim follows the same layout WSADQE mapped to IKJLUO
  ex. If quad is pitching backwards at neutral position, you need to pitch it forward to get into perfect hower state, press ‘I’ which corresponds to ‘W’ to set the forward pitch trim.
  Besides each indicating stick, there is some text ‘t = #’
  This number indicates the value of current trim

- Sensitivity -
  To pitch forward, you press ‘W’, but to control the extent of the value change, there is this parameter -

  map the layout WSADQE to IKJLUO

  after you press ‘W’ (pitch forward),
  If you want to make it go more forward (relatively forward), press corresponding ‘I’
  Else if you want it go less forward (relatively backward), press corresponding ‘S’

  after you press ‘S’ (pitch backward),
  If you want to make it go more backward (relatively backward), press corresponding ‘S’
  Else if you want it go less backward (relatively forward), press corresponding ‘I’

- use ‘C’ to enter Calibration mode -
  - press ‘C’ again to cancel without saving
  - press ‘V’ to save the current calculated offset

  It will be explained in detail in the esp part...
  What it does -
  Sends a String “**Calib**” to esp, which tells esp to start the accelerometer calibration mode, in this mode, move the board such that the accelerometer will face the max and the min values on all the 3 axes – place an imaginary cube around quad and rest it on all the 6 sides (all side downward facing atleast once)...
  Then when you feel that the min and max values are not changing much, you can stop & save it by pressing ‘V’. This sends the text “**Csave**” and the offsets are saved in the EEPROM of esp.

  Take care to move the board very smoothly in the process... any jerk might spike up the max value of accelerometer and give bad offsets.. just start again if it happens..

One of the main features of these codes is that PID values can be changed realtime and remotely - you don't need to reprogram you quad always when you need to change the values

There is a display that indicates current position of throttle, yaw, pitch, roll controls
and curent PID values, trims and sensitivity
Also provision is made so that we can get and see the live aceel / gyro / motor values from the quad

Explanation of all Strings transferred-

- Normal drive (sent to ESP)-
  All max values (constrained) - aAAb175b175d175e175f175g120h
  All min values (constrained) - aAAb5c5d5e5f5g60h
  Breakup -

channel = ch
(PID type A or B or C or D) trial and error with different types of PID.. you can add at will

ch0 (pitch) ch2 (throttle) ch4 (roll)

aAAb5c5d5e5f5g60h

ch1 (yaw) ch3 () ch5 (arm channel – 60 is disarmed, 120 is armed)

(Display mode - A or G or H or I or J) LiPo Voltage

A - no values are shown ; Received String - vo (V) l
G - gyro values are shown ; Received String - gg (V) i (gx) j (gy) k (gz) l
H - acceleration values are shown ; Received String - hh (V) i (ax) j (ay) k (az) l
I - motor values are shown ; Received String - nn (V) i (m1) j (m2) k (m3) l (m4) m
J - roll/pitch angles are shown ; Received String – pr (V) i (pitch) j (roll) k

The received strings are identified according to the first 2 letters and thus processed by processing

- Changing PID (sent to ESP)-
  all max values - pMq999r999s999t999u999v999w999x999y999z
  all min values - pMq1r1s1t1u1v1w1x1y1z ;

p(change or save- M or N)q (kpP) r (kiP) s (kdP) t (kpR) u (kiR) v (kdR) w (kpY) x (kiY) y (kdY) z

Here, for ex. (kpR) - the third letter (capital) indicates Pitch/Roll/Yaw - Roll (in this case)
and the second letter indicates proportional/ Integral / Derivative - Proportional (in this case)

Received String - vo (V) l – to indicate voltage

- Displaying PID values from quad into the GUI
  When you press 'p', Processing sends this string - "GPID"
  When ESP recieves it, it responds by sending the current PID values in it in the following format -

  Kq1r1s1t1u1v1w1x1y1z

  It has the PID constants in the same order as in the above String

When you want to calibrate accelerometer - String sent is “Calib”

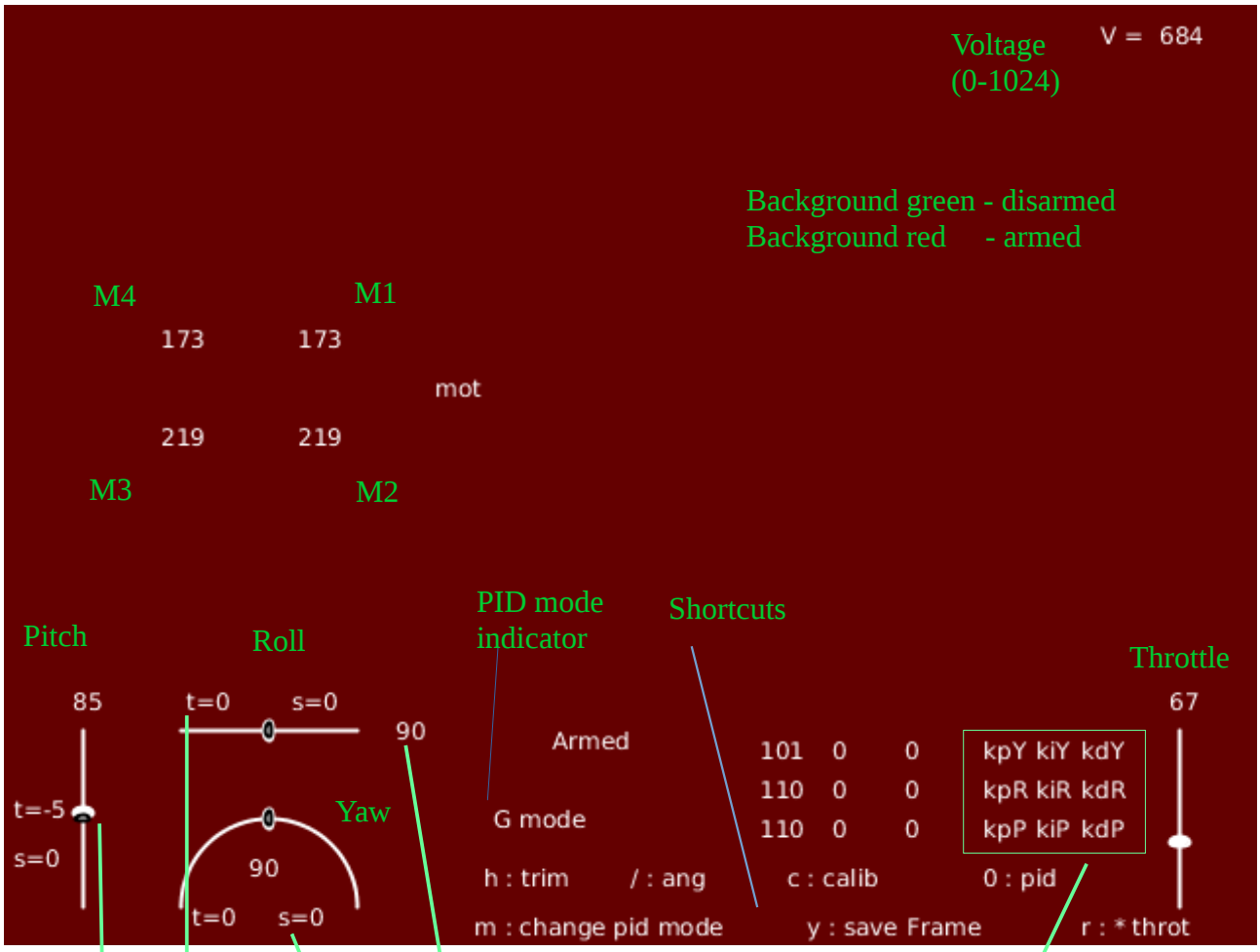
Strings returned are (in the same order then repeat) -
Range of recieved strings - LiPo Voltage
Current value of accel -> cu0i-1676j-1676k-1676l to vv1023i1676j1676k1676l
Max value of accel (during calib) -> ma0i-1496j-1496k-1496l to ma1023i1496j1496k1496l
Min value of accel (during calib) -> mi0i-1496j-1496k-1496l to mi1023i1496j1496k1496l
Values of offset (during calib) -> oo0i-1496j-1496k-1496l to oo1023i1496j1496k1496l

The first 2 letters are indicators of what the value is so that processing will understand and put it into proper collumn..
next number is battery voltage -> gives value between 0 to 1023 for 0 – 1V
Number between i & j -> X axis
Number between j & k -> Y axis
Number between k & l -> Z axis

When you want to save the calculated offsets -
String sent is “Csave”

GUI Screenshots -

10K & 47K were used for voltage reduction
Hence, battery Voltage = (684/1023)*(57/10)
= 3.811



Trim indicator

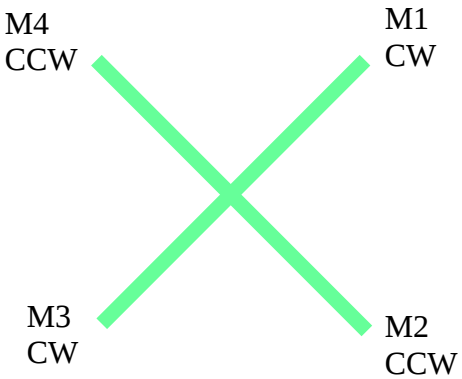
Sensitivity indicator

Black – centre position
White – current position

Current value

Shows PID mapping -
Same structure is followed to display value and to change it..

i.e. – kpY = 101 ; kiR = 0
Now map this to your numpad -
i.e. kpY (top left) corresponds to 7 (top left)
kdP (bottom right) corresponds to 3 (bottom right)
Hence, to change kdP,
Enter PID mode by pressing 0,
Hold 3 and press
+ to increase or
– to decrease
Then '.' to save



M1, M2, M3, M4 corresponds to P1, P2, P3, P4.. respectively in the esp code

ESP code -

Some theory -
There are 2 types of motors which are generally used for RC copters -

- **Brushless DC motor**
  - Has 3 terminals and need an ESC (Electronic Speed Controller) to run
  - ESCs need ppm as input which is provided by Servo Library
  - Have a 3 pin connector (to connect to microcontroller)
    usual colors – Yellow/Orange/White is signal, Red (always central wire) is Voltage and Black/Brown is GND
  - Connect the signal terminals directly to pins of ESP
  - Some types of ESC don't provide 5V output (opto ESC) a BEC is needed to power the esp from LiPo.

- **Coreless DC motor**
  - Applications are generally micro quads (palm sized)
  - Have just 2 terminals.. changing polarity changes direction
  - Need a motor driver component in between as microcontroller cannot provide that much power
  - L2938 or MOSFETS can be used as motor drivers
  - Need pwm to change speed – Higher the duty cycle, higher the speed
  - L2938 and MOSFETS usually need 5V input to function – ESP provides max 3.3V
    Hence a Voltage level shifter might be needed or you can hook up a 1.5v button cell - GND towards GPIO and positive towards mosfet
  - Else get a MOSFET that needs a gate voltage less than 3.3V
  - Try both MOSFETS and L2938 to know what's better for your application

GPIO 16 has some RTC applications so it behaves differently with respect to other pins -
it turns high for a few miliseconds at startup, so it is risky to use for pwm applications as it will cause the corresponding motor to run at full speed for a few miliseconds.
Hence, dont use this pin if you're planning to use coreless motors...
It is fine to use with brushless motors
Now, as pin 16 cannot be used, I am using pin 3 (RX) as the one of the pins for the motors.

The builtin LED of ESP can be on different pins.. usually 2 or 1.. depending on the model
But it is almost always the TX pin
The LED is used for showing current status of the quad

As both the RX and TX pins are use in different purposes, Serial communication can never be started.. So there is this variable 'debug' which when false, enables using of the TX pin as LED indicator.

However, if you are using pin 3 (RX) as one of motor controls, then don't make this variable true.. as there might arise a conflict between Serial code and GPIO code for the same pin..

The variable 'pwm' when true, makes the code for coreless motors and when false, makes the code for Brushless motors..

Communication over Wi-Fi is done by Websockets.. server is created on the ESP on port 300 and the client (processing) connects to it

Many elements in the arduino code are self-explanatory here and code have been commented well...
So I am just explaining an overview.

Setup part -

- Starts Wi-Fi and connects to the static ip mentioned in the code...
- (can also start its own AP but it increases the load on ESP which might make it slower..)
- loads the accelerometer offset from EEPROM
- Calculates the gyro offset
- Writes 0 to motor in case of pwm
- Or calibrates the ESC in case of using ESC
- There are many delays in the setup as they were found to be essential for smooth running of the code –
- ESP tends to overflow its stack in case of minor code mistakes, also,
- When Wi-Fi is connected, it has to do some hidden Wi-Fi functions to ensure proper connectivity, it does the functions during delay and end of loop.. so proper delays are essential..

Loop Part -

- Checks if any data is being recieved over Wi-Fi
  it expects 3 types of potential strings -
  drive String - aAAb5c5d5e5f5g60h
  PID String - pMq1r1s1t1u1v1w1x1y1z
  calibration String - Calib
  They are identified by the first letter and processed accordingly..
  some variables like had_wifi or no_wifi are flags that check if Wi-Fi is connected..

- If Wi-Fi / client is lost for more than 370ms, it resets the yaw pitch roll controls to neutral positions..
  if Wi-Fi is lost for more than 500ms, it a function dead_rf() is called which decreases the throttle slowly.. till it reaches 0 and then disarms quad

- Raw values are read, offsets are subtracted, Low-pass filter is applied and pitch/roll are calculated...
- Error is found for use in PID..
  as currently there are 4 types of PID mode are present, 4 types of errors are found..

- Motor values are calculated by using PID..
  the PID I have used doesn't take in amount time curently.. as I am trying out different methods.. but you can easily tune it to however you want...
- Motor values are contrained and accordingly send to the motors

- There is an option to see if motor / gyro / accel etc values are proper, in Serial monitor..
  for this you need debug = true and ensure that motors are not connected to RX pin or the motors are not powered

- LED blinking meaning -
  Slowest - 2 sec on & 2 sec off -> Disarmed and client not connected
  Slow - 1 sec on & 1 sec off -> Disarmed and client connected
  Fast - 0.5 sec on & 0.5 sec off -> Armed and client disconnected – soon throttle will decrease and quad will get disarmed
  Fastest - 0.1 sec on & 0.1 sec off -> Armed and client connected – better keep distance :P

Tips -

Before your first flight, here are a few things to do -

Don't connect the propellers to motors the first thing,
First - turn on both the sketches and do an intesive check up whether everything is good -

- Check if the values given by sensors are consistent with the motion you give

- Check if change in motor values is consistent with -
  - the motion you give (yaw/pitch/roll)
  - the controls you give (yaw/pitch/roll)

  To make the change easily visible when motor values are displayed in processing, average of last 30 values are taken and depending on whether current value is larger or smaller than the average, yellow or cyan is shown respectively.

  Hence, to check ->
  Give control left roll -
  If M1, M2 increases (becomes yellow) and M3, M4 decreases (becomes cyan) then its good
  Roll the board/quad towards right
  Hence to nullify this effect, If M1, M2 increases (becomes yellow) and M3, M4 decreases (becomes cyan) then its good
  Accordingly perform rigirous tests on each channel and motion till you're sure everything is perfect.
  In case of problems, you just need to change the signs of some variables

- After connecting the propellers, fist hold the quad in your hand properly and tightly (props should be as away as possible) and increase the throttle till you feel its the hower throttle.
  Then give some motions to quad and analyse the resistance..
  If the resistance is high and quad starts oscillating, kp is high and should be reduced..
  If resistance seems to be low, increase the corersponding kp
  Fly only when you feel resistance is proper

- Calibration is performed at every boot and so, don't move the quad for a few seconds after starting it..(when LED is blinking randomly)
  The setup part is finished when you can see the LED blinking with time period of 2 sec off and 2 sec on. This is default status and it indicates quad is disarmed and no client connected..

- It is advisable to wait a few seconds after boot for connecting the client till the above mentioned LED status is seen.

- When you run the skecth first time, the EEPROM of ESP might be filled with garbage / last values... so, save the Calibration and PID values as early as possible so that the values are synced between both devices

- Even though the quad worked fine for my trials, there is no coprocessor in the structure, so no failsafe is present.. There is a chance that you might lose your quad forever or end up hurting someone in case something goes wrong, so choose the flying place carefully.

- **About Calibration -**

  There are 2 types of Calibration performed and both are a bit different

  - **Gyro Calibration -**
    - Gyro Sensor returns angular velocity values
    - When quad is at rest position, all the values should be ideally 0 or atleast oscillate around 0, but usually its not the case because these sensors develop offsets, i.e. the value for x axis might oscillate at around 150 and the number might be different for each axes.
    - Hence, at each boot, we call some values from sensor, till the transient effect is taken care of. (320 are called in this code)
    - Then average of next 1000 values is taken for all axes and these are the unfiltered offsets..
    - Now, when offsets are subtracted from raw values, we get proper values that almost oscillate around 0
    - But these values oscillate a lot, hence a low pass filter is used
    - Now, from this offset-nullified and filtered data, average of next 560 values are taken to find the filtered offset.
    - The values are divided to scale them to needed use

  - **Accelerometer Calibration -**
    - This sensor return the acceleration on all the 3 axes
    - Now ideally, when an axis is perpendicular to gravitational field, it will show 0 as acceleration, but there are offsets almost everytime..
    - This sensor cannot be calibrated like gyro because every time, the quad will be needed to be in perfect horizontal postion for proper calibration and this is not possible always
    - Hence the method followed here is little bit different..
    - In the Calibraion mode, all the axes of sensor are exposed to all the possible values.. ranging from the highest positve acceleration to lowest negative acceleration -
      ex- keep the quad horizontal (nose forward) and and rotate it 360° around the pitch axis (as slowly as possible as jerks will cause value to spike and affect the offset)
    - This will cause the x axis of the sensor to get all the possible values and we can get the max positive and minimum negative
    - Now when you feel the max and min values are properly registered (they are continuously shown in GUI), press ‘v’ to save and the offsets are stored
    - Now we take average of these 2 values and and that is the offset for the particular axis
    - These offsets are stored in EEPROM, so just calibrate once and you’re good to go, though it’s a good practice to Calibrate every now and then..

Please do inform on the mentioned id for any mistakes/suggestions

Thank You and Happy flying :)