PlainFlight

Flight Stabilisation Software

Instruction Manual For

PlainFlightController V1.1.1

Revision History

Version	Comments	Date
V02-00	Updates for new levelled mode, battery monitor, ESC calibration and LED outputs.	12/2/2024
V01-02	Initial release.	24/1/2024
V00-05	Development	15/12/2023

Contents

1. Introduction

1.1 Background

PlainFlight stabilisation software is for the RC pilot who wants to get the most from their model, needs to master an unstable aircraft, or simply counteract environmental conditions for an enjoyable flight.

Originally created as a home project it quickly became more with its performance, ease of build and low budget parts. These qualities led to it being refined and posted on GitHub for other hobbyists to have a go and enjoy.

While PlainFlight has been developed for small electric powered model planes it could easily be modified and used upon other small radio-controlled craft. This can be done with little effort as the code is broken down into logical modules and is well commented for those that want to understand or modify for their own purposes.

The flight controller hardware is based upon the Seeed Studio XIAO ESP32 boards and the ever-popular MPU6050 IMU. It's simple to build, easy to program via Arduino IDE and cheap when compared to many commercially available flight controllers.

1.2 Specifications

PlainFlight has the following specifications as detailed in Table 1 as standard:

Table 1 - Specifications

Feature	Detail
Model Mixes	Plane Full House (aileron/flaps/elevator/rudder) Plane Full House V-Tail (aileron/flaps/V-tail mix) Plane V-Tail (V-tail mix for rudder/elevator) Plane Rudder/Elevator (rudder/elevator) Flying Wing (elevons/rudder)
Flight Modes	Pass Through Gyro Rate Self-Levelled Heading Hold
Actuators	4 servos, 2 motors (Or any combination of the 6 with modification)
Actuator Refresh Rates	50Hz, 100Hz, 150Hz, 250Hz, 300Hz, Oneshot125 (limited to 2KHz) (Or custom with modification)
Motors	Standard throttle response, or throttle/yaw based differential thrust can be configured for twin motors for any model mixer.
Radio Protocols	Sbus
Battery Monitor	1-3s Lipo as standard, actively reduces power to motor when voltage drops below a defined threshold.
Failsafe	Automatic transition to self-levelled mode.
LED	Flight mode indication
Target	Seeed Studio XIAO ESP32S3. (ESP32C3 will also work with small modification to software).
IMU	MPU6050 (GY-521 breakout board)
Control Loop	Stable 1KHz

1.3 Scope

PlainFlight software is only intended to be used on small electrically powered model aircraft, with possible adaptions for other small radio-controlled craft.

No warranty or guarantee is given with this software or accompanying documentation, refer to license agreement for full details upon using this software.

2. How To Assemble

It is recommended that the flight controller is built onto PCB prototype board for rigidity and robustness, but for the lightest weight application it can be built with 'flying wires' connecting between the components. Figure 1 and Figure 2 detail the wiring diagrams of both 5V (BEC powered) and 3.3V (1s Lipo powered) systems. They both show connections for a 'full house' plane with 4 servo outputs and 2 motors. This wiring configuration can be used for any of the 'Model Mixers' in the software as all servo/motor outputs are given.

Note: Servo/motor power supplies have been omitted for clarity.
Note: The ESP32S3 I/O's are not 5V tolerant and needs the series resistors to drop voltage / current limit.
Note: 470uF 16V capacitor protects the ESP32S3 from voltage spikes.

Figure 1 – 5V Wiring Diagram

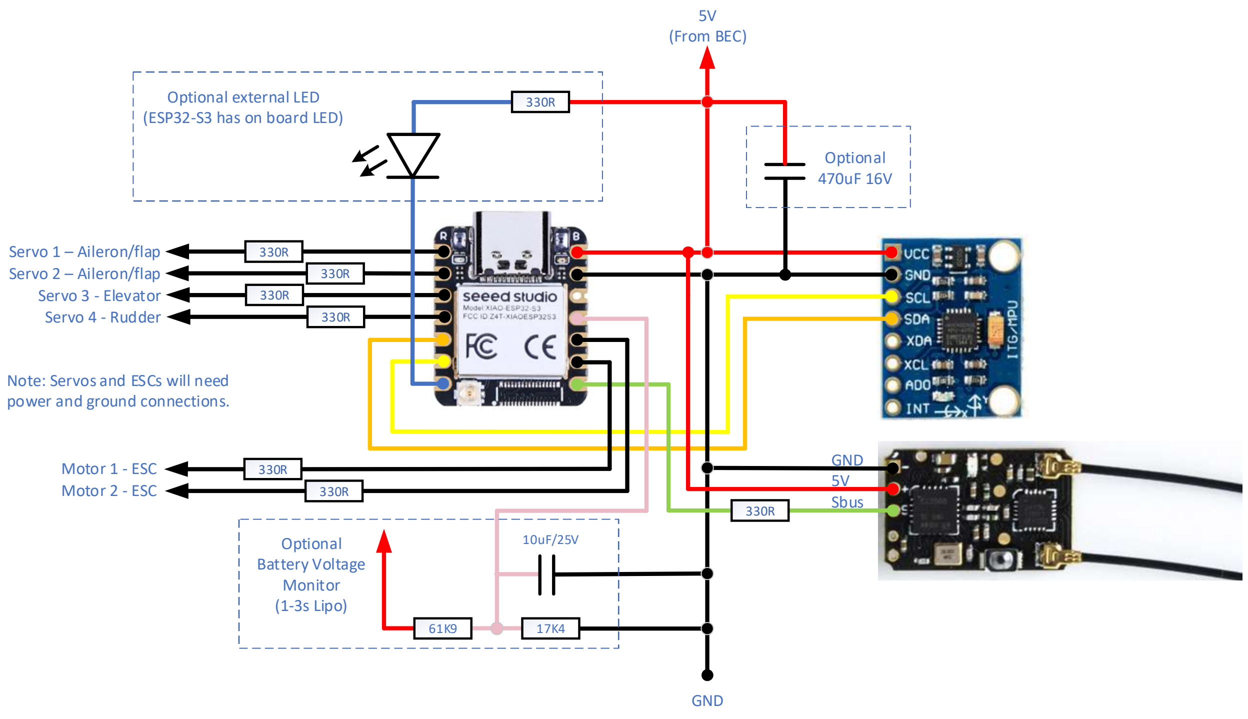
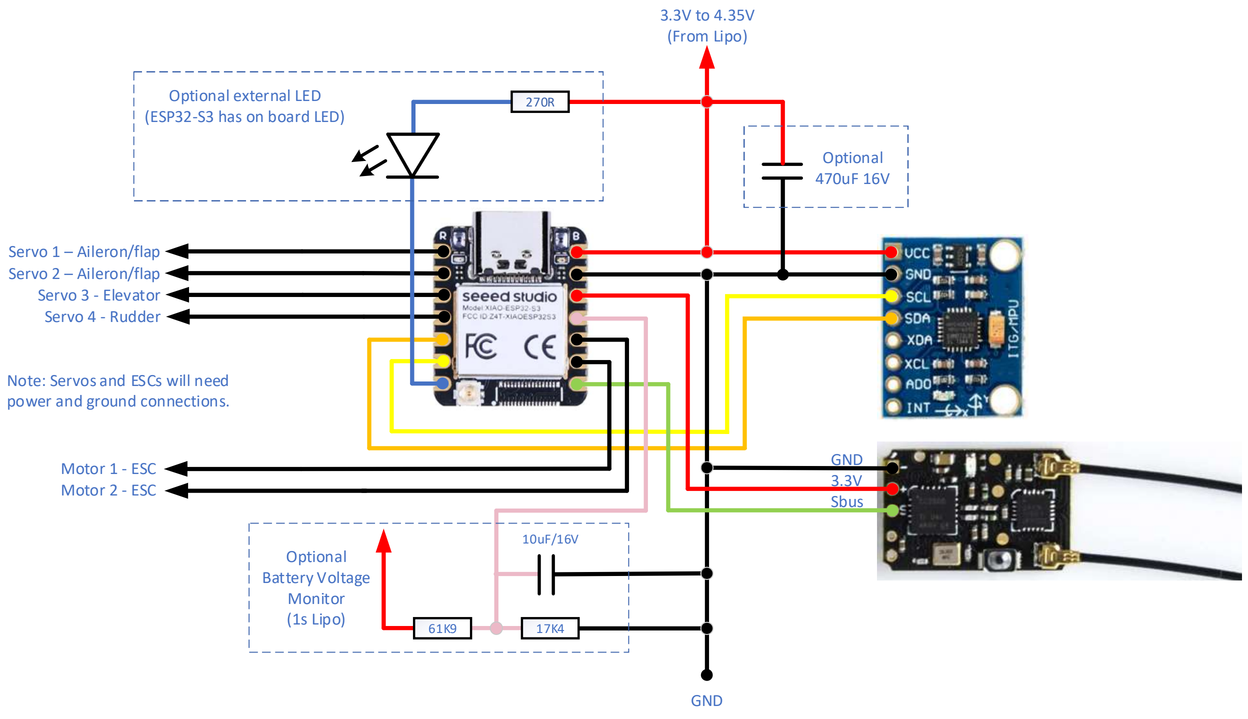


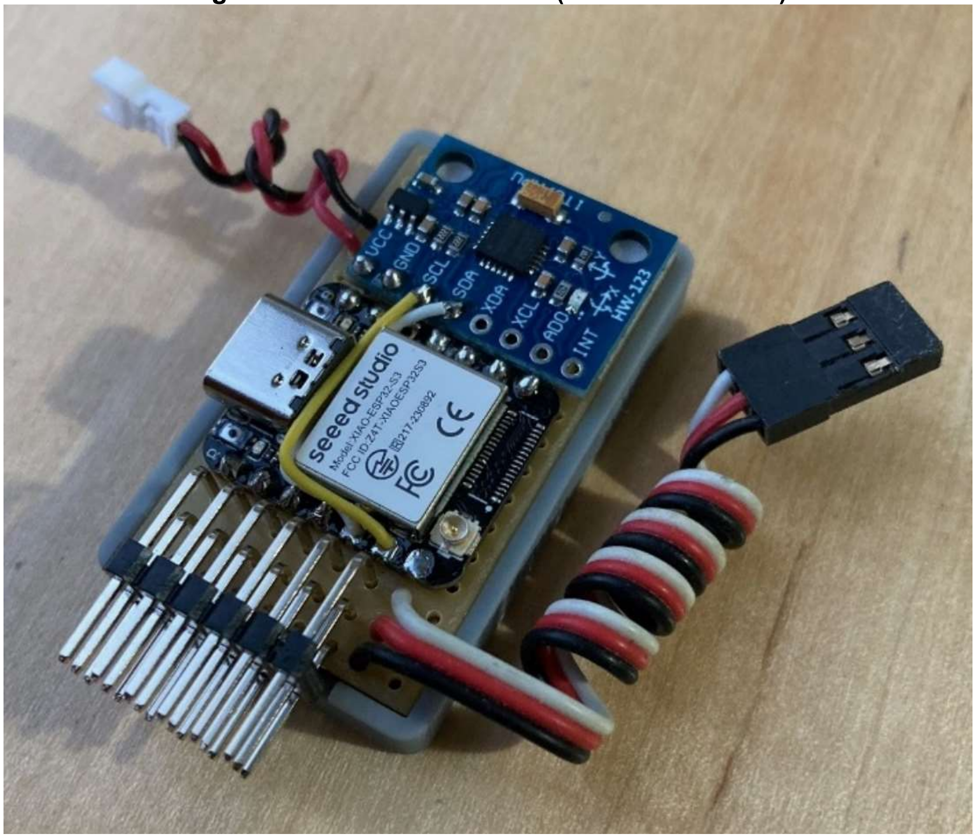
Figure 2 - 3.3V Wiring Diagram



Note: Strongly recommend 470uF 16V capacitor on 3.3V systems supply rail if using BLHeli ESC's to absorb voltage spikes, and generally recommended to prevent the possibility of brown out on 3.3V and 5V systems.

Figure 3 details a typical build upon prototype board. This is the 5V version and all resistors are 0805 package soldered onto the back of the prototype board. Note this version does not use the external LED, but does use battery monitoring input.

Figure 3 - 6 Channel Version (Plane Full House)



2.1 Components

To build the flight controller you will need the items listed in Table 2. These can be obtained from a good hobbyist electronics store, eBay, or Amazon.

Table 2 – 5V Flight Controller Components

Component	Quantity	Comment
Seeed Studio XIAO ESP32S3	1	XIAO ESP32C3 can be used with simple modification to software though its clock speed is lower and may affect control loop times.
GY-521 (MPU6050 breakout)	1	
330 Ohm resistor	8	For servo/motor and receiver; These are only required when interfacing 5V components. A 1s lipo powered model with 3.3V servos/ESC will not require them. The external LED option always requires a series resistor.
10uF 16V/25V capacitor	1	Used to filter/smooth battery voltage and helps remove any electrical voltage spikes in its sampling. If not fitted increase weighted filter in the software.
LED	1	Optional LED for external use. Can use LED_BUILTIN of Arduino.

Component	Quantity	Comment
Sbus Receiver	1	Standard 100Kb Sbus receiver of your choice. Inverted Sbus used, but non-inverted can be used with simple code modification.
30AWG Silicone Wire	-	Multiple colours for wiring of components.
61K9 0.25W	1	If not using exact values software can be modified. Ensure potential divider output does not exceed 3.3V.
17K4 0.25W	1	If not using exact values software can be modified. Ensure potential divider output does not exceed 3.3V.
PCB Prototype Pad Board (2.54mm/0.1 inch pitch)	~42x22mm	Use what you are comfortable with or have. Or build the controller with ‘flying wires’.
1mm double sided foam tape	20mm	Used to soft mount gyro.

3. Arduino IDE Installation

For this installation process it is assumed you will be using the Seeed Studio XIAO ESP32-S3. The installation process should be the same for other ESP32 Arduino boards. This document examples are based upon Arduino version 2.3.

3.1 Arduino IDE Installation

For your operating system download and install the newest version of 2.X.X Arduino IDE:

https://www.arduino.cc/en/software

Once installed launch the Arduino application.

3.2 Arduino Library Installation

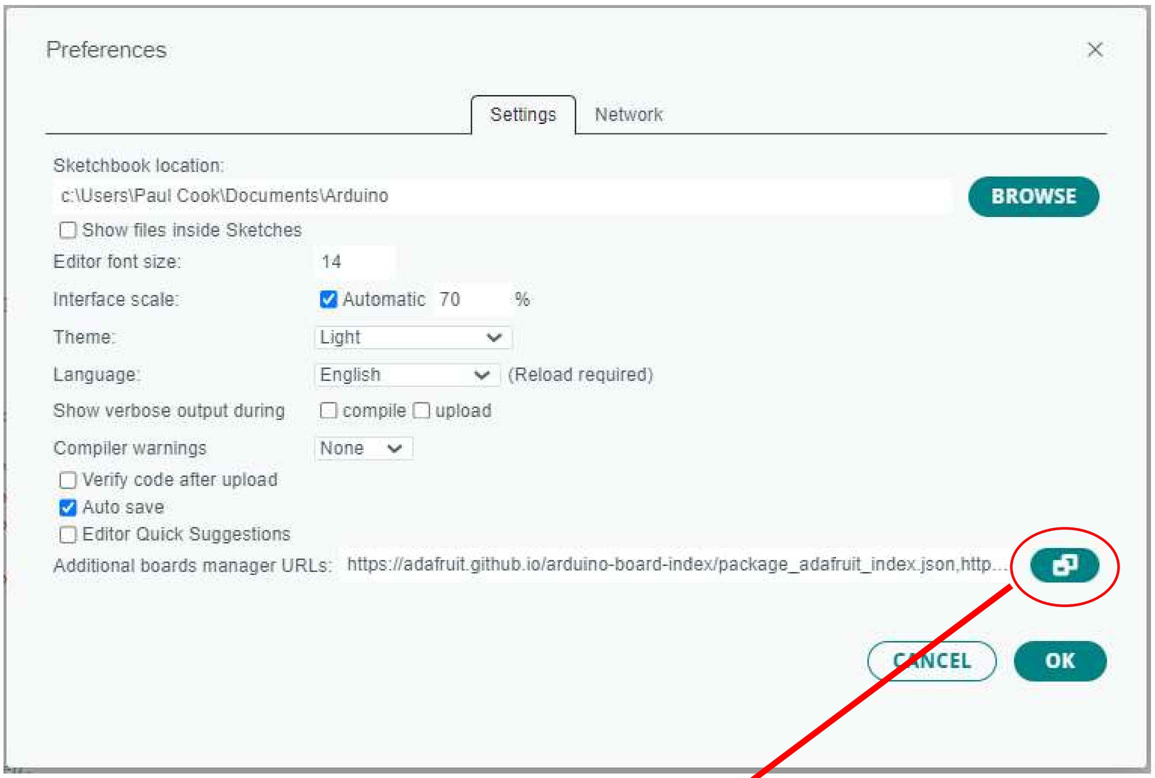
To compile the ESP32 XIAO boards within Arduino you will need to install the 'Espressif Systems ESP32' board package and the 'MPU6050 by Electronic Cats' into your Arduino IDE.

3.2.1 Espressif Systems ESP32 Installation

First you need to map the 3rd party library location . To do this copy the URL given below then navigate to **File > Preferences**, and in the pop-up window fill **'Additional Boards Manager URLs'** with the following URL as shown by Figure 4:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

Figure 4 - Espressif Systems URL Mapping



Preferences

Settings | Network

Sketchbook location:
c:\Users\Paul Cook\Documents\Arduino

- [ ] Show files inside Sketches

Editor font size: 14

Interface scale: [✓] Automatic 70 %

Theme: Light

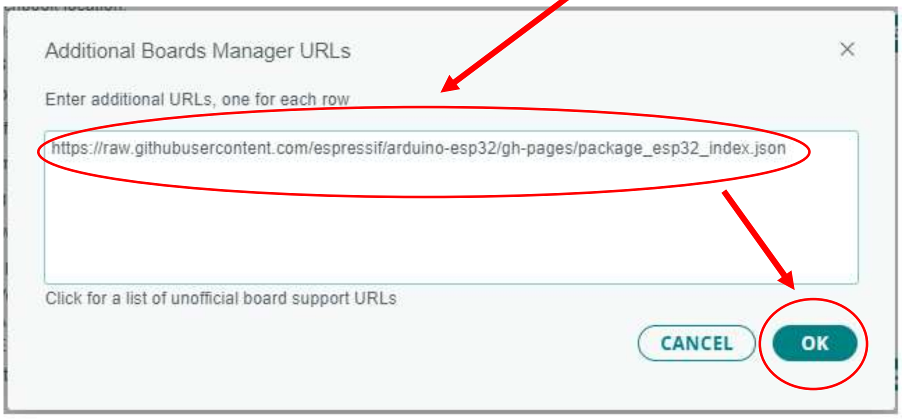
Language: English (Reload required)

Show verbose output during [ ] compile [ ] upload

Compiler warnings None

- [ ] Verify code after upload
- [✓] Auto save
- [ ] Editor Quick Suggestions

Additional boards manager URLs: https://adafruit.github.io/arduino-board-index/package_adafruit_index.json,http...



Additional Boards Manager URLs

Enter additional URLs, one for each row

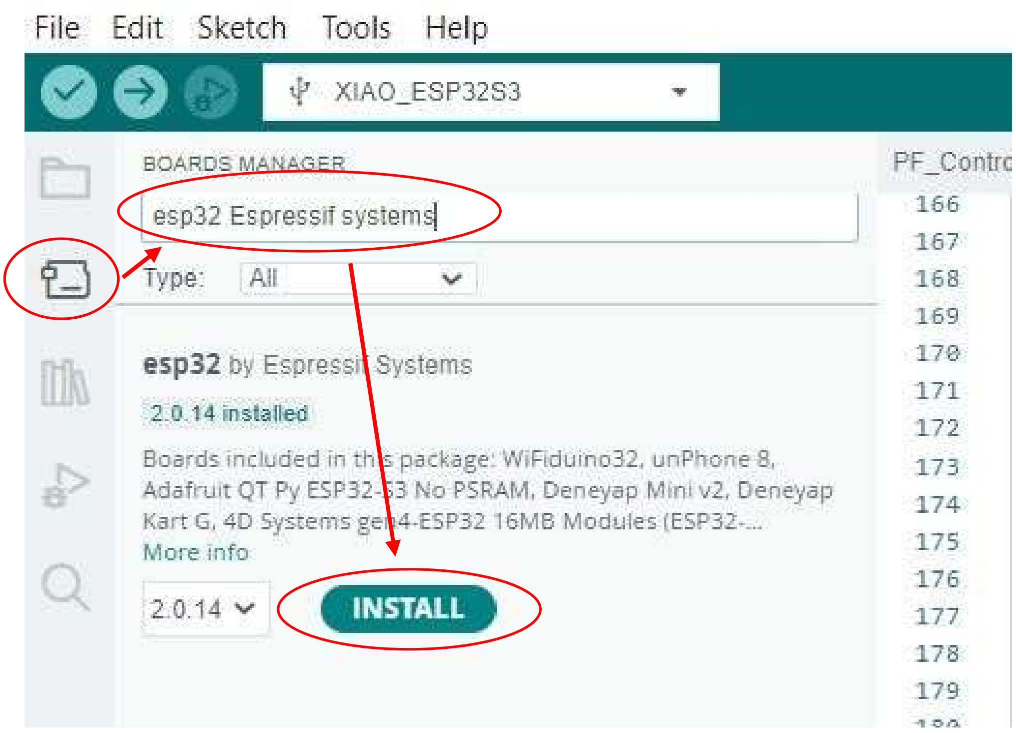
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

Click for a list of unofficial board support URLs

Note: Click **'OK'** to accept and close all pop-up windows.

Now to actually install the 'Espressif Systems ESP32' board package into the Arduino IDE; Navigate to **Boards Manager** icon, type the keywords '**esp32 Espressif Systems**' in the search box, select the latest version and click the install button as indicated by Figure 5.

Figure 5 - ESP32 Espressif Systems Board Package

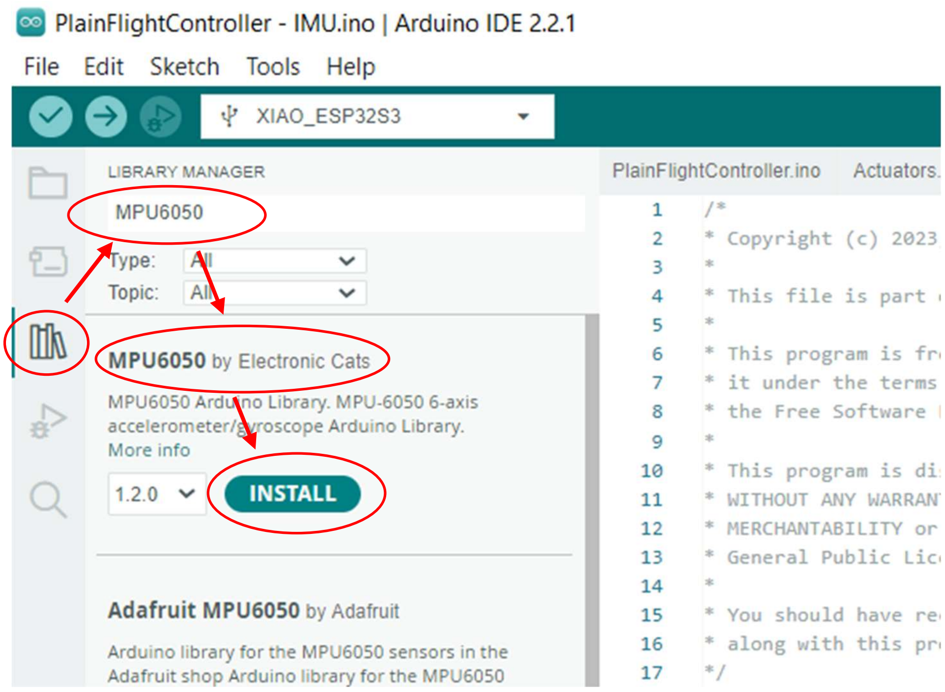


esp32 by Espressif Systems
2.0.14 installed
Boards included in this package: WiFiduino32, unPhone 8, Adafruit QT Py ESP32-S3 No PSRAM, Deneyap Mini v2, Deneyap Kart G, 4D Systems gen4-ESP32 16MB Modules (ESP32-...
More info

3.2.2 MPU6050 By Electronic Cats Installation

In the library manager search ‘MPU6050’, several MPU6050 libraries will appear but ensure you select and install the ‘MPU6050 by Electronic Cats’ as detailed in Figure 1.

Figure 6 - MPU6050 Library Install



MPU6050 by Electronic Cats
MPU6050 Arduino Library. MPU-6050 6-axis accelerometer/gyroscope Arduino Library.
More info

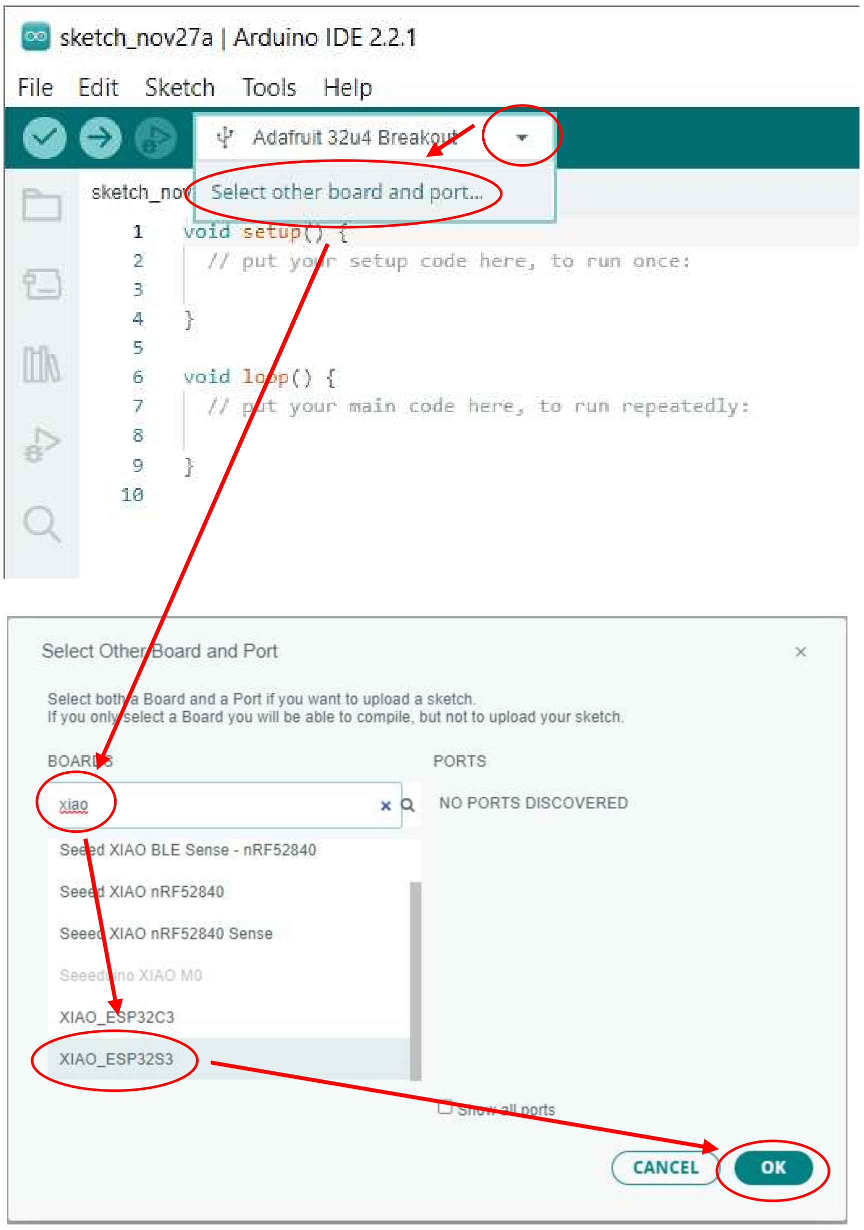
Adafruit MPU6050 by Adafruit
Arduino library for the MPU6050 sensors in the Adafruit shop Arduino library for the MPU6050

```
/*
 * Copyright (c) 2023
 *
 * This file is part
 *
 * This program is fr
 * it under the terms
 * the Free Software
 *
 * This program is di
 * WITHOUT ANY WARRAN
 * MERCHANTABILITY or
 * General Public Lic
 *
 * You should have re
 * along with this pr
 */
```

3.3 XIAO Board Selection

To compile and program the XIAO ESP32S3 you need to select the correct board. Figure 7 details how to navigate to drop down board menu and **‘select other board and port...’**, from the pop up window find and select **’XIAO_ESP32S3’** from the list of boards.

Figure 7 - XIAO ESP32S3 Board Selection



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Select both a Board and a Port if you want to upload a sketch.
If you only select a Board you will be able to compile, but not to upload your sketch.

3.4 Port Selection

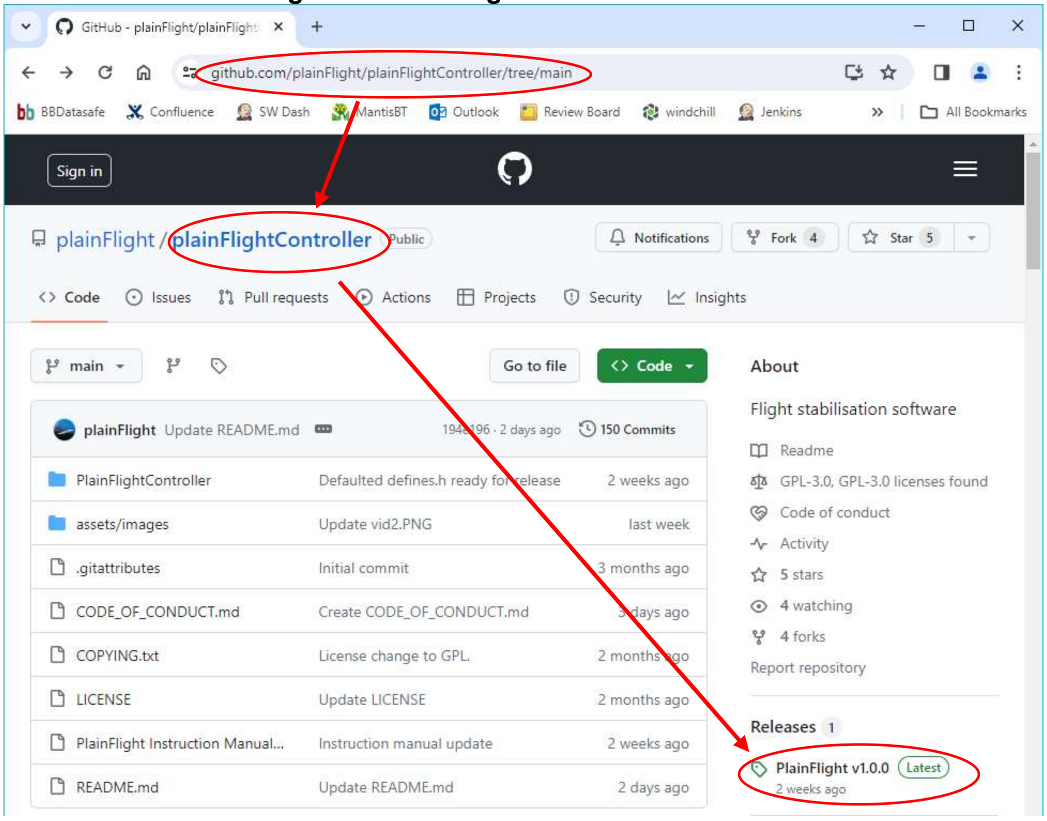
Connect the XIAO_ESP32S3 to your computer via USB. Wait for the drivers to be automatically installed then navigate to **Tools > Port** and select the serial port name of the connected XIAO_ESP32S3.

Note: The XIAO boards often comes up with the wrong name, so select the named board that appears upon USB connection.

4. Downloading PlainFlight Software

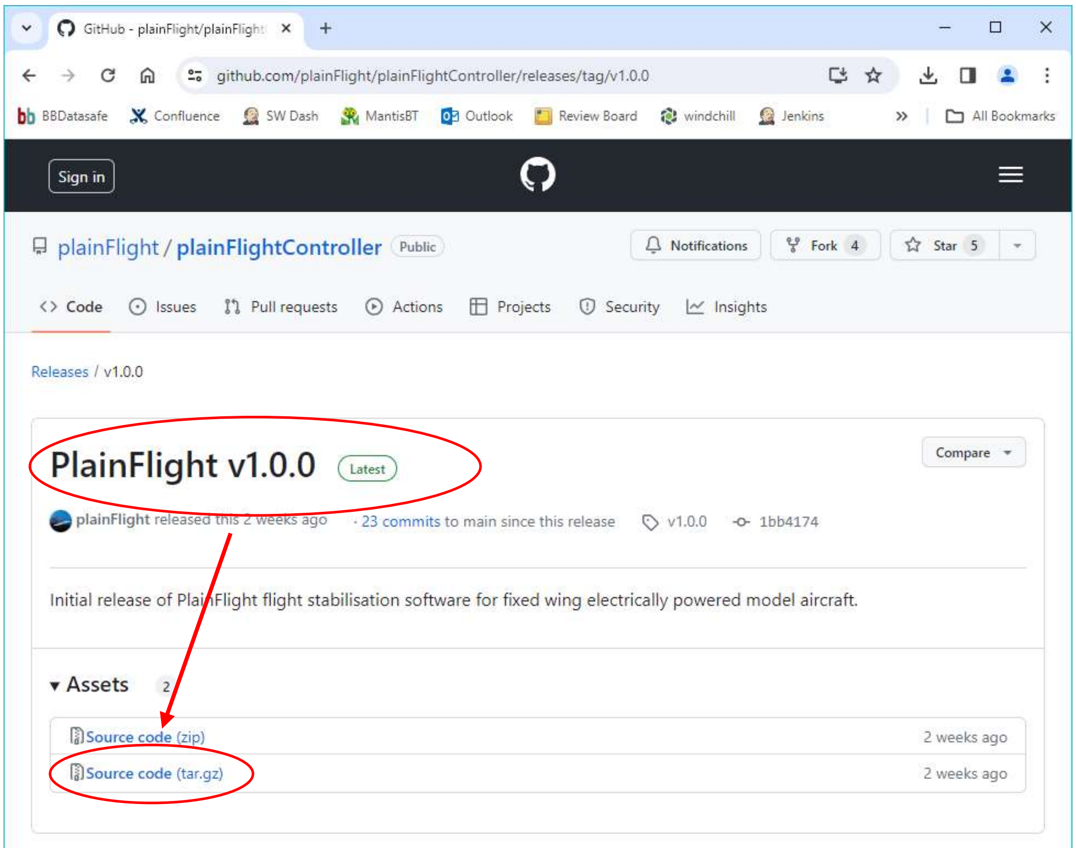
From github.com search 'plainFlightController' to find the project home page. To download the flight controller software, click on the latest release (v1.0.0 shown in example) then select 'Download ZIP'. This will then download the Arduino code, instructions and license to your computer. Figure 8 details the sequence for downloading the software.

Figure 8 - PlainFlight Software Download



plainFlight Update README.md 194e196 · 2 days ago 150 Commits

PlainFlightController	Defaulted defines.h ready for release	2 weeks ago
assets/images	Update vid2.PNG	last week
.gitattributes	Initial commit	3 months ago
CODE_OF_CONDUCT.md	Create CODE_OF_CONDUCT.md	3 days ago
COPYING.txt	License change to GPL.	2 months ago
LICENSE	Update LICENSE	2 months ago
PlainFlight Instruction Manual...	Instruction manual update	2 weeks ago
README.md	Update README.md	2 days ago



PlainFlight v1.0.0 Latest

plainFlight released this 2 weeks ago · 23 commits to main since this release v1.0.0 1bb4174

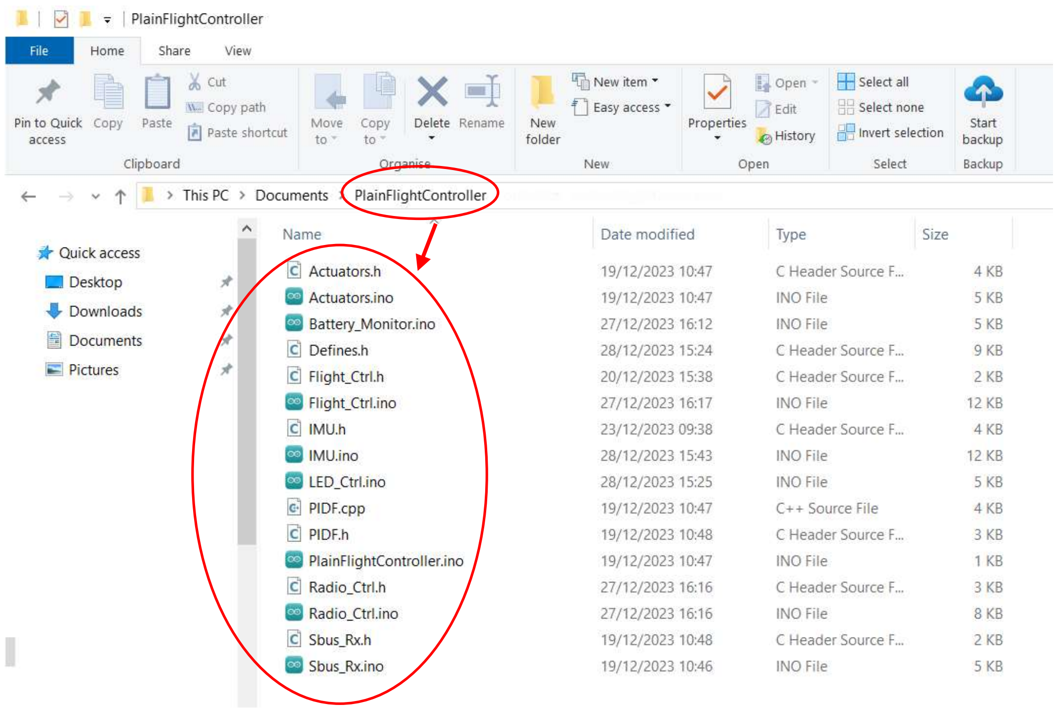
Initial release of PlainFlight flight stabilisation software for fixed wing electrically powered model aircraft.

Assets 2

Source code (zip)	2 weeks ago
Source code (tar.gz)	2 weeks ago

Once downloaded unzip the files at a suitable location. Note the software files should appear inside a folder called ‘PlainFlightController’. If they do not create it and place the software files inside it, as detailed by Figure 9, or it will not compile.

Figure 9 - Software Folder & Files



Name	Date modified	Type	Size
Actuators.h	19/12/2023 10:47	C Header Source F...	4 KB
Actuators.ino	19/12/2023 10:47	INO File	5 KB
Battery_Monitor.ino	27/12/2023 16:12	INO File	5 KB
Defines.h	28/12/2023 15:24	C Header Source F...	9 KB
Flight_Ctrl.h	20/12/2023 15:38	C Header Source F...	2 KB
Flight_Ctrl.ino	27/12/2023 16:17	INO File	12 KB
IMU.h	23/12/2023 09:38	C Header Source F...	4 KB
IMU.ino	28/12/2023 15:43	INO File	12 KB
LED_Ctrl.ino	28/12/2023 15:25	INO File	5 KB
PIDF.cpp	19/12/2023 10:47	C++ Source File	4 KB
PIDF.h	19/12/2023 10:48	C Header Source F...	3 KB
PlainFlightController.ino	19/12/2023 10:47	INO File	1 KB
Radio_Ctrl.h	27/12/2023 16:16	C Header Source F...	3 KB
Radio_Ctrl.ino	27/12/2023 16:16	INO File	8 KB
Sbus_Rx.h	19/12/2023 10:48	C Header Source F...	2 KB
Sbus_Rx.ino	19/12/2023 10:46	INO File	5 KB

5. How To Program

CAUTION: Always remove the flight battery when programming. The motor may start unexpectedly if you do not.

CAUTION: If you have built the 3.3V system never plug USB and 1s lipo flight battery in at the same time as the 5V USB supply will back feed the 4.2V lipo battery.

Open the Arduino project by double clicking 'PlainFlightController.ino' file. The Arduino IDE should open and all of the software files will appear as tabs inside the IDE.

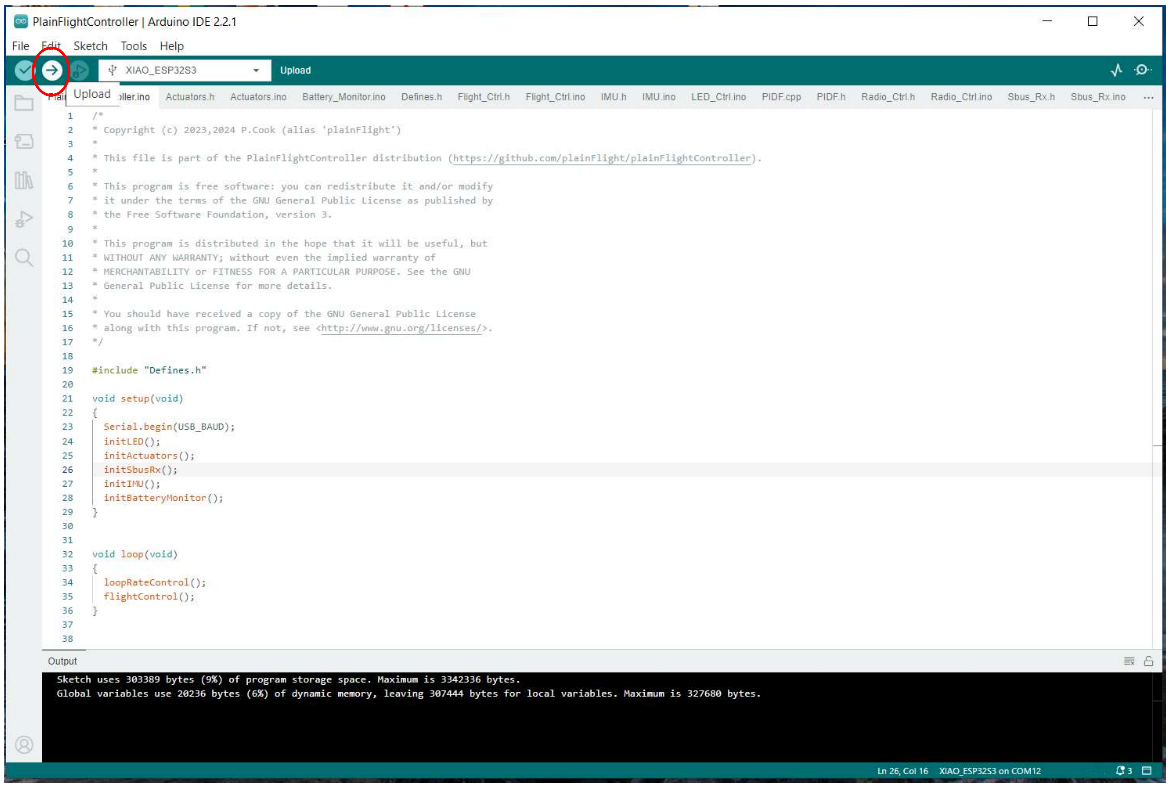
Ensure the correct target board is selected (XIAO_ESP32S3) as detailed in section 3.3.

Plug in your XIAO ESP32S3 to your computer via the USB and select the correct port as detailed in section 3.4.

You can now select the upload button to program your XIAO ESP32S3 with PlainFlight software. Figure 10 shows how the Arduino IDE should appear with al software files, correct board selected, and upload button highlighted.

The Upload can take several minutes, especially if it's the first time the software has been compiled, a progress bar will be given to indicate the upload status.

Figure 10 - PlainFlight Upload



```
/*
* Copyright (c) 2023,2024 P.Cook (alias 'plainFlight')
*
* This file is part of the PlainFlightController distribution (https://github.com/plainFlight/plainFlightController).
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, version 3.
*
* This program is distributed in the hope that it will be useful, but
* WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
* General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program. If not, see <http://www.gnu.org/licenses/>.
*/

#include "Defines.h"

void setup(void)
{
  Serial.begin(USB_BAUD);
  initLED();
  initActuators();
  initSbusRx();
  initIMU();
  initBatteryMonitor();
}


void loop(void)
{
  loopRateControl();
  flightControl();
}
```

```
Sketch uses 303389 bytes (9%) of program storage space. Maximum is 3342336 bytes.
Global variables use 20236 bytes (6%) of dynamic memory, leaving 307444 bytes for local variables. Maximum is 327680 bytes.
```

Should you XIAO board fail to program, disconnect power/USB and hold down the boot button on the XIAO board. Now reapply power to enter boot mode ready for programming.

6. How To Configure Your Model

The flight controller is configured via the defines.h file and the #defines contained within it. The file is well commented, but the following sections describe these in detail.

6.1 DEBUG

Debug defines are used to help problem solve the flight controller. It is recommended that each of these is enabled one at a time to confirm correct operation of the flight controller following its construction. Use Arduino IDE 'Serial Monitor' to view the data, Table 3 details what debug data you can view.

Table 3 - Debug Data Output

Debug	Description
SBUS_DEBUG	Outputs raw Sbus data. Confirms correct wiring and that Sbus data is being received.
DEBUG_RADIO_COMMANDS	Outputs refactored Sbus data that the flight controller uses.
DEBUG_GYRO_DATA	Outputs 6-axis data from the gyro. Can be used to confirm the gyro is connected and working correctly.
DEBUG_PID	Outputs PIDF data. Caution, only instantiate one PIDF class at a time for this to work correctly.
DEBUG_GYRO_CALIBRATION	Outputs data relevant to gyro calibration. If you have a noisy gyro then CALIBRATE_MAX_MOTION may need to increase.
DEBUG_MADGWICK	Outputs 3-axis IMU data from the Madgwick filter. This data is used for self levelled flight mode.
DEBUG_BATTERY_VOLTS	Outputs battery volts as a float. Use to calibrate battery voltage correctly with a voltmeter and modification of ADC_MULTIPLIER.
DEBUG_LOOP_RATE	Outputs the loop rate as float in seconds. You should see a value of 0.001 (up to 0.001002 is normal). If using ESP32C3 with modified code you may not achieve a 1ms loop rate, so increase LOOP_RATE_US.
DEBUG_FLIGHT_STATE	Outputs the flight mode, values from 0 to 4 will be seen when changing arm and mode switches.
DEBUG_SERVO_MIXER	Outputs servos 1 to 4 as signed integers. May help with debugging cross mixing of channels.
DEBUG_MOTOR_MIXER	Outputs motors 1 and 2 as signed integers. May help with debugging cross mixing of channels.

6.2 Model Mixer

There is a choice of 5 model types to choose from and each of these have differential thrust option if using 2 motors.

The model type is set via the MIXER_ defines. Uncomment the mixer type you require for your model and note the servo numbering that relates to the control surface.

6.2.1 MIXER_PLANE_FULL_HOUSE

This mixer offers 2 servo for aileron/flap functions, rudder and elevator, single or dual motor. Flaps have 3 positions and are controlled via Aux1.

Figure 11 - MIXER_PLANE_FULL_HOUSE

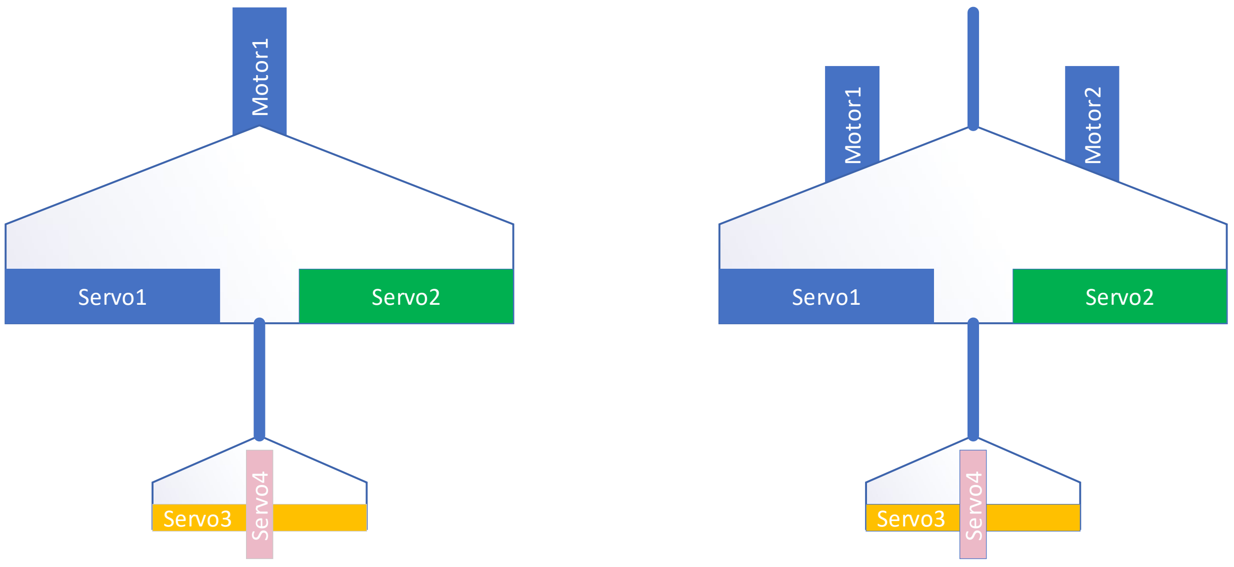


Table 4 - Plane Full House Actuators

Actuator	Function	Comment
Motor 1	Throttle	
Motor 2	Throttle	If fitted on model.
Servo 1	Left aileron/flap	Aux1 operates 3 position flap
Servo 2	Right aileron/flap	Aux1 operates 3 position flap
Servo 3	Elevator	
Servo 4	Rudder	

Note: To enable differential thrust uncomment the define USE_DIFFERENTIAL_THRUST.
Note: To enable Oneshot125 for BLHeli type ESCs uncomment the define USE_ONESHOT125_ESC.
Note: Pusher configuration has no impact upon the flight controller mixer.

6.2.2 MIXER_PLANE_FULL_HOUSE_V_TAIL

This mixer offers 2 servo for aileron/flap functions, and mixed rudder/elevator for a V-tail model. Single or dual motors can be used and flaps have 3 positions and are controlled via Aux1.

Table 5 - MIXER_PLANE_V_TAIL

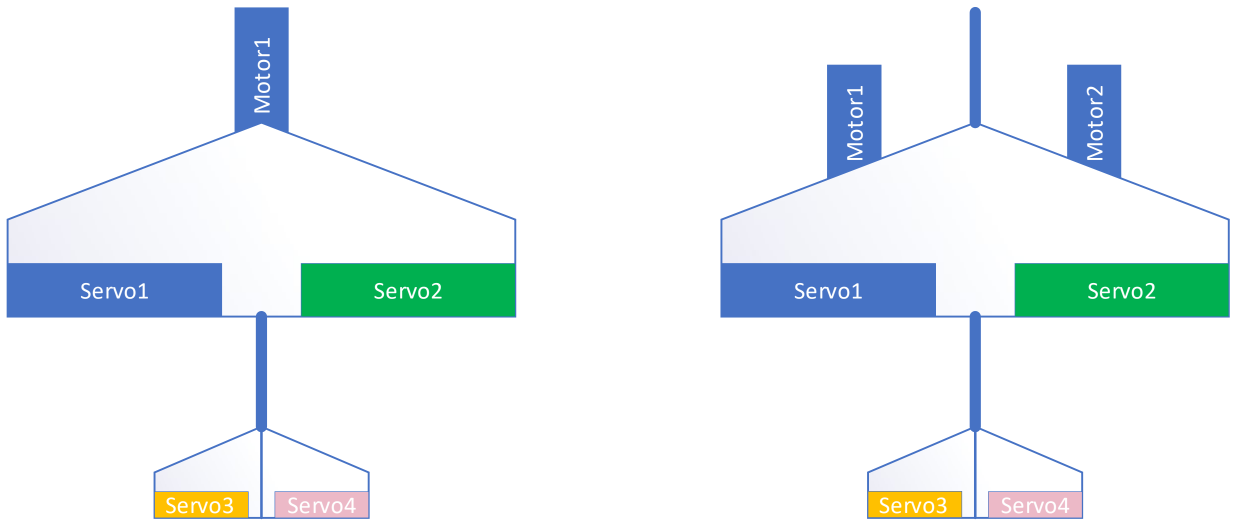


Table 6 - Plane V-Tail Actuators

Actuator	Function	Comment
Motor 1	Throttle	
Motor 2	Throttle	If fitted on model.
Servo 1	Left aileron/flap	Aux1 operates 3 position flap
Servo 2	Right aileron/flap	Aux1 operates 3 position flap
Servo 3	Elevator/rudder	
Servo 4	Elevator/rudder	

Note: To enable differential thrust uncomment the define USE_DIFFERENTIAL_THRUST.
Note: To enable Oneshot125 for BLHeli type ESCs uncomment the define USE_ONESHOT125_ESC.
Note: Pusher configuration has no impact upon the flight controller mixer.

6.2.3 MIXER_PLANE_RUDDER_ELEVATOR

This mixer offers rudder and elevator control with single or dual motors. Use this for rudder/elevator control a flap functions are disabled.

Figure 12 - MIXER_PLANE_RUDDER_ELEVATOR

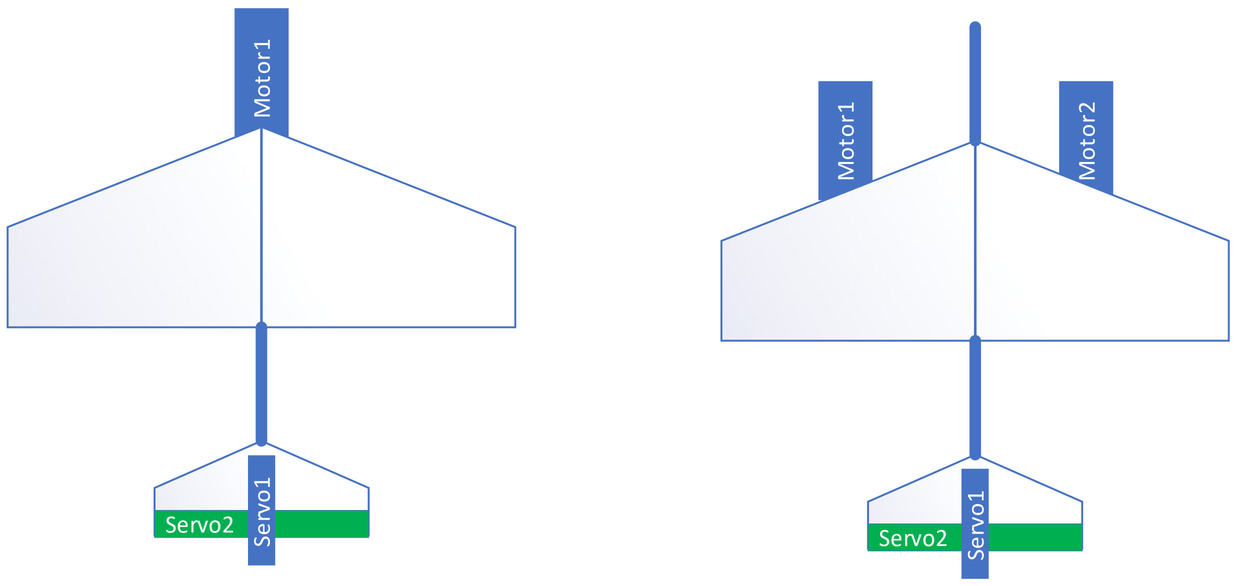


Table 7 - Plane Rudder Elevator Actuators

Actuator	Function	Comment
Motor 1	Throttle	
Motor 2	Throttle	If fitted on model.
Servo 1	Rudder	
Servo 2	Elevator	
Servo 3	N/A	Spare actuator could be repurposed with software modification.
Servo 4	N/A	Spare actuator could be repurposed with software modification.

Note: To enable differential thrust uncomment the define USE_DIFFERENTIAL_THRUST.
Note: To enable Oneshot125 for BLHeli type ESCs uncomment the define USE_ONESHOT125_ESC.
Note: Pusher configuration has no impact upon the flight controller mixer.

6.2.4 MIXER_PLANE_V_TAIL

This mixer offers rudder and elevator control with single or dual motors. Use this for rudder/elevator control as flap function is disabled.

Figure 13 - MIXER_PLANE_V_TAIL

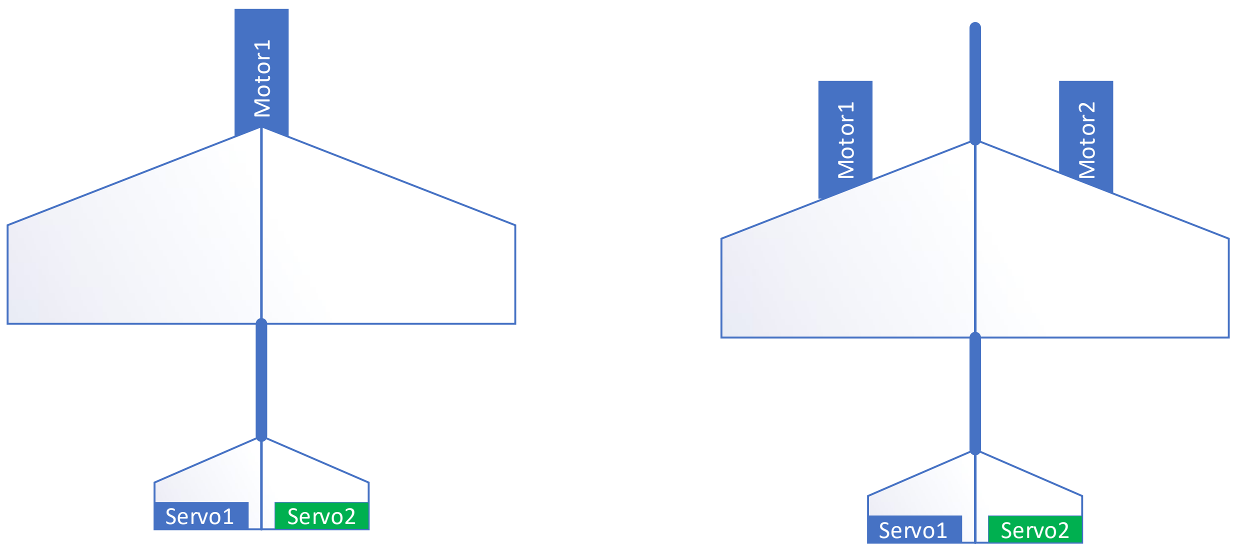


Table 8 - Plane V-Tail Actuators

Actuator	Function	Comment
Motor 1	Throttle	
Motor 2	Throttle	If fitted on model.
Servo 1	Left Rudder/elevator	
Servo 2	Right Rudder/elevator	
Servo 3	N/A	Spare actuator could be repurposed with software modification.
Servo 4	N/A	Spare actuator could be repurposed with software modification.

Note: To enable differential thrust uncomment the define USE_DIFFERENTIAL_THRUST.
Note: To enable Oneshot125 for BLHeli type ESCs uncomment the define USE_ONESHOT125_ESC.
Note: Pusher configuration has no impact upon the flight controller mixer.

6.2.5 MIXER_FLYING_WING

This mixer offers elevon mixing of roll and elevator controls for flying wings. Rudder control is also available if fitted upon the model. As standard single or dual motor control is given.

Figure 14 – MIXER_FLYING_WING

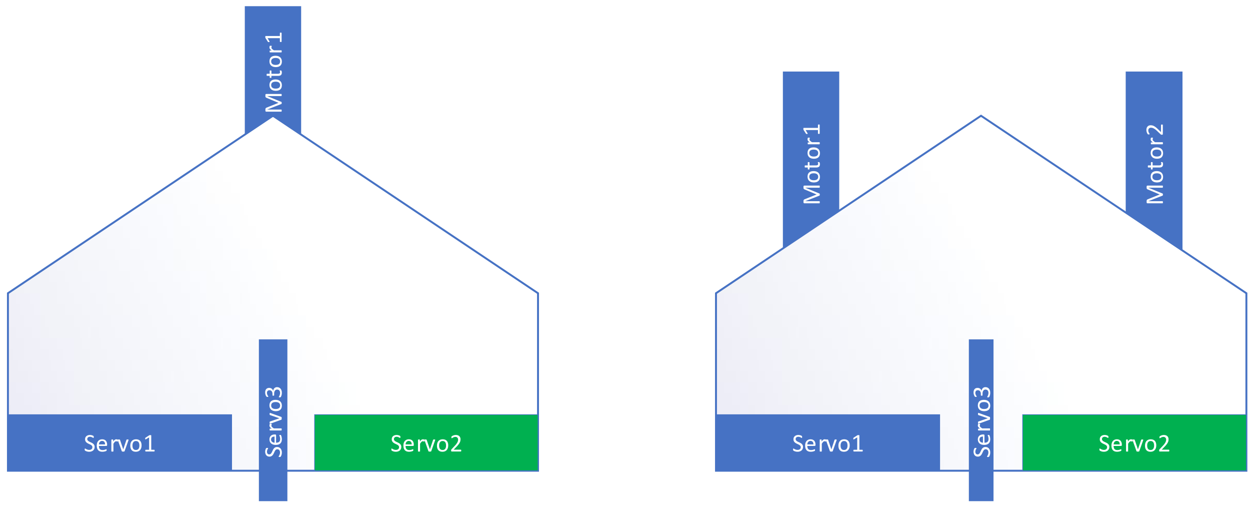


Table 9 - Plane Rudder Elevator Actuators

Actuator	Function	Comment
Motor 1	Throttle	
Motor 2	Throttle	If fitted on model.
Servo 1	Left elevon	
Servo 2	Right elevon	
Servo 3	Rudder	
Servo 4	N/A	Spare actuator could be repurposed with software modification.

Note: To enable differential thrust uncomment the define USE_DIFFERENTIAL_THRUST.
Note: To enable Oneshot125 for BLHeli type ESCs uncomment the define USE_ONESHOT125_ESC
Note: Pusher configuration has no impact upon the flight controller mixer.

6.3 GYRO_FS_SEL_XXX

Sets the maximum degrees per second the gyro will output.

Function	Comment
GYRO_FS_SEL_250	Maximum allowed rotation rate of 250 degrees/second. Recommended for models that can do mild/pattern aerobatics.
GYRO_FS_SEL_500	Maximum allowed rotation rate of 500 degrees/second. Recommended for highly aerobatic models. Using this setting will lower resolution of the sensor output and may reduce performance for majority of flying manoeuvres.

6.4 SINK_LED

The XIAO ESP32S3 has an onboard LED that will display the flight state via sequences of flashes. There is also the option to use an external LED for the same purpose should the flight controller be hidden inside the fuselage.

Depending how you wire the external LED you will need to tell the flight controller if it needs to sink the LED (output low) to illuminate, or source the LED (output high) to illuminate. To sink the LED make sure SINK_LED is uncommented, or comment out to source the LED.

6.5 REVERSE_PITCH_CORRECTIONS

When in self-levelled or rate modes if you find that the pitch correction is in the wrong sense then use this define to change the direction to the correct sense.

Note: If you find that control surfaces are correcting in the right sense, but Tx commands give wrong sense then reverse the output direction within you transmitter.

6.6 REVERSE_ROLL_CORRECTIONS

When in self-levelled or rate modes if you find that the roll correction is in the wrong sense then use this define to change the direction to the correct sense.

Note: If you find that control surfaces are correcting in the right sense, but Tx commands give wrong sense then reverse the output direction within you transmitter.

6.7 REVERSE_YAW_CORRECTIONS

When in self-levelled or rate modes if you find that the yaw correction is in the wrong sense then use this define to change the direction to the correct sense.

Note: If you find that control surfaces are correcting in the right sense, but Tx commands give wrong sense then reverse the output direction within you transmitter.

6.8 SERVO_REFRESH_xxxHZ

The ESP32 LEDC PWM timer peripheral is set to 14-bit resolution. This allows a range of servo PWM/PPM refresh rates to be set from 50Hz up to 2KHz and therefore allows use of analogue or digital servos. Check the manufacturers data upon your servos and set the appropriate refresh rate for your servos via the SERVO_REFRESH_ defines.

Note: Setting the refresh rate too high for your servos may permanently damage them.

Due to how the LEDC PWM peripheral works, the lower the refresh rate results in lower resolution output for the servo i.e. lower resolution equals fewer servo steps over full travel of the servo. Table 10 details the servo resolution relationship and clearly shows that digital servos will offer better precision.

Table 10 - Servo Refresh Verses Resolution

Refresh Rate Define	Resolution/Servo Steps
SERVO_REFRESH_50HZ	819
SERVO_REFRESH_100HZ	1638
SERVO_REFRESH_150HZ	2458
SERVO_REFRESH_250HZ	4096
SERVO_REFRESH_300HZ	4915
USING_ONESHOT125_ESC (2KHz implementation)	4096 (ESC output)

6.9 USE_ONESHOT125

When this is enabled motor outputs will use the Oneshot125 protocol as found on BLHeli ESCs. The oneshot125 protocol is a fast 4KHz PWM/PPM output with pulse width ranging from 125-250us.

This protocol allows faster and more accurate updates to the motors and when used for differential thrust aircraft and may increase stability in prop-hanging type manoeuvres.

Note: While Oneshot125 can output at up to ~4KHz PlainFlight only output at 2KHz, this is done as the maths for the PWM/PPM works out nicely at ~12-bit resolution (4096 steps). Also, the main flight controller loop runs at 1KHz so there is no benefit to update above 1Khz, or at the maximum of 4KHz.

Note: BlHeli ESCs may have had different end points set. Either use the BlHeli App to configure or adjust the ONESHOT125_MIN_TICKS and/or ONESHOT125_MAX_TICKS defines in Actuators.h.

6.10 TX_DEAD_BAND_xxx

Radio control transmitters may have a bit of jitter or inaccuracy around centred positions. This dead band can be used to mask this jitter or inaccuracy.
If you find that certain control surfaces are drifting in rate mode with Tx sticks centred, then try increasing the dead band for that channel.

Note: Control surface drift could also be caused by a poor power on gyro calibration i.e. model was moving during calibration, but not by enough to trip a calibration reset.

6.11 TRIM_SERVOx

Set all control surfaces to be mechanically neutral with the servo horn perpendicular to the servo case. If you find that the servo horn does not align perpendicular to the servo case due to the spline not lining up, then use these trims to adjust the perpendicular (centre position) of the servo.

Note: trims are a signed integer value (+ve/-ve) and a value of 10 gives approximately 1 degree of servo travel.

6.12 TRIM_LEVELLED_xxx

When flying in self-levelled mode; if the model does not fly straight or level then use these trim values to correct for straight and levelled flight.

Note: Trims are floating point and represent degrees.

6.13 USE_DIFFERENTIAL_THROTTLE

If your model has twin motors then by uncommenting this allows you to use differential thrust via rudder Tx commands. Differential throttle will be gyro stabilised in rate and self-levelled modes.

6.14 USE_HEADING_HOLD

When enabled a heading hold function will be mapped to the rudder. This is done by applying i gain to the PIDF yaw calculation when heading hold switch is enabled.

The intention of heading hold is to assist tracking during manoeuvres such as run off ground take off's, vertical climbs, loops and prop hanging.

Note: Comment out if you have a transmitter with only 6 channels.

6.15 USE_HEADING_HOLD_WHEN_YAW_CENTRED

When enabled a heading hold function will be mapped to the rudder. It will only take affect when the transmitter yaw stick is centred. You may need to increase yaw dead band to make it work reliably.

Note: Comment out if you have a transmitter with only 6 channels.

6.16 MAX_xxx_RATE_DEGS_x100

These defines set the maximum degrees per second that the model can do for each axis. Note that when using gyro set to 250 degrees per second do not exceed a value of 230 degrees per second. If you require higher rates then set the gyro to 500 degrees per second via GYRO_FS_SEL_500. It should be noted that 250 degrees per second is sufficient for most models and is the recommended setting.

Note: The values given are unsigned integer and multiplied by 100 e.g. you require 230 degrees per second so 230x100 = 23000.

Note: If you change the gyro scaling this will affect you gains, scale all gains accordingly.

6.17 MAX_xxx_ANGLE_DEGS_x100

These defines set the maximum angle in degrees that the model can pitch and roll when in self-levelled mode.

Note: The values given are unsigned integer and multiplied by 100 e.g. you require 45 degrees of bank, so 45x100 = 4500.

6.18 FAILSAFE_xxx_ANGLE

Should a failsafe occur when flying it will enter self-levelled mode and cut throttle. Failsafe will be commanded form the Sbus protocol, or when Sbus comms is lost. Should failsafe occur these defines allow you set the levelled trim when in failsafe.

It is recommended that you set a roll bank of a few degrees and pitch up slightly to cause a gently spiral decent.

6.19 USE_FLAPS

When using MIXER_PLANE_FULL_HOUSE or MIXER_PLANE_FULL_HOUSE_V_TAIL that has two aileron servos you can enable the flaps function. Comment out of you do not wish to use flaps or have a transmitter with only 6 channels.

6.20 USE_LOW _VOLTS_CUT_OFF

When enabled battery voltage will be monitored and the system will react to low battery volts by actively reducing the throttle output(s) and ultimately cutting the throttle.

Motor power reduction starts when the battery falls to 3.5V per cell, and complete throttle cut off occurs at 3.3V per cell. If you are using lithium ion cells then these values will need to be changed to 2.7V and 2.5V respectively.

This option is only recommended when using BlHeli type ECSs where low voltage detection is not part of the ESC software.

Note: For this to operate correctly you must have the potential divider hardware connected to D10 analogue input.

6.21 CALIBRATE_ESCS

CAUTION: ALWAYS REMOVE PROPELLER(S) WHEN CALIBRATING ESC(S). YOU RISK SERIOUS INJURY IF YOU DO NOT.

When enabled, at power on the flight controller will output maximum throttle value and hold for CALIBRATE_HOLD_TIME, then set minimum throttle value. This should calibrate the ESC(s) full range to match the flight controller output.

Note: When calibrating you will need to use the flight battery to power the ESC(s) as the 5V USB supply may not power the ESC. Depending upon your ESC(s) you may need to reduce CALIBRATE_HOLD_TIME to prevent it from entering programming mode.

Once calibrating has finished, code execution is halted. You will need to comment out CALIBRATE_ESCS and reprogram the flight controller.

Note: For the 3.3V version this should work from USB power.

6.22 USB_BAUD

You should not need to change this but can be lowered if you are having USB connection issues.

6.23 IO Pin Allocation

You should not need to change these unless you are modifying the software for your purposes.

Seeed Studio XIAO ESP32S3 I/O	ESP32S3 GPIO	PlainFlight IO Allocation
D0/A0	GPIO1	Servo1
D1/A1	GPIO2	Servo2
D2/A2	GPIO3	Servo3
D3/A3	GPIO4	Servo4
D4/I2C_SDA	GPIO5	I2C SDA
D5/I2C_SCL	GPIO6	I2C SCL
D6/TX	U0TXD	External LED
D7/RX	U0RXD	Sbus
D8/A8/SCK	GPIO7	Motor1
D9/A9/MISO	GPIO8	Motor2
D10/A10/MOSI	GPIO9	Battery Monitor

Note: Should you use an ESP32-C3 then you will need to reallocate the SBUS_TX_PIN to a spare unrouted IO.

7. Transmitter Configuration

To obtain full functionality of PlainFlight you will need an 8-channel transmitter. Channels 1 to 4 are your primary flight controls of throttle, aileron, elevator, rudder and need to be set to 100% travel. Auxiliary channels are configured for switch inputs with an example shown in Figure 15, switches must also be set for 100% travelFigure 1.

Figure 15 - Typical Tx Switch Setup

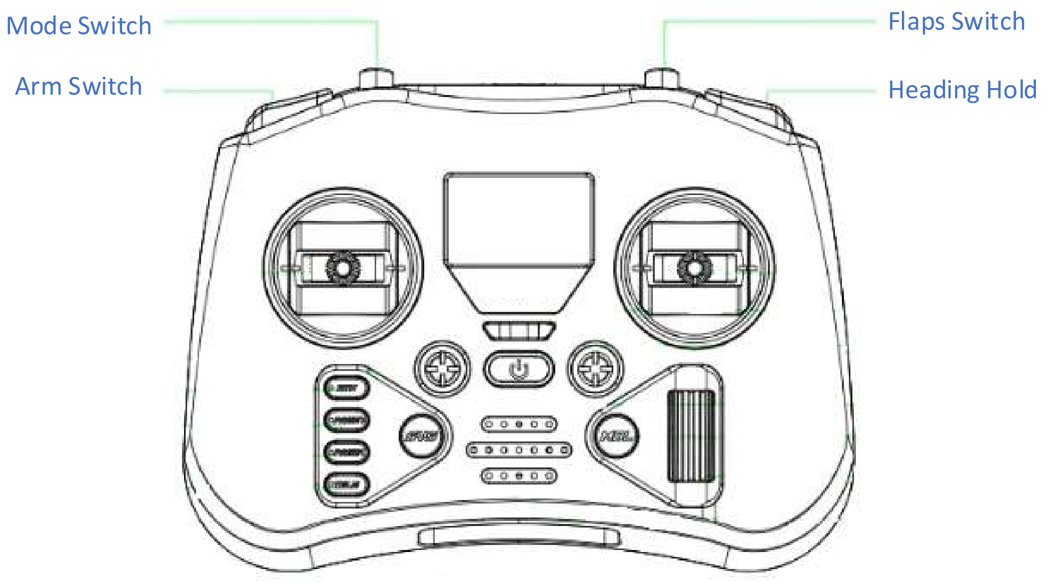


Table 11 details the requirements of these switch inputs so that flight controller functions operate correctly.

Table 11 - Tx Switch Requirements

Switch	Type	Tx Channel	Usage
Arm Switch	2 position latching	5	>1500us = armed
Mode Switch	3 position	6	<1250us = pass through >1250us & <1750us = rate mode >1750us = levelled mode
Flaps Switch	3 position	7	<1250us = no flaps >1250us & <1750us = flaps 1 >1750us = flaps 2
Heading Hold Switch	2 position (latching if preferred)	8	>1500us = heading hold active.

Note: Flight controller will not arm if throttle is high.
Note: Failsafe will be entered if Tx reception is not initiated or is lost.

7.1 Trims

It is essential that your transmitter trims are set to their neutral position, if they are not when you enter rate or levelled modes the flight controller will see the trim(s) offset as a demand and make the model pitch, roll, or yaw.
You will need to set servo control arms/horns perpendicular to the servo body and adjust the linages mechanically to make the model fly straight and level.

7.2 Arming Switch

Like any flight controller it will need to be armed before the motor(s) will operate. Whilst disarmed all servos will operate in pass through mode i.e. no stabilisation will be applied. Arming is controlled via channel 5.

7.3 Mode Switch

Once armed the flight controller operates in 3 main modes of pass-through, rate and self levelled. These modes are controlled via channel 6.

7.3.1 Pass Through Mode

When in pass-through mode the transmitter commands are passed directly through to the servos and motor(s). Should you experience any control problems in flight switch back to pass-through mode to disable all PIDF calculations.

7.3.2 Rate Mode

When in rate mode the 3 axis gyro is used to apply stabilisation to roll, pitch and yaw axes. Transmitter stick commands are converted to degrees/second and PIDF calculations are applied for each axes based upon stick commands and gyro data, these calculations depend upon the gains set for each corresponding axis.

Note: Setting gains too high will likely make the model suffer from oscillations and could make it uncontrollable.

7.3.3 Levelled Mode

Self-levelled mode using both the gyro and accelerometer; this mode combines data via a fusion algorithm known as a Madgwick filter which computes the IMUs attitude relative to gravity. Transmitter stick commands are converted to a desired angle in degrees and PIDF calculations are applied for pitch and roll axes based upon stick commands and IMU attitude, these calculations depend upon the gains set for each corresponding axis.

7.4 Flaps Switch

When enabled in the software a 3-position switch driven b channel 7 will give either, no flaps, half flaps, or full flaps. Currently half flaps result in 25% of servo travel and full flaps 50% of servo travel.

7.5 Heading Hold Switch

When enabled in the software a 2-position switch driven by channel 8 will give a heading hold function to the rudder (yaw axis). This is done by applying ‘I’ gain to the PIDF yaw calculation when heading hold switch is enabled.

Caution: Do not attempt to fly with this permanently enabled as the heading hold will fight an aileron turn.

The intention of heading hold is to assist tracking during manoeuvres such as run off ground take off’s, vertical climbs, loops and prop hanging and knife-edge flight. The user will need to switch this heading hold function on and off as and when needed during flight manoeuvres.

8. How To Test

Following your build of your PlainFlight controller you will need to test the following items.

Note: If at any time your flight controller refuses to program, remove USB from you computer, press and hold the boot button on the XIAO ESP32 then insert the USB and after 1 second of it being powered release the boot button. Now reprogram.

Caution: Ensure propeller(s) are removed when connecting USB with flight battery in circuit.

Caution: The 1s circuit will back feed 5V USB power to the ESC's, ensure ESC's can tolerate 5V and remove propeller(s).

8.1 DEBUG_GYRO_CALIBRATION

In defines.h uncomment DEBUG_GYRO_CALIBRATION and reprogram your flight controller. Unplug your controller from the USB at the computer and open the Arduino Serial Monitor.

Note: When the flight controller gyro calibrates at power on, small or sustained movements will cause it to halt and to retry calibration. Calibration will not complete until the flight controller is still.

Place the flight controller on a flat surface, ensure it is still and plug the USB into your computer.

On Arduino's Serial Monitor you should see messages similar to:

```
MPU6050 initialization successful
Calibration...
Calibration reset !
Calibration reset !
Calibration reset !
Calibration complete...
x: 120   y: -91   z: -46
```

Note: you may see more or less 'Calibration reset !' messages depending upon how still your flight controller was at power on.
Note: Your X, Y and Z calibration values will be different.

If you do not see these messages, then you may have a power or I2C wiring issue between the ESP32 XIAO and the MPU6050.

Alternatively, if you have a noisy gyro you may need to increase CALIBRATE_MAX_MOTION to a larger value. If you do need to increase this value, consider changing the MPU6050 for another. A lot of these MPU6050's are Chinese clones and the quality varies.

Note: The calibration state is only exited upon successful calibration. Should you be moving the flight controller or model during calibration then it will not exit, and the flight controller cannot be armed. Calibration state is indicated by a long 3 second LED flash.

8.2 DEBUG_GYRO_DATA

In defines.h comment out DEBUG_GYRO_CALIBRATION and uncomment DEBUG_GYRO_DATA then reprogram your flight controller.

The Serial Monitor should now be continually displaying gyro and accelerometer data. Confirm that rotating/tilting your flight controller in X, Y and Z axes causes the corresponding axis data to change. On your Arduino Serial Monitor you should see data similar to:

```
ax:0.21,ay:0.00,    az:0.76,    gx:-0.18,    gy:0.04,    gz:-0.21
ax:0.21,ay:0.00,    az:0.76,    gx:-0.18,    gy:0.05,    gz:-0.21
```

```
ax:0.21,ay:0.00,    az:0.76,    gx:-0.20,    gy:0.05,    gz:-0.22
ax:0.21,ay:0.00,    az:0.76,    gx:-0.20,    gy:0.04,    gz:-0.23
ax:0.21,ay:0.00,    az:0.75,    gx:-0.20,    gy:0.04,    gz:-0.24
```

Note: All data should change as you tilt and rotate the flight controller.

In the unlikely case that some data is stuck and does not change you likely have a bad MPU6050.

8.3 DEBUG_MADGWICK

If the gyro tests in 8.1 and 8.2 are successful, then there is no need to perform this test. However, it may be of interest to you.

In defines.h comment out DEBUG_GYRO_DATA and uncomment DEBUG_MADGWICK then reprogram your flight controller.

The Serial Monitor should now be continually displaying roll, pitch and yaw data. Confirm that rotating/tilting your flight controller in X, Y and Z axes causes the corresponding axis data to change between 0 and +-360 degrees. On your Arduino Serial Monitor you should see data similar to:

```
Roll: 0.04, Pitch: -15.46, Yaw:0.00
Roll: 0.04, Pitch: -15.46, Yaw:0.00
Roll: 0.05, Pitch: -15.46, Yaw:-0.00
Roll: 0.05, Pitch: -15.46, Yaw:-0.00
Roll: 0.06, Pitch: -15.46, Yaw:-0.00
Roll: 0.06, Pitch: -15.46, Yaw:-0.00
Roll: 0.07, Pitch: -15.46, Yaw:-0.01
Roll: 0.07, Pitch: -15.45, Yaw:-0.01
```

Comment out DEBUG_MADGWICK then reprogram your flight controller to disable the comms output.

8.4 DEBUG_LOOP_RATE

There is no need to perform this test unless you have modified the software for your own purposes and wish to check that your changes have not affect the 1KHz loop rate.

In defines.h uncomment DEBUG_LOOP_RATE then reprogram your flight controller.

The Serial Monitor should now be continually displaying a floating-point number of value 0.001000, it may fluctuate up to 0.001002, this is ok. Below is data captured from the Serial Monitor:

```
0.001000
0.001001
0.001001
0.001000
0.001000
0.001001
```

Note: 1/0.001000 = 1000Hz (1KHz) loop rate.

Note: The XIAO ESP32-C3 will achieve a loop rate of 1KHz, but additional code modifications will likely slow it down due to its lower clock speed.

Re-comment DEBUG_LOOP_RATE then reprogram your flight controller to disable the comms output.

8.5 DEBUG_SBUS

In defines.h uncomment DEBUG_SBUS then reprogram your flight controller.

With you transmitter on and having bound the transmitter to your receiver the Serial Monitor should now display 18 channels of Sbus data similar to:

```
172   995   992   992   1810   992   172   1810   0   0   0   0   0   0   0   0   0   0
172   995   992   992   1810   992   172   1810   0   0   0   0   0   0   0   0   0   0
172   995   992   992   1810   992   172   1810   0   0   0   0   0   0   0   0   0   0
172   995   992   992   1810   992   172   1810   0   0   0   0   0   0   0   0   0   0
172   995   992   992   1810   992   172   1810   0   0   0   0   0   0   0   0   0   0
```

Note: Channels 8 to 16 will display 0 as they are disabled by default in the software.

Moving your transmitter sticks should cause corresponding channel data to change. Turning off your transmitter should result in lost frame and failsafe bits at the end of the coms data line to be set to '1'.

If no data is displayed, then check your wiring and that your receiver is bound to your transmitter.

If data is displayed but does not change, ensure you have setup your transmitter correctly to output 8 channels for pitch, roll, yaw, throttle, aux1, aux2, aux3 and aux4.

8.6 DEBUG_RADIO_COMMANDS

In defines.h re-comment out DEBUG_SBUS and uncomment DEBUG_RADIO_COMMANDS then reprogram your flight controller.

In Serial Monitor verify operation by moving the sticks or turning switches on/off, the outputs should be as described by Table 12.Typical data will look like:

```
armed: 0, mode: 1, aux1: 1, aux2: 2, thr: 2458, ail: 3686, pch: 3686, yaw: 3686
armed: 0, mode: 1, aux1: 1, aux2: 2, thr: 2458, ail: 3686, pch: 3686, yaw: 3686
armed: 0, mode: 1, aux1: 1, aux2: 2, thr: 2458, ail: 3686, pch: 3686, yaw: 3686
armed: 0, mode: 1, aux1: 1, aux2: 2, thr: 2458, ail: 3686, pch: 3686, yaw: 3686
armed: 0, mode: 1, aux1: 1, aux2: 2, thr: 2458, ail: 3686, pch: 3686, yaw: 3686
armed: 0, mode: 1, aux1: 1, aux2: 2, thr: 2458, ail: 3686, pch: 3686, yaw: 3686
armed: 0, mode: 1, aux1: 1, aux2: 2, thr: 2458, ail: 3686, pch: 3686, yaw: 3686
armed: 0, mode: 1, aux1: 1, aux2: 2, thr: 2458, ail: 3686, pch: 3686, yaw: 3686
```

Note: It is important to verify that your switches are operating correctly so correct flight modes can be set. Stick command outputs depends upon the flight mode set, flight mode/state testing is verified in 8.7.

Table 12 - Rx Commands

Command	Outputs
Armed	0 and 1
Mode	0, 1, and 2
Aux1 (Flaps)	0, 1, and 2
Aux2 (Heading Hold)	0 and 1
Throttle	Is an unsigned number that varies between MOTOR_MIN_TICKS and MOTOR_MAX_TICKS.
Aileron	When disarmed is an unsigned number that varies between SERVO_MIN_TICKS and SERVO_MAX_TICKS. When armed may become a signed number, values depend upon flight mode and defines of _ANGLE_DEGS_x100 or _RATE_DEGS_x100.
Pitch	When disarmed is an unsigned number that varies between SERVO_MIN_TICKS and SERVO_MAX_TICKS. When armed may become a signed number, values depend upon flight mode and defines of _ANGLE_DEGS_x100 or _RATE_DEGS_x100.
Yaw	When disarmed is an unsigned number that varies between SERVO_MIN_TICKS and SERVO_MAX_TICKS. When armed may become a signed number, values depend upon flight mode and defines of _ANGLE_DEGS_x100 or _RATE_DEGS_x100.

8.7 DEBUG_FLIGHT_STATE

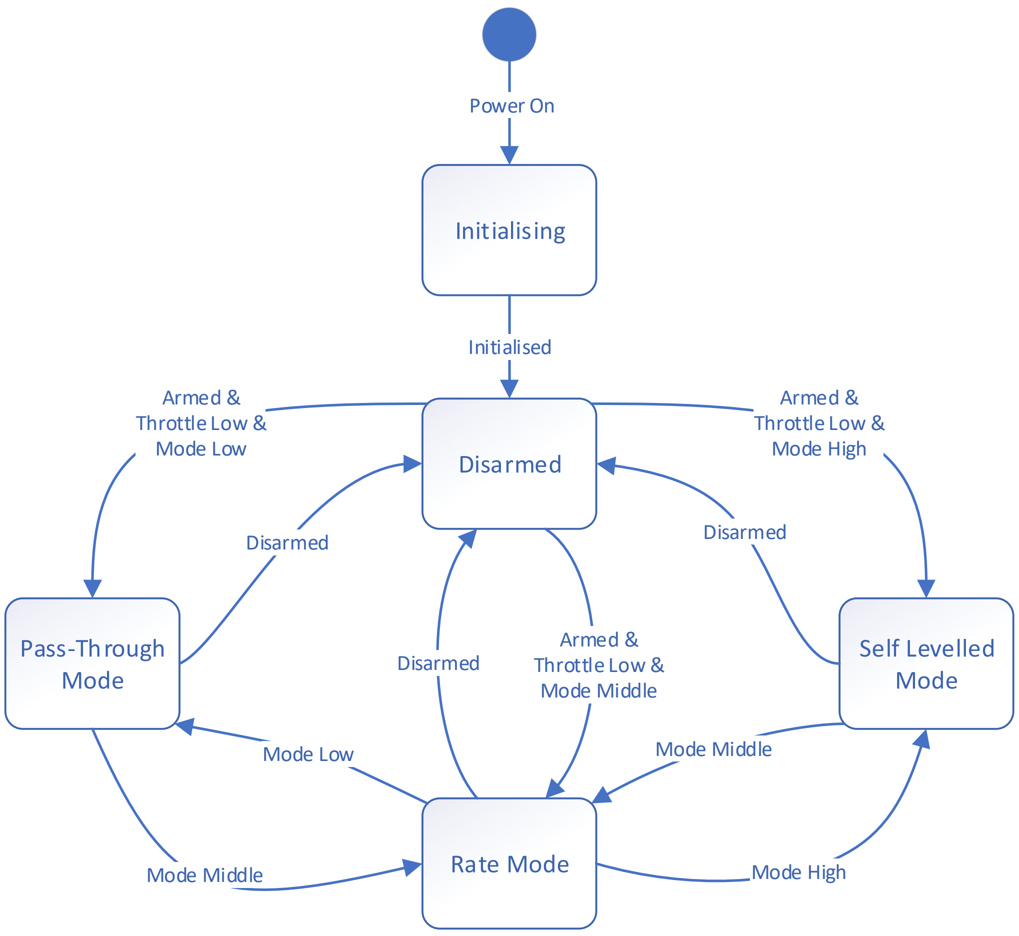
In defines.h re-comment out DEBUG_RADIO_COMMANDS and uncomment DEBUG_FLIGHT_STATE then reprogram your flight controller.

Verify that you can achieve all states via the Serial Monitor. If you cannot then ensure your switches are configured correctly and for 100% travel. The Serial Monitor will display data similar to:

```
State: 2
State: 0
State: 2
State: 1
State: 2
State: 3
```

Figure 16 details the flight states based upon the Armed and Mode switches. The term ‘Low’ is effectively 1000us upon your transmitter, and ‘High’ being 2000us.

Figure 16 - Flight States



Note: Initialising state will only exit once flight controller is still.
Note: For clarity Failsafe state has been omitted; loss of transmitter reception or SBus communications will result in failsafe entry. In failsafe the flight controller will implement ‘self-levelled’ operation but cut the throttle.

8.7.1 LED Status

The LED indicates the status of the flight controller via 5 different sequences, these are detailed in Table 13.

Table 13 - LED Sequences

Mode	Sequence
Failsafe	Constant quick flash
Disarmed	Slow 1 second on/off flash
Armed & passthrough mode	1 quick flash
Armed & rate mode	2 quick flashes
Armed & self-levelled mode	3 quick flashes
Initialising	1 long 3 second flash

Verify that your Armed switch and Mode switch operate and give all LED sequences shown in Table 13.

8.8 DEBUG_BATTERY_VOLTS

For this test you will need a multimeter to measure DC voltage and your flight battery. If motor is connected ensure your propeller(s) are removed.

In defines.h comment out DEBUG_FLIGHT_STATE and uncomment DEBUG_BATTERY_VOLTS then reprogram your flight controller.

Serial Monitor will now be displaying the flight battery voltage. Use your multimeter to verify that the battery voltage given by the flight controller is the same as it reads on your multimeter. It will likely be different due to resistor tolerances, if it differs by more than 0.1V then you can calibrate it by adjusting the value given by ADC_MULTIPLIER in Battery_Monitor.ino and reprogramming.

Once calibrated comment out DEBUG_BATTERY_VOLTS and reprogram your flight controller.

CAUTION: If you are powering from the flight controller from 1s 3.3V lipo battery, do not plug in the flight battery and USB in at the same time, as the USB will feed 5V to the lipo battery. For 1s lipo powered system calibrate the battery voltage via the USB 5V supply.

Caution: Ensure propeller(s) are removed when connecting USB with flight battery in circuit.

Caution: The 1s circuit will back feed 5V USB power to the ESC's, ensure ESC's can tolerate 5V and remove propeller(s).

8.9 DEBUG_PID

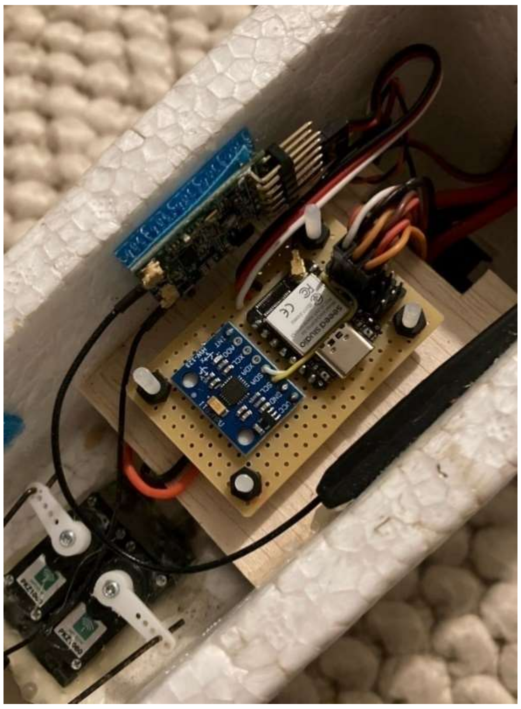
While this debug output is available it is not advised to use it as it will output all 3 instances of the PIDF controllers. Basically, it will be unreadable real-time to the human eye. Consider it for advanced users only.

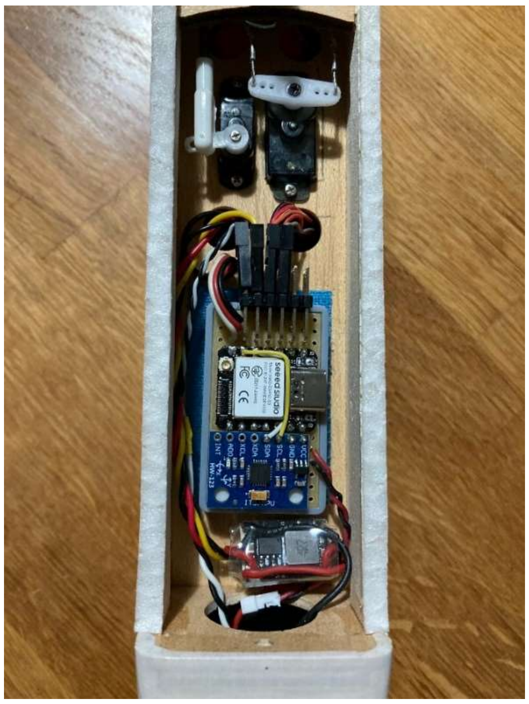
9. How To Install In Your Model

The flight control needs to be mounted with the MPU flat as shown in Figure 17. The gyro Z plane must be parallel to gravity, the X axis must be aligned with the pitch moment, and Y aligned with roll.

The IMU/MPU6050 can face forwards or backwards i.e. X axis +ve facing forward to the nose or towards the tail. However, depending upon the orientation set you may need to configure pitch/roll corrections as described in section 6 How To Configure Your Model.

Figure 17 - Flight Controller Installation





The flight controller should be mounted on foam tape or Velcro to help absorb any vibrations and ensure that no wires or objects can vibrate or knock onto the flight controller when in flight.

10. How To Tune

Firstly, if you have never tuned a flight controller or do not understand PIDF then seek assistance or training as this document assumes you have knowledge of flight controllers and understand PIDF control systems.

To tune your flight controller, you will have to set PIDF parameters for rate mode. You will also need to set heading hold (yaw i) gain when enabled in the software.

10.1 Rate Mode Tuning

Gyro rate mode is the most challenging to tune as you have three axis (pitch, roll and yaw) and four main parameters to tune per axis (PIDF).

Note: There is no gain attenuation based upon throttle position as you will find on other flight controllers. To date I have found that conservative P with higher I gains resolve this and gives excellent results. Adding D also can significantly improve the performance and reduces ‘bobble’ in flight by dampening P gain, it also reduces/removes speed oscillations and ultimately allows higher P gains to be used – I have been amazed by how well D term works on servos, see my YouTube video on this.

To those who believe D gain has no effect on servos you are partially correct. Servos will not respond to instantaneous pulses of D to speed the servo response, but the D term will prevent P gain overshoot, softens ends of manoeuvres and disturbances and reduces/removed speed oscillations. This is of course with correct tuning.

In defines.h find the following #define’s that control rate mode responses:

```
RATE_PITCH_P
RATE_PITCH_I
RATE_PITCH_D
RATE_PITCH_F
RATE_ROLL_P
RATE_ROLL_I
RATE_PITCH_D
RATE_ROLL_F
RATE_YAW_P
RATE_YAW_I          (Note: Yaw I gain is only used for heading hold feature)
RATE_PITCH_D
RATE_YAW_F
```

Table 14 - Gyro Rate Mode Gains

#define	Comment
RATE_xxx_F	Feed forward does most of the work during a manoeuvre and the P and I terms are there to ensure that the degrees per second demand is met or maintained. With rate PID gains set to zero, set F gain so you have at least 80% servo travel for full stick deflection when compared to pass-through mode. Note: As an example; When flying with PIF gains, perform a roll at maximum stick deflection and stop that roll quickly when level. If you notice bounce back (wings rolled back slightly) then F gain is a little too high. If you notice or feel that the wings rolled more than you commanded, then feedforward is too low.
RATE_xxx_P	Increasing the P gain results in faster corrections to disturbances. Too high a value will result in oscillation and possibly an uncontrollable situation – change to pass-through mode if you're ever in this situation. Airspeed will also affect this parameter and possible oscillations, so set this value conservatively for all axis. Note: Typical values that I have used vary between 45 and 100, but it will depend upon control surface area, deflection and servo response times.
RATE_xxx_I	I gain gives 'stiffness' or 'heading hold' to each axis. Initially set the I gain to the same value as your P. However, do not be concerned to exceed the P gain value. Note: Typical values that I have used vary between 100 and 300, but it will depend upon control surface area, deflection and servo response times. **Note**: I gain on yaw is only used when heading hold is enabled.
RATE_xxx_D	D gain is used to stop P gain overshoot and helps reduce oscillations due to the P term. Get your model flying with PIF initially, if you feel it is on the verge of oscillation, 'bobbles' in flight, or oscillates when at speed then add D gain to the affected control axis to soften the settling of the control surface. I have found D gains in the order of 500 to 1000 are effective. See my video on D gains for fixed wing.

Note: Always set gains conservatively to avoid uncontrolled behaviour on first flights.

10.2 Self-Levelled Mode Tuning

As of PlainFlight v1.1.0 There are no PIDF parameters to tune for levelled mode. It uses the gains set for rate mode. However, a few parameters relate to its operation.

The maximum angular speed at which it will self-correct to level is set by **MAX_xxx_RATE_DEGS_x100**, as used by rate mode. While the maximum bank angle allowed is set by **MAX_xxx_ANGLE_DEGS_x100**.

The algorithm implemented gives a very locked in flight feel as it uses the gyro rate controller and provides a snappy return to level if the pitch/roll stick is released quickly.

10.3 Heading Hold

When heading hold is enabled you will have to tune yaw 'I' gain for it. In defines.h find the following #define that controls heading hold mode response:

RATE_YAW_I

'I' gain gives the 'heading hold' feature to the yaw axis. Typical values that I have used have been between 100 and 300. However, it will depend upon control surface area, deflection, and servo response times.

10.4 Model Gain Examples

10.4.1 FLYING_WING with USE_DIFFERENTIAL_THROTTLE

~12" span (A4 paper size) twin motor (1102-10000kv) flying wing powered from 1s 650mAh lipo. Large elevon surfaces took a bit of tuning but found that D gain dampened roll P gain overshoot. 1.7g digital servos are running at SERVO_REFRESH_300HZ giving quickest possible update rate. Will hover hands off when using heading hold with differential thrust motors. With USE_LOW_VOLTS_CUT_OFF set to 3.4V limit and 3.3V cut off, flight times are around 8 minutes.

Note: Strongly advise use of 470uF 16V capacitors on power input of 1s BLHeli ESC's to prevent voltage spikes getting to the ESP32-S3 processor – I learnt the hard way.

Table 15 - Example Of Gains Used By A4

Gain	Value
RATE_PITCH_P	18
RATE_PITCH_I	100
RATE_PITCH_D	750
RATE_PITCH_F	18
RATE_ROLL_P	30
RATE_ROLL_I	100
RATE_PITCH_D	1000
RATE_ROLL_F	20
RATE_YAW_P	12
RATE_YAW_I	0
RATE_PITCH_D	20
RATE_YAW_F	3
MAX_ROLL_RATE_DEGS_x100	23000
MAX_PITCH_RATE_DEGS_x100	20000
MAX_YAW_RATE_DEGS_x100	10000

10.4.2 PLANE_FULL_HOUSE

This is an old FPV model I build back in 2015, balsa construction results in a stiff and responsive airframe. Has a non-symmetrical aerofoil that would cause it to 'balloon' when closing the throttle and when travelling at speed. Total of 10 degrees dihedral gives stability and allows rudder to steer the model. The high mounted motor can cause the nose to pitch down if throttle is applied quickly.

Adding PlainFlight to this model has resolved any 'ballooning', rudder coupling and power pitching issues. This model is now so locked in it feels like a mini pattern ship to fly (it is stupidly overpowered with the Tmotor F40 too). 4-point rolls are fantastic, and with heading hold (and a lot of throttle) it will knife edge hands off.

Initially tuned with PIF gains only, with these settings I found that aileron corrections would cause the wing to 'bobble' when disturbed (not oscillate, just observe fast aileron corrections), not a problem and what you would expect for a flight controller. The yaw corrections would do the same for the rudder when travelling at speed. This bobbling was resolved by adding D gain to both aileron and rudder which smooths any P gain overshoot around the set point (neutral). This is now an incredibly smooth, locked in model to fly and is an absolute pleasure for aerobatics, which it was not originally designed for.

Four off 4g digital MG BlueArrow servos are running at SERVO_REFRESH_150HZ giving a reasonable response rate, and USING_ONESHOT125 to operate a BLHeli ESC.

Table 16 - Example Of Gains Used By OldFPV

Gain	Value
RATE_PITCH_P	65
RATE_PITCH_I	175
RATE_PITCH_D	0
RATE_PITCH_F	18
RATE_ROLL_P	35
RATE_ROLL_I	150
RATE_PITCH_D	750
RATE_ROLL_F	15
RATE_YAW_P	40
RATE_YAW_I	200
RATE_PITCH_D	600
RATE_YAW_F	45
MAX_ROLL_RATE_DEGS_x100	23000
MAX_PITCH_RATE_DEGS_x100	12000
MAX_YAW_RATE_DEGS_x100	10000

10.4.3 PLANE_FULL_HOUSE

My original test bed during development of PlainFlight was this donated and tatty Horizon Hobby Super Cub.

Now fitted with an AXI 2808 and some hastily added ailerons, it has made this into a nice slow fun-fly model. Nothing is straight or square upon it and the wing flexes terribly when pulling manoeuvres. However, despite these problems its now a nice model that flies straight and true. Its party piece being hands off knife edge the length of our field when using heading hold, and very low slow knife edge (no heading hold). Flying this in self-levelled mode using only throttle and rudder controls to perform touch and goes has also been quite good fun. With heading hold this model will hold itself in prop-hang hands off, though lack of airflow over ailerons does mean that it drifts off after several seconds.

The PI gains are quite high on this model for 2 reasons, the original servos are quite slow to respond (even though SERVO_REFRESH_150HZ is set) and there is so much flex in the airframe that control corrections are absorbed before the model responds. To date I have not found D gain to give any beneficial effect upon this model and believe this is because of the flex in the model.

Gain	Value
RATE_PITCH_P	100
RATE_PITCH_I	300
RATE_PITCH_D	0
RATE_PITCH_F	18
RATE_ROLL_P	75
RATE_ROLL_I	150
RATE_PITCH_D	0
RATE_ROLL_F	25
RATE_YAW_P	80
RATE_YAW_I	300
RATE_PITCH_D	0
RATE_YAW_F	45
MAX_ROLL_RATE_DEGS_x100	18000
MAX_PITCH_RATE_DEGS_x100	18000
MAX_YAW_RATE_DEGS_x100	10000

11. Pre-Flight Checks

Strongly recommend that the following pre-flight check are carried out as a minimum. Not doing so may result in the loss of your model. Perform any additional checks you feel are necessary or wise to ensure safety and reliability of your model.

Note: It is assumed that you have experience in using and tuning flight controllers and understand PIDF control systems. If you are not confident seek advice or training.

Caution	Check	Comments
⚠	Ensure servos are centred and model control surfaces are mechanically trimmed to neutral positions.	Set servo horns to correct spline location. Use TRIM_SERVOx to centre servos before flying.
⚠	Ensure flight controller is firmly attached to model and in correct orientation.	Attach with double sided foam take or Velcro. Make sure no lose items can rub or strike the flight controller when in flight. Make sure all wired connections are secure.
⚠	Ensure pitch, roll, and yaw gyro corrections operate in the correct sense when in rate mode.	Use REVERSE_xxx_GYRO to reverse pitch, roll, yaw corrections for rate mode.
⚠	Ensure pitch and roll corrections operate in the correct sense when in self-levelled mode.	Use REVERSE_xxx_IMU to reverse pitch and/or roll corrections for self-levelled mode.
⚠	Ensure pitch, roll, and yaw operate in the correct sense to Tx commands.	If gyro and self-levelled corrections work correctly but Tx commands are the wrong sense, then reverse direction of affected channel(s) in your transmitter software.
⚠	Ensure power supply is sufficient for increased current demands of servos.	Move model around rapidly in rate mode for 10 seconds to ensure power supply does not brown out. With a helper; repeat rapid movements whilst pulsing throttle between low and full to ensure motor load does not affect power supply.
⚠	Set rate mode gains to sensible values.	Default gains given may not be appropriate for you model. Set gains so you can just see corrections occurring for first flight.
⚠	For first flight, fly model in pass through mode until you are confident.	For first tuning flights only change out of pass-through mode when you have sufficient height to try and recover the model in case gains are set too high tuned.

12. Disclaimer & License