# Table of Contents

[Table of Contents 1](#_Toc349929864)

[Definitions 2](#_Toc349929865)

[Documentation 3](#_Toc349929866)

[Error Codes 4](#_Toc349929867)

[ICARUS Communications Protocol 6](#_Toc349929868)

[Quickstart Code 9](#_Toc349929869)

[Arduino Sample Code 10](#_Toc349929870)

[Arducopter (v2.8.1a) Default Code 11](#_Toc349929871)

[ArduCopter 12](#_Toc349929872)

[APM\_Config.h 41](#_Toc349929873)

[APM\_Config\_mavlink\_hil.h 43](#_Toc349929874)

[Attitude 44](#_Toc349929875)

[Camera 52](#_Toc349929876)

[GCS.h 53](#_Toc349929877)

[GCS 56](#_Toc349929878)

[GCS\_Mavlink 58](#_Toc349929879)

[Log 83](#_Toc349929880)

[Parameters.h 95](#_Toc349929881)

[Parameters 101](#_Toc349929882)

[UserCode 104](#_Toc349929883)

[UserVariables.h 105](#_Toc349929884)

[commands 106](#_Toc349929885)

[commands\_logic 109](#_Toc349929886)

[commands\_process 120](#_Toc349929887)

[config.h 123](#_Toc349929888)

[config\_channels.h 136](#_Toc349929889)

[control\_modes 137](#_Toc349929890)

[defines.h 140](#_Toc349929891)

[events 146](#_Toc349929892)

[flip 148](#_Toc349929893)

[heli 150](#_Toc349929894)

[leds 155](#_Toc349929895)

[motors 157](#_Toc349929896)

[motors\_quad 160](#_Toc349929897)

[navigation 163](#_Toc349929898)

[planner 171](#_Toc349929899)

[radio 172](#_Toc349929900)

[sensors 175](#_Toc349929901)

[setup 178](#_Toc349929902)

[system 194](#_Toc349929903)

[ArduCopter 2.9.1 Default Code 203](#_Toc349929904)

[ArduCopter 204](#_Toc349929905)

[APM\_Config.h 234](#_Toc349929906)

[APM\_Config\_mavlink\_hil.h 235](#_Toc349929907)

[AP\_State 236](#_Toc349929908)

[Attitude 238](#_Toc349929909)

[GCS.h 254](#_Toc349929910)

[GCS 257](#_Toc349929911)

[GCS\_Mavlink 259](#_Toc349929912)

[Log 287](#_Toc349929913)

[Parameters.h 305](#_Toc349929914)

[Parameters 311](#_Toc349929915)

[UserCode 321](#_Toc349929916)

[UserVariables.h 322](#_Toc349929917)

[commands 323](#_Toc349929918)

[commands\_logic 326](#_Toc349929919)

[commands\_process 338](#_Toc349929920)

[config.h 341](#_Toc349929921)

[config\_channels.h 357](#_Toc349929922)

[control\_modes.pde 373](#_Toc349929923)

[defines.h 376](#_Toc349929924)

[events.pde 379](#_Toc349929925)

[failsafe.pde 381](#_Toc349929926)

[failsafe.pde 382](#_Toc349929927)

[flip 383](#_Toc349929928)

[inertia 384](#_Toc349929929)

[leds 385](#_Toc349929930)

[limits 390](#_Toc349929931)

[motors 395](#_Toc349929932)

[navigation 398](#_Toc349929933)

[perf\_info 408](#_Toc349929934)

[radio 409](#_Toc349929935)

[sensors 412](#_Toc349929936)

[setup 414](#_Toc349929937)

[system 429](#_Toc349929938)

[test 439](#_Toc349929939)

[toy 454](#_Toc349929940)

# Definitions

HIL: Hardware In the Loop

# Documentation

Text means that this highlighted code has been reviewed.

Text means that this highlighted code has been tested.

Text means that this highlighted code has errors/bugs.

Text means that this highlighted code has been edited.

## Error Codes

All error codes will be comprised ofa string of 5 fields of numeric digits separated by dashes, such as:

“1-3-7-5-1”

“5-14-0-1”

etc.

Unless otherwise specified, all fields that are not specified are not used.

### Field 1: System

Indicates what System the Error Code is being reported from.

|  |  |
| --- | --- |
| ***Digit 1*** | ***Description*** |
| 1 | Flyer |
| 2 | Not Used, Reserved for Future Use: Additional Vehicle |
| 3 | Not Used, Reserved for Future Use: Additional Vehicle |
| 4 | Not Used, Reserved for Future Use: Additional Vehicle |
| 5 | Ground Station |
| 6 | Not Used, Reserved for Future Use: Additional Ground Station |
| 7 | Remote Control |
| 8 | Not Used, Reserved for Future Use. |
| 9 | Not Used, Reserved for Future Use. |

### Field 2: Subsystem

Indicates which Subsystem the Error Code is being reported from.

|  |  |  |
| --- | --- | --- |
| ***Field 1*** | ***Field 2*** | ***Description*** |
| x | 0 | Entire System |
| 1 | 1 | Primary Controller |
| 1 | 2 | Flight Controller |
| 1 | 3 | Flight Controller GPS |
| 1 | 4 | Motion Controller |
|  |  |  |
|  |  |  |

### Field 3: Error Type

Indicated the general type of failure

|  |  |
| --- | --- |
| ***Field 3*** | ***Description*** |
| 0 | No Error |
| 1 | Electrical |
| 2 | Software |
| 3 | Communication |
| 4 | Sensors |
| 5 | Actuators |
| 6 | Data Storage |
| 9 | General Error |
|  |  |

### Field 4: Severity

Indicates the Error Severity Level

|  |  |
| --- | --- |
| ***Field 4*** | ***Description*** |
| 0 | No Error |
| 1 | Information |
| 2 | Minimal |
| 3 | Caution |
| 4 | Severe |
| 5 | Emergency |
|  |  |
|  |  |
|  |  |
|  |  |

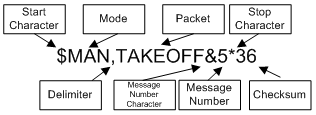
### Field 5: Error Message

|  |  |
| --- | --- |
| ***Field 5*** | ***Description*** |
| 0 | No Error, Normal Operation |
| 1 | Initializing |
| 2 | Initialization Error |
| 3 | General Error |
| 4 | Dropping Packets |
| 5 | Missing Heartbeats |
| 6 | Device not Present or Available |
|  |  |

# ICARUS Communications Protocol

**Packet Structures**

All Packets will be started with a “$”, their 3 character type code, any parameters, a "&", the message number, and will be terminated by a “\*”, and be delimited by a “,”.  Immediately following the "\*" character will be the complete packet size, starting with the "$" character and ending with the "\*" character.



\*DEPRECATED\* Sequence Numbers will be unsigned and 3 decimal digits long and will rollover after reaching 255.

Values encoded in the Packet Payload will be of varying datatypes and precisions with the following definitions:

PWM values range between 1000 and 2000.

GPS values (latitude and longitude) will be up to 3 digits long before a decimal point, 6 digits after with an optional sign bit.

INTEGERS will be unsigned and 4 decimal digits long and will rollover after reaching 9999.

DECIMAL will be 4 decimal digits (with an optional sign byte) and represent the value scaled by the number that is defined with that Data Type (to avoid any decimal point).

Orange packets are for example only, the document [ICARUS Protocol Specifications](https://spreadsheets.google.com/ccc?key=0As1I5roXKLwsdG9qZ1A4T29ONU1xeTZhR1BLYnU0TEE&hl=en#gid=0) should be used for further reference.

**Packet Types**

General Packets

*CAL*: Calibration

$CAL,<Message>|,<Value>\*

-Message:

"INFO":  Set Value on Vehicle.

“NEXT”: Go to next step in Calibration Procedure, where Value is the step.

“DONE”: When Calibration is complete.

*CAM*: Camera

$CAM,<Message>|,<Value,|<Value 2>, …<Value n>\*

-Message:

“DIST”: Depth Camera, where Value is the Value in inches from the Depth Sensor to the nearest obstacle in each Sector.

$CAM,DIST,000,111,222,333,444,555,666,777,888\* Distance to 9 Sectors. Sector 2 is 0 distance, meaning it is within the sensor’s minimum range.

*CON*: Control

$CON,<Message>|,<Value>\*

-Message:

“BOOT”: Selects Boot mode, where Value is the specific Boot Mode.

$CON,BOOT,1\*

“RESET”: Reboot Device.

$CON,RESET\*

"OFF":  Kills Device.

$CON,OFF\*

"TAKEOFFVTOL":  Command Device to enter TAKEOFF-VTOL Mode.

$CON,TAKEOFFVTOL\*

"HOVER":  Command Device to enter HOVER Mode.

$CON,HOVER\*

"LANDVTOL":  Command Device to enter LAND-VTOL Mode.

$CON,LANDVTOL\*

"CRUISE":  Command Device to enter CRUISE Mode.

$CON,CRUISE\*

"MANUAL":  Command Device to enter MANUAL Mode.

“ADVANCED”: Command Device to enter ADVANCED Mode.

$CON,ADVANCED\*

“MODE”: Command Device to change MODE based on the MAVLink Protocol:

$CON,MODE,256\* ‘Sets Mode to MAV\_MODE\_MANUAL\_DISARMED

*INF*: Informational Message

$INF, <Message>\*

-Message:  Any information to be passed between Interface and Vehicle.

*ERR*:  Error Message

$ERR,<Error #>\*

-Error Number is Error Code as described in Documentation.

$ERROR,12345&345\*12

*MOT:* Motor Control

$MOT,<Value 1>|<Value 2>...<Value n>\* Value 1 - 4 is a PWM value from 0-2000.  \*This packet controls each motor specifically.

$MOT,1000,1100,1900,2000\*

*NET*: Network Messages

$NET,<Message>|,<Value>\*

-Message:

"ACK", Message is received and acknowledged.

$NET,NCK\*12

"NCK", Message was not received correctly and is not acknowledged.

$NET,ACK&345\*12

"TEST": Performs Network Test, Vehicle Should respond back with: "$NET,ACK\*"

$NET,TEST&345\*12

“ID”: Network ID, where Value is the Channel between 0x00 and 0xFF.

$NET,ID,255&345\*12

“BAUD”: Network Baud Rate, where Value is the Baud Rate.

$NET,BAUD,1152&345\*12

“HRT”, Heartbeat, where Value is the Heartbeat ID

$NET,HRTBT,122\*

“TIME”,Time, where Value1 thru Value3 is the current GPS Time signal, in hours,minutes,seconds, respectively.

$NET,TIME,17,38,17\* *NOTE: This represents the time: 5:38 PM and 17 seconds.*

*SEN*: Sensor Data

$SEN,<Sensor Type>,<Value 1>,|<Value 2>, …<Value n>\*

-Sensor Types:

"ACC": Value 1 - x axis, Value 2 - y axis, Value 3 - z axis, in meters/second^2.

$SEN,ACC,0000,1111,2222\*12

"CMP":  Value 1 - heading, in degrees.

$SEN,CMP,000$345\*12

"ULT":  Value n - Ultrasonic Distance for Sensor n, in inches.

$SEN,ULT,000,111,222,333,444,555\*

"GYR":  Value 1 - yaw, Value 2 - roll, Value 3 - pitch, in degrees/second.

$SEN,GYR,0000,1111,2222\*12

"ENC":  Value n - Motor Speed for Encoder n, in revolutions per minute.

$SEN,ENC,0000,1111,2222,3333\*12

"ALT":  Value 1:  Altitude in meters

$SEN,ALT,123&345\*12

"INU":  Value 1 - x-axis displacement, Value 2 - y-axis displacement, Value 3 - z-axis displacement, in meters.  Value 4 - Pitch Angle, Value 5 - Roll Angle, Value 6 - Yaw Angle, in Degrees.

$SEN,INU,0000,1111,2222,3333,4444,5555\*12

"GPS":  Value 1 - Time, Value 2 - Latitude, Value 3 - Longitude, Value 4 - Altitude.

$SEN,GPS,000000,111111,222222,333333\*12

"PWMIN":  Values 1-4 are PWM Values ranging from 1000 to 2000

$PWMIN,1100,1300,1500,1700\*

*STA*: Status

$STA,<Message>|,<Value>\*

-Message:

"ALT":  Current Altitude in meters.

$STA,ALT,1000&345\*

“ARMED”: Armed State of device

$STA,ARMED,64\* Device is in Manual Control-Disarmed Mode.

“BEAR”: Current bearing to target, where Value is bearing in degrees.

$STA,BEAR,180&345\*

“DIST”: Current distance to target, where Value is distance in feet.

$STA,DIST,1000&345\*

“ERR”: Error Code, Errors defined in:

$STA,ERR,000001\*

"GPSFIX":  GPS Location is available.

$STA,GPSNOFIX&345\*

"GPSNOFIX":  GPS Location is not available.

$STA,GPSNOFIX&345\*

"INUFIX":  INU Data is available.

$STA,INUNOFIX&345\*

"INUNOFIX":  INU Data is not available.

$STA,INUNOFIX&345\*

“MODE”: Current MAVLink Flight Mode of device.

$STA,MODE,1\*

“POW5V”: Power level, where Value is the 5V battery voltage, in mV.

$STA,POW5V,1300&345\*

“POWMV”: Power level, where Value is the Main Supply Voltage, in mV

$STA,POWMV,1100&345\*

“QRY”: Query device for current status

$STA,QRY\*

“STATE”: Current MAVLink State of device.

$STA,STATE,1\* Device is in state MAV\_STATE\_BOOT

“VID”: Vehicle ID, where Value is the Vehicle ID.

$STA,VID,1000&345\*

New Packets

*SRV:* Servo Control

$SRV,<Value 1>,<Value 2>,<Value 3>,<Value 4>,<Value 5>,<Value 6>,<Value 7>,<Value 8>\*

Value 1-8 is a PWM value in mS for Servo Channels 1-8.

$SRV,1000,1100,1200,1500,1600,1700,1900,2000\*

Minimized Packets

The purpose of these packets is to minimize the overhead required.

*Motor Control*

Packet will be built like so:

Each motor, M1, M2, M3, M4 gets a value from 0x00 - 0xFE (0 - 254).

Packet will have a start byte of 0xFF.  So a sample packet would be:

0xFF01020304

Test Plan Packets

The purpose of these packets is for different Tests that must be performed to measure sensor and system characteristics.

*Phase 1 Test 2*

$RSSI,latitude,longitude,rssi\_value\*

*Phase 1 Test 5*

$RSSI,latitude,longitude,rssi\_value,seq\_number\*

GPS Packets

GPS Packets will follow the same conventions, following the NMEA 0183 Standard.

*RMC*: Recommended Minimum Specific GPS Data, gives Latitude/longitude, bearing, ground speed, etc.

*GSV*: Gives number of Satellite Views, etc.

# Quickstart Code

# Arduino Sample Code

### Blink LED

A\_LED\_PIN

B\_LED\_PIN

C\_LED\_PIN

digitalWrite(A\_LED\_PIN, LED\_OFF);

digitalWrite(B\_LED\_PIN, !digitalRead(B\_LED\_PIN));

### Send text to GCS

gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("apm mode - manua!"));

# ArduCopter 2.9.1 Default Code

### ROS Mode

ROS\_COMMAND\_MODE

### ROS Flight Modes:

YAW\_ROS

ROLL\_PITCH\_ROS

THROTTLE\_ROS

### Attitude Parameters

ros\_roll\_command

long ros\_pitch\_command

ros\_yaw\_command

ros\_thrust\_command

## ArduCopter

/// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

#define THISFIRMWARE "ArduCopter V2.9.1"

/\*

\* ArduCopter Version 2.9

\* Lead author: Jason Short

\* Based on code and ideas from the Arducopter team: Randy Mackay, Pat Hickey, Jose Julio, Jani Hirvinen, Andrew Tridgell, Justin Beech, Adam Rivera, Jean-Louis Naudin, Roberto Navoni

\* Thanks to: Chris Anderson, Mike Smith, Jordi Munoz, Doug Weibel, James Goppert, Benjamin Pelletier, Robert Lefebvre, Marco Robustini

\*

\* This firmware is free software; you can redistribute it and/or

\* modify it under the terms of the GNU Lesser General Public

\* License as published by the Free Software Foundation; either

\* version 2.1 of the License, or (at your option) any later version.

\*

\* Special Thanks for Contributors (in alphabetical order by first name):

\*

\* Adam M Rivera :Auto Compass Declination

\* Amilcar Lucas :Camera mount library

\* Andrew Tridgell :General development, Mavlink Support

\* Angel Fernandez :Alpha testing

\* Doug Weibel :Libraries

\* Christof Schmid :Alpha testing

\* Dani Saez :V Octo Support

\* Gregory Fletcher :Camera mount orientation math

\* Guntars :Arming safety suggestion

\* HappyKillmore :Mavlink GCS

\* Hein Hollander :Octo Support

\* Igor van Airde :Control Law optimization

\* Leonard Hall :Flight Dynamics, INAV throttle

\* Jonathan Challinger :Inertial Navigation

\* Jean-Louis Naudin :Auto Landing

\* Max Levine :Tri Support, Graphics

\* Jack Dunkle :Alpha testing

\* James Goppert :Mavlink Support

\* Jani Hiriven :Testing feedback

\* John Arne Birkeland :PPM Encoder

\* Jose Julio :Stabilization Control laws

\* Randy Mackay :General development and release

\* Marco Robustini :Lead tester

\* Michael Oborne :Mission Planner GCS

\* Mike Smith :Libraries, Coding support

\* Oliver :Piezo support

\* Olivier Adler :PPM Encoder

\* Robert Lefebvre :Heli Support & LEDs

\* Sandro Benigno :Camera support

\*

\* And much more so PLEASE PM me on DIYDRONES to add your contribution to the List

\*

\* Requires modified "mrelax" version of Arduino, which can be found here:

\* http://code.google.com/p/ardupilot-mega/downloads/list

\*

\*/

////////////////////////////////////////////////////////////////////////////////

// Header includes

////////////////////////////////////////////////////////////////////////////////

// AVR runtime

#include <avr/io.h>

#include <avr/eeprom.h>

#include <avr/pgmspace.h>

#include <avr/wdt.h>

#include <math.h>

// Libraries

#include <FastSerial.h>

#include <AP\_Common.h>

#include <AP\_Menu.h>

#include <Arduino\_Mega\_ISR\_Registry.h>

#include <APM\_RC.h> // ArduPilot Mega RC Library

#include <AP\_GPS.h> // ArduPilot GPS library

#include <I2C.h> // Arduino I2C lib

#include <SPI.h> // Arduino SPI lib

#include <SPI3.h> // SPI3 library

#include <AP\_Semaphore.h> // for removing conflict between optical flow and dataflash on SPI3 bus

#include <DataFlash.h> // ArduPilot Mega Flash Memory Library

#include <AP\_ADC.h> // ArduPilot Mega Analog to Digital Converter Library

#include <AP\_AnalogSource.h>

#include <AP\_Baro.h>

#include <AP\_Compass.h> // ArduPilot Mega Magnetometer Library

#include <AP\_Math.h> // ArduPilot Mega Vector/Matrix math Library

#include <AP\_Curve.h> // Curve used to linearlise throttle pwm to thrust

#include <AP\_InertialSensor.h> // ArduPilot Mega Inertial Sensor (accel & gyro) Library

#include <AP\_PeriodicProcess.h> // Parent header of Timer

// (only included for makefile libpath to work)

#include <AP\_TimerProcess.h> // TimerProcess is the scheduler for MPU6000 reads.

#include <AP\_AHRS.h>

#include <APM\_PI.h> // PI library

#include <AC\_PID.h> // PID library

#include <RC\_Channel.h> // RC Channel Library

#include <AP\_Motors.h> // AP Motors library

#include <AP\_MotorsQuad.h> // AP Motors library for Quad

#include <AP\_MotorsTri.h> // AP Motors library for Tri

#include <AP\_MotorsHexa.h> // AP Motors library for Hexa

#include <AP\_MotorsY6.h> // AP Motors library for Y6

#include <AP\_MotorsOcta.h> // AP Motors library for Octa

#include <AP\_MotorsOctaQuad.h> // AP Motors library for OctaQuad

#include <AP\_MotorsHeli.h> // AP Motors library for Heli

#include <AP\_MotorsMatrix.h> // AP Motors library for Heli

#include <AP\_RangeFinder.h> // Range finder library

#include <AP\_OpticalFlow.h> // Optical Flow library

#include <Filter.h> // Filter library

#include <AP\_Buffer.h> // APM FIFO Buffer

#include <ModeFilter.h> // Mode Filter from Filter library

#include <AverageFilter.h> // Mode Filter from Filter library

#include <AP\_LeadFilter.h> // GPS Lead filter

#include <LowPassFilter.h> // Low Pass Filter library

#include <AP\_Relay.h> // APM relay

#include <AP\_Camera.h> // Photo or video camera

#include <AP\_Mount.h> // Camera/Antenna mount

#include <AP\_Airspeed.h> // needed for AHRS build

#include <AP\_InertialNav.h> // ArduPilot Mega inertial navigation library

#include <DigitalWriteFast.h> // faster digital write for LEDs

#include <memcheck.h>

// Configuration

#include "defines.h"

#include "config.h"

#include "config\_channels.h"

#include <GCS\_MAVLink.h> // MAVLink GCS definitions

// Local modules

#include "Parameters.h"

#include "GCS.h"

#include <AP\_Declination.h> // ArduPilot Mega Declination Helper Library

// Limits library - Puts limits on the vehicle, and takes recovery actions

#include <AP\_Limits.h>

#include <AP\_Limit\_GPSLock.h> // a limits library module

#include <AP\_Limit\_Geofence.h> // a limits library module

#include <AP\_Limit\_Altitude.h> // a limits library module

////////////////////////////////////////////////////////////////////////////////

// Serial ports

////////////////////////////////////////////////////////////////////////////////

//

// Note that FastSerial port buffers are allocated at ::begin time,

// so there is not much of a penalty to defining ports that we don't

// use.

//

FastSerialPort0(Serial); // FTDI/console

FastSerialPort1(Serial1); // GPS port

FastSerialPort3(Serial3); // Telemetry port

// port to use for command line interface

static FastSerial \*cliSerial = &Serial;

// this sets up the parameter table, and sets the default values. This

// must be the first AP\_Param variable declared to ensure its

// constructor runs before the constructors of the other AP\_Param

// variables

AP\_Param param\_loader(var\_info, WP\_START\_BYTE);

Arduino\_Mega\_ISR\_Registry isr\_registry;

////////////////////////////////////////////////////////////////////////////////

// Parameters

////////////////////////////////////////////////////////////////////////////////

//

// Global parameters are all contained within the 'g' class.

//

static Parameters g;

////////////////////////////////////////////////////////////////////////////////

// prototypes

static void update\_events(void);

////////////////////////////////////////////////////////////////////////////////

// RC Hardware

////////////////////////////////////////////////////////////////////////////////

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

APM\_RC\_APM2 APM\_RC;

#else

APM\_RC\_APM1 APM\_RC;

#endif

////////////////////////////////////////////////////////////////////////////////

// Dataflash

////////////////////////////////////////////////////////////////////////////////

AP\_Semaphore spi\_semaphore;

AP\_Semaphore spi3\_semaphore;

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

DataFlash\_APM2 DataFlash(&spi3\_semaphore);

#else

DataFlash\_APM1 DataFlash(&spi\_semaphore);

#endif

////////////////////////////////////////////////////////////////////////////////

// the rate we run the main loop at

////////////////////////////////////////////////////////////////////////////////

static const AP\_InertialSensor::Sample\_rate ins\_sample\_rate = AP\_InertialSensor::RATE\_200HZ;

////////////////////////////////////////////////////////////////////////////////

// Sensors

////////////////////////////////////////////////////////////////////////////////

//

// There are three basic options related to flight sensor selection.

//

// - Normal flight mode. Real sensors are used.

// - HIL Attitude mode. Most sensors are disabled, as the HIL

// protocol supplies attitude information directly.

// - HIL Sensors mode. Synthetic sensors are configured that

// supply data from the simulation.

//

// All GPS access should be through this pointer.

static GPS \*g\_gps;

// flight modes convenience array

static AP\_Int8 \*flight\_modes = &g.flight\_mode1;

#if HIL\_MODE == HIL\_MODE\_DISABLED

// real sensors

#if CONFIG\_ADC == ENABLED

AP\_ADC\_ADS7844 adc;

#endif

#ifdef DESKTOP\_BUILD

AP\_Baro\_BMP085\_HIL barometer;

AP\_Compass\_HIL compass;

#include <SITL.h>

SITL sitl;

#else

#if CONFIG\_BARO == AP\_BARO\_BMP085

# if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

AP\_Baro\_BMP085 barometer(true);

# else

AP\_Baro\_BMP085 barometer(false);

# endif

#elif CONFIG\_BARO == AP\_BARO\_MS5611

AP\_Baro\_MS5611 barometer;

#endif

AP\_Compass\_HMC5843 compass;

#endif

#if OPTFLOW == ENABLED

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

AP\_OpticalFlow\_ADNS3080 optflow(OPTFLOW\_CS\_PIN);

#else

AP\_OpticalFlow\_ADNS3080 optflow(OPTFLOW\_CS\_PIN);

#endif

#else

AP\_OpticalFlow optflow;

#endif

// real GPS selection

#if GPS\_PROTOCOL == GPS\_PROTOCOL\_AUTO

AP\_GPS\_Auto g\_gps\_driver(&Serial1, &g\_gps);

#elif GPS\_PROTOCOL == GPS\_PROTOCOL\_NMEA

AP\_GPS\_NMEA g\_gps\_driver(&Serial1);

#elif GPS\_PROTOCOL == GPS\_PROTOCOL\_SIRF

AP\_GPS\_SIRF g\_gps\_driver(&Serial1);

#elif GPS\_PROTOCOL == GPS\_PROTOCOL\_UBLOX

AP\_GPS\_UBLOX g\_gps\_driver(&Serial1);

#elif GPS\_PROTOCOL == GPS\_PROTOCOL\_MTK

AP\_GPS\_MTK g\_gps\_driver(&Serial1);

#elif GPS\_PROTOCOL == GPS\_PROTOCOL\_MTK19

AP\_GPS\_MTK19 g\_gps\_driver(&Serial1);

#elif GPS\_PROTOCOL == GPS\_PROTOCOL\_NONE

AP\_GPS\_None g\_gps\_driver(NULL);

#else

#error Unrecognised GPS\_PROTOCOL setting.

#endif // GPS PROTOCOL

#if CONFIG\_IMU\_TYPE == CONFIG\_IMU\_MPU6000

AP\_InertialSensor\_MPU6000 ins;

#else

AP\_InertialSensor\_Oilpan ins(&adc);

#endif

#if DMP\_ENABLED == ENABLED && CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

AP\_AHRS\_MPU6000 ahrs(&ins, g\_gps); // only works with APM2

#else

AP\_AHRS\_DCM ahrs(&ins, g\_gps);

#endif

// ahrs2 object is the secondary ahrs to allow running DMP in parallel with DCM

#if SECONDARY\_DMP\_ENABLED == ENABLED && CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

AP\_AHRS\_MPU6000 ahrs2(&ins, g\_gps); // only works with APM2

#endif

#elif HIL\_MODE == HIL\_MODE\_SENSORS

// sensor emulators

AP\_ADC\_HIL adc;

AP\_Baro\_BMP085\_HIL barometer;

AP\_Compass\_HIL compass;

AP\_GPS\_HIL g\_gps\_driver(NULL);

AP\_InertialSensor\_Stub ins;

AP\_AHRS\_DCM ahrs(&ins, g\_gps);

static int32\_t gps\_base\_alt;

#elif HIL\_MODE == HIL\_MODE\_ATTITUDE

AP\_ADC\_HIL adc;

AP\_InertialSensor\_Stub ins;

AP\_AHRS\_HIL ahrs(&ins, g\_gps);

AP\_GPS\_HIL g\_gps\_driver(NULL);

AP\_Compass\_HIL compass; // never used

AP\_Baro\_BMP085\_HIL barometer;

#if OPTFLOW == ENABLED

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

AP\_OpticalFlow\_ADNS3080 optflow(OPTFLOW\_CS\_PIN);

#else

AP\_OpticalFlow\_ADNS3080 optflow(OPTFLOW\_CS\_PIN);

#endif // CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

#endif // OPTFLOW == ENABLED

#ifdef DESKTOP\_BUILD

#include <SITL.h>

SITL sitl;

#endif // DESKTOP\_BUILD

static int32\_t gps\_base\_alt;

#else

#error Unrecognised HIL\_MODE setting.

#endif // HIL MODE

// we always have a timer scheduler

AP\_TimerProcess timer\_scheduler;

////////////////////////////////////////////////////////////////////////////////

// GCS selection

////////////////////////////////////////////////////////////////////////////////

GCS\_MAVLINK gcs0;

GCS\_MAVLINK gcs3;

////////////////////////////////////////////////////////////////////////////////

// SONAR selection

////////////////////////////////////////////////////////////////////////////////

//

ModeFilterInt16\_Size3 sonar\_mode\_filter(1);

#if CONFIG\_SONAR == ENABLED

#if CONFIG\_SONAR\_SOURCE == SONAR\_SOURCE\_ADC

AP\_AnalogSource\_ADC sonar\_analog\_source( &adc, CONFIG\_SONAR\_SOURCE\_ADC\_CHANNEL, 0.25);

#elif CONFIG\_SONAR\_SOURCE == SONAR\_SOURCE\_ANALOG\_PIN

AP\_AnalogSource\_Arduino sonar\_analog\_source(CONFIG\_SONAR\_SOURCE\_ANALOG\_PIN);

#endif

AP\_RangeFinder\_MaxsonarXL sonar(&sonar\_analog\_source, &sonar\_mode\_filter);

#endif

// agmatthews USERHOOKS

////////////////////////////////////////////////////////////////////////////////

// User variables

////////////////////////////////////////////////////////////////////////////////

#ifdef USERHOOK\_VARIABLES

#include USERHOOK\_VARIABLES

#endif

////////////////////////////////////////////////////////////////////////////////

// Global variables

////////////////////////////////////////////////////////////////////////////////

/\* Radio values

\* Channel assignments

\* 1 Ailerons (rudder if no ailerons)

\* 2 Elevator

\* 3 Throttle

\* 4 Rudder (if we have ailerons)

\* 5 Mode - 3 position switch

\* 6 User assignable

\* 7 trainer switch - sets throttle nominal (toggle switch), sets accels to Level (hold > 1 second)

\* 8 TBD

\* Each Aux channel can be configured to have any of the available auxiliary functions assigned to it.

\* See libraries/RC\_Channel/RC\_Channel\_aux.h for more information

\*/

//Documentation of GLobals:

static union {

struct {

uint8\_t home\_is\_set : 1; // 1

uint8\_t simple\_mode : 1; // 2 // This is the state of simple mode

uint8\_t manual\_attitude : 1; // 3

uint8\_t manual\_throttle : 1; // 4

uint8\_t low\_battery : 1; // 5 // Used to track if the battery is low - LED output flashes when the batt is low

uint8\_t loiter\_override : 1; // 6 // Are we navigating while holding a positon? This is set to false once the speed drops below 1m/s

uint8\_t armed : 1; // 7

uint8\_t auto\_armed : 1; // 8

uint8\_t failsafe : 1; // 9 // A status flag for the failsafe state

uint8\_t do\_flip : 1; // 10 // Used to enable flip code

uint8\_t takeoff\_complete : 1; // 11

uint8\_t land\_complete : 1; // 12

uint8\_t compass\_status : 1; // 13

uint8\_t gps\_status : 1; // 14

uint8\_t fast\_corner : 1; // 15 // should we take the waypoint quickly or slow down?

};

uint16\_t value;

} ap;

static struct AP\_System{

uint8\_t GPS\_light : 1; // 1 // Solid indicates we have full 3D lock and can navigate, flash = read

uint8\_t motor\_light : 1; // 2 // Solid indicates Armed state

uint8\_t new\_radio\_frame : 1; // 3 // Set true if we have new PWM data to act on from the Radio

uint8\_t nav\_ok : 1; // 4 // deprecated

uint8\_t CH7\_flag : 1; // 5 // manages state of the ch7 toggle switch

uint8\_t usb\_connected : 1; // 6 // true if APM is powered from USB connection

uint8\_t run\_50hz\_loop : 1; // 7 // toggles the 100hz loop for 50hz

uint8\_t alt\_sensor\_flag : 1; // 8 // used to track when to read sensors vs estimate alt

uint8\_t yaw\_stopped : 1; // 9 // Used to manage the Yaw hold capabilities

} ap\_system;

////////////////////////////////////////////////////////////////////////////////

// velocity in lon and lat directions calculated from GPS position and accelerometer data

// updated after GPS read - 5-10hz

static int16\_t lon\_speed; // expressed in cm/s. positive numbers mean moving east

static int16\_t lat\_speed; // expressed in cm/s. positive numbers when moving north

// The difference between the desired rate of travel and the actual rate of travel

// updated after GPS read - 5-10hz

static int16\_t x\_rate\_error;

static int16\_t y\_rate\_error;

////////////////////////////////////////////////////////////////////////////////

// Radio

////////////////////////////////////////////////////////////////////////////////

// This is the state of the flight control system

// There are multiple states defined such as STABILIZE, ACRO,

static int8\_t control\_mode = STABILIZE;

// Used to maintain the state of the previous control switch position

// This is set to -1 when we need to re-read the switch

static byte oldSwitchPosition;

// receiver RSSI

static uint8\_t receiver\_rssi;

////////////////////////////////////////////////////////////////////////////////

// Motor Output

////////////////////////////////////////////////////////////////////////////////

#if FRAME\_CONFIG == QUAD\_FRAME

#define MOTOR\_CLASS AP\_MotorsQuad

#endif

#if FRAME\_CONFIG == TRI\_FRAME

#define MOTOR\_CLASS AP\_MotorsTri

#endif

#if FRAME\_CONFIG == HEXA\_FRAME

#define MOTOR\_CLASS AP\_MotorsHexa

#endif

#if FRAME\_CONFIG == Y6\_FRAME

#define MOTOR\_CLASS AP\_MotorsY6

#endif

#if FRAME\_CONFIG == OCTA\_FRAME

#define MOTOR\_CLASS AP\_MotorsOcta

#endif

#if FRAME\_CONFIG == OCTA\_QUAD\_FRAME

#define MOTOR\_CLASS AP\_MotorsOctaQuad

#endif

#if FRAME\_CONFIG == HELI\_FRAME

#define MOTOR\_CLASS AP\_MotorsHeli

#endif

#if FRAME\_CONFIG == HELI\_FRAME // helicopter constructor requires more arguments

MOTOR\_CLASS motors(CONFIG\_APM\_HARDWARE, &APM\_RC, &g.rc\_1, &g.rc\_2, &g.rc\_3, &g.rc\_4, &g.rc\_8, &g.heli\_servo\_1, &g.heli\_servo\_2, &g.heli\_servo\_3, &g.heli\_servo\_4);

#elif FRAME\_CONFIG == TRI\_FRAME // tri constructor requires additional rc\_7 argument to allow tail servo reversing

MOTOR\_CLASS motors(CONFIG\_APM\_HARDWARE, &APM\_RC, &g.rc\_1, &g.rc\_2, &g.rc\_3, &g.rc\_4, &g.rc\_7);

#else

MOTOR\_CLASS motors(CONFIG\_APM\_HARDWARE, &APM\_RC, &g.rc\_1, &g.rc\_2, &g.rc\_3, &g.rc\_4);

#endif

////////////////////////////////////////////////////////////////////////////////

// PIDs

////////////////////////////////////////////////////////////////////////////////

// This is a convienience accessor for the IMU roll rates. It's currently the raw IMU rates

// and not the adjusted omega rates, but the name is stuck

static Vector3f omega;

// This is used to hold radio tuning values for in-flight CH6 tuning

float tuning\_value;

// used to limit the rate that the pid controller output is logged so that it doesn't negatively affect performance

static uint8\_t pid\_log\_counter;

////////////////////////////////////////////////////////////////////////////////

// LED output

////////////////////////////////////////////////////////////////////////////////

// This is current status for the LED lights state machine

// setting this value changes the output of the LEDs

static byte led\_mode = NORMAL\_LEDS;

// Blinking indicates GPS status

static byte copter\_leds\_GPS\_blink;

// Blinking indicates battery status

static byte copter\_leds\_motor\_blink;

// Navigation confirmation blinks

static int8\_t copter\_leds\_nav\_blink;

////////////////////////////////////////////////////////////////////////////////

// GPS variables

////////////////////////////////////////////////////////////////////////////////

// This is used to scale GPS values for EEPROM storage

// 10^7 times Decimal GPS means 1 == 1cm

// This approximation makes calculations integer and it's easy to read

static const float t7 = 10000000.0;

// We use atan2 and other trig techniques to calaculate angles

// We need to scale the longitude up to make these calcs work

// to account for decreasing distance between lines of longitude away from the equator

static float scaleLongUp = 1;

// Sometimes we need to remove the scaling for distance calcs

static float scaleLongDown = 1;

////////////////////////////////////////////////////////////////////////////////

// Mavlink specific

////////////////////////////////////////////////////////////////////////////////

// Used by Mavlink for unknow reasons

static const float radius\_of\_earth = 6378100; // meters

// Used by Mavlink for unknow reasons

static const float gravity = 9.80665; // meters/ sec^2

// Unions for getting byte values

union float\_int {

int32\_t int\_value;

float float\_value;

} float\_int;

////////////////////////////////////////////////////////////////////////////////

// Location & Navigation

////////////////////////////////////////////////////////////////////////////////

// This is the angle from the copter to the "next\_WP" location in degrees \* 100

static int32\_t wp\_bearing;

// Status of the Waypoint tracking mode. Options include:

// NO\_NAV\_MODE, WP\_MODE, LOITER\_MODE, CIRCLE\_MODE

static byte wp\_control;

// Register containing the index of the current navigation command in the mission script

static int16\_t command\_nav\_index;

// Register containing the index of the previous navigation command in the mission script

// Used to manage the execution of conditional commands

static uint8\_t prev\_nav\_index;

// Register containing the index of the current conditional command in the mission script

static uint8\_t command\_cond\_index;

// Used to track the required WP navigation information

// options include

// NAV\_ALTITUDE - have we reached the desired altitude?

// NAV\_LOCATION - have we reached the desired location?

// NAV\_DELAY - have we waited at the waypoint the desired time?

static uint8\_t wp\_verify\_byte; // used for tracking state of navigating waypoints

// used to limit the speed ramp up of WP navigation

// Acceleration is limited to 1m/s/s

static int16\_t max\_speed\_old;

// Used to track how many cm we are from the "next\_WP" location

static int32\_t long\_error, lat\_error;

static int16\_t control\_roll;

static int16\_t control\_pitch;

static uint8\_t rtl\_state;

////////////////////////////////////////////////////////////////////////////////

// Orientation

////////////////////////////////////////////////////////////////////////////////

// Convienience accessors for commonly used trig functions. These values are generated

// by the DCM through a few simple equations. They are used throughout the code where cos and sin

// would normally be used.

// The cos values are defaulted to 1 to get a decent initial value for a level state

static float cos\_roll\_x = 1;

static float cos\_pitch\_x = 1;

static float cos\_yaw\_x = 1;

static float sin\_yaw\_y;

static float sin\_roll;

static float sin\_pitch;

////////////////////////////////////////////////////////////////////////////////

// SIMPLE Mode

////////////////////////////////////////////////////////////////////////////////

// Used to track the orientation of the copter for Simple mode. This value is reset at each arming

// or in SuperSimple mode when the copter leaves a 20m radius from home.

static int32\_t initial\_simple\_bearing;

////////////////////////////////////////////////////////////////////////////////

// Rate contoller targets

////////////////////////////////////////////////////////////////////////////////

static uint8\_t rate\_targets\_frame = EARTH\_FRAME; // indicates whether rate targets provided in earth or body frame

static int32\_t roll\_rate\_target\_ef = 0;

static int32\_t pitch\_rate\_target\_ef = 0;

static int32\_t yaw\_rate\_target\_ef = 0;

static int32\_t roll\_rate\_target\_bf = 0; // body frame roll rate target

static int32\_t pitch\_rate\_target\_bf = 0; // body frame pitch rate target

static int32\_t yaw\_rate\_target\_bf = 0; // body frame yaw rate target

////////////////////////////////////////////////////////////////////////////////

// Throttle variables

////////////////////////////////////////////////////////////////////////////////

static int16\_t throttle\_accel\_target\_ef; // earth frame throttle acceleration target

static bool throttle\_accel\_controller\_active; // true when accel based throttle controller is active, false when higher level throttle controllers are providing throttle output directly

static float throttle\_avg; // g.throttle\_cruise as a float

static int16\_t desired\_climb\_rate; // pilot desired climb rate - for logging purposes only

////////////////////////////////////////////////////////////////////////////////

// ACRO Mode

////////////////////////////////////////////////////////////////////////////////

// Used to control Axis lock

int32\_t roll\_axis;

int32\_t pitch\_axis;

// Filters

AP\_LeadFilter xLeadFilter; // Long GPS lag filter

AP\_LeadFilter yLeadFilter; // Lat GPS lag filter

#if FRAME\_CONFIG == HELI\_FRAME

LowPassFilterFloat rate\_roll\_filter; // Rate Roll filter

LowPassFilterFloat rate\_pitch\_filter; // Rate Pitch filter

// LowPassFilterFloat rate\_yaw\_filter; // Rate Yaw filter

#endif // HELI\_FRAME

// Barometer filter

AverageFilterInt32\_Size5 baro\_filter;

////////////////////////////////////////////////////////////////////////////////

// Circle Mode / Loiter control

////////////////////////////////////////////////////////////////////////////////

// used to determin the desired location in Circle mode

// increments at circle\_rate / second

static float circle\_angle;

// used to control the speed of Circle mode

// units are in radians, default is 5° per second

static const float circle\_rate = 0.0872664625;

// used to track the delat in Circle Mode

static int32\_t old\_wp\_bearing;

// deg : how many times to circle \* 360 for Loiter/Circle Mission command

static int16\_t loiter\_total;

// deg : how far we have turned around a waypoint

static int16\_t loiter\_sum;

// How long we should stay in Loiter Mode for mission scripting

static uint16\_t loiter\_time\_max;

// How long have we been loitering - The start time in millis

static uint32\_t loiter\_time;

// The synthetic location created to make the copter do circles around a WP

static struct Location circle\_WP;

////////////////////////////////////////////////////////////////////////////////

// CH7 control

////////////////////////////////////////////////////////////////////////////////

// This register tracks the current Mission Command index when writing

// a mission using CH7 in flight

static int8\_t CH7\_wp\_index;

////////////////////////////////////////////////////////////////////////////////

// Battery Sensors

////////////////////////////////////////////////////////////////////////////////

// Battery Voltage of battery, initialized above threshold for filter

static float battery\_voltage1 = LOW\_VOLTAGE \* 1.05;

// refers to the instant amp draw – based on an Attopilot Current sensor

static float current\_amps1;

// refers to the total amps drawn – based on an Attopilot Current sensor

static float current\_total1;

////////////////////////////////////////////////////////////////////////////////

// Altitude

////////////////////////////////////////////////////////////////////////////////

// The (throttle) controller desired altitude in cm

static float controller\_desired\_alt;

// The cm we are off in altitude from next\_WP.alt – Positive value means we are below the WP

static int32\_t altitude\_error;

// The cm/s we are moving up or down based on sensor data - Positive = UP

static int16\_t climb\_rate\_actual;

// Used to dither our climb\_rate over 50hz

static int16\_t climb\_rate\_error;

// The cm/s we are moving up or down based on filtered data - Positive = UP

static int16\_t climb\_rate;

// The altitude as reported by Sonar in cm – Values are 20 to 700 generally.

static int16\_t sonar\_alt;

static uint8\_t sonar\_alt\_health; // true if we can trust the altitude from the sonar

// The climb\_rate as reported by sonar in cm/s

static int16\_t sonar\_rate;

// The altitude as reported by Baro in cm – Values can be quite high

static int32\_t baro\_alt;

// The climb\_rate as reported by Baro in cm/s

static int16\_t baro\_rate;

static int16\_t saved\_toy\_throttle;

////////////////////////////////////////////////////////////////////////////////

// flight modes

////////////////////////////////////////////////////////////////////////////////

// Flight modes are combinations of Roll/Pitch, Yaw and Throttle control modes

// Each Flight mode is a unique combination of these modes

//

// The current desired control scheme for Yaw

static uint8\_t yaw\_mode;

// The current desired control scheme for roll and pitch / navigation

static uint8\_t roll\_pitch\_mode;

// The current desired control scheme for altitude hold

static uint8\_t throttle\_mode;

////////////////////////////////////////////////////////////////////////////////

// flight specific

////////////////////////////////////////////////////////////////////////////////

// An additional throttle added to keep the copter at the same altitude when banking

static int16\_t angle\_boost;

// counter to verify landings

static uint16\_t land\_detector;

////////////////////////////////////////////////////////////////////////////////

// Navigation general

////////////////////////////////////////////////////////////////////////////////

// The location of home in relation to the copter, updated every GPS read

static int32\_t home\_bearing;

// distance between plane and home in cm

static int32\_t home\_distance;

// distance between plane and next\_WP in cm

// is not static because AP\_Camera uses it

int32\_t wp\_distance;

////////////////////////////////////////////////////////////////////////////////

// 3D Location vectors

////////////////////////////////////////////////////////////////////////////////

// home location is stored when we have a good GPS lock and arm the copter

// Can be reset each the copter is re-armed

static struct Location home;

// Current location of the copter

static struct Location current\_loc;

// Next WP is the desired location of the copter - the next waypoint or loiter location

static struct Location next\_WP;

// Prev WP is used to get the optimum path from one WP to the next

static struct Location prev\_WP;

// Holds the current loaded command from the EEPROM for navigation

static struct Location command\_nav\_queue;

// Holds the current loaded command from the EEPROM for conditional scripts

static struct Location command\_cond\_queue;

// Holds the current loaded command from the EEPROM for guided mode

static struct Location guided\_WP;

////////////////////////////////////////////////////////////////////////////////

// Crosstrack

////////////////////////////////////////////////////////////////////////////////

// deg \* 100, The original angle to the next\_WP when the next\_WP was set

// Also used to check when we pass a WP

static int32\_t original\_wp\_bearing;

// The amount of angle correction applied to wp\_bearing to bring the copter back on its optimum path

static int16\_t crosstrack\_error;

////////////////////////////////////////////////////////////////////////////////

// Navigation Roll/Pitch functions

////////////////////////////////////////////////////////////////////////////////

// all angles are deg \* 100 : target yaw angle

// The Commanded ROll from the autopilot.

static int32\_t nav\_roll;

// The Commanded pitch from the autopilot. negative Pitch means go forward.

static int32\_t nav\_pitch;

// The desired bank towards North (Positive) or South (Negative)

static int32\_t auto\_roll;

static int32\_t auto\_pitch;

// Don't be fooled by the fact that Pitch is reversed from Roll in its sign!

static int16\_t nav\_lat;

// The desired bank towards East (Positive) or West (Negative)

static int16\_t nav\_lon;

// The Commanded ROll from the autopilot based on optical flow sensor.

static int32\_t of\_roll;

// The Commanded pitch from the autopilot based on optical flow sensor. negative Pitch means go forward.

static int32\_t of\_pitch;

////////////////////////////////////////////////////////////////////////////////

// Navigation Throttle control

////////////////////////////////////////////////////////////////////////////////

// The Commanded Throttle from the autopilot.

static int16\_t nav\_throttle; // 0-1000 for throttle control

// This is a simple counter to track the amount of throttle used during flight

// This could be useful later in determining and debuging current usage and predicting battery life

static uint32\_t throttle\_integrator;

////////////////////////////////////////////////////////////////////////////////

// Climb rate control

////////////////////////////////////////////////////////////////////////////////

// Time when we intiated command in millis - used for controlling decent rate

// Used to track the altitude offset for climbrate control

static int8\_t alt\_change\_flag;

////////////////////////////////////////////////////////////////////////////////

// Navigation Yaw control

////////////////////////////////////////////////////////////////////////////////

// The Commanded Yaw from the autopilot.

static int32\_t nav\_yaw;

static uint8\_t yaw\_timer;

// Yaw will point at this location if yaw\_mode is set to YAW\_LOOK\_AT\_LOCATION

static struct Location yaw\_look\_at\_WP;

// bearing from current location to the yaw\_look\_at\_WP

static int32\_t yaw\_look\_at\_WP\_bearing;

// yaw used for YAW\_LOOK\_AT\_HEADING yaw\_mode

static int32\_t yaw\_look\_at\_heading;

// Deg/s we should turn

static int16\_t yaw\_look\_at\_heading\_slew;

////////////////////////////////////////////////////////////////////////////////

// Repeat Mission Scripting Command

////////////////////////////////////////////////////////////////////////////////

// The type of repeating event - Toggle a servo channel, Toggle the APM1 relay, etc

static byte event\_id;

// Used to manage the timimng of repeating events

static uint32\_t event\_timer;

// How long to delay the next firing of event in millis

static uint16\_t event\_delay;

// how many times to fire : 0 = forever, 1 = do once, 2 = do twice

static int16\_t event\_repeat;

// per command value, such as PWM for servos

static int16\_t event\_value;

// the stored value used to undo commands - such as original PWM command

static int16\_t event\_undo\_value;

////////////////////////////////////////////////////////////////////////////////

// Delay Mission Scripting Command

////////////////////////////////////////////////////////////////////////////////

static int32\_t condition\_value; // used in condition commands (eg delay, change alt, etc.)

static uint32\_t condition\_start;

////////////////////////////////////////////////////////////////////////////////

// IMU variables

////////////////////////////////////////////////////////////////////////////////

// Integration time for the gyros (DCM algorithm)

// Updated with the fast loop

static float G\_Dt = 0.02;

////////////////////////////////////////////////////////////////////////////////

// Inertial Navigation

////////////////////////////////////////////////////////////////////////////////

#if INERTIAL\_NAV\_XY == ENABLED || INERTIAL\_NAV\_Z == ENABLED

AP\_InertialNav inertial\_nav(&ahrs, &ins, &barometer, &g\_gps);

#endif

////////////////////////////////////////////////////////////////////////////////

// Performance monitoring

////////////////////////////////////////////////////////////////////////////////

// Used to manage the rate of performance logging messages

static int16\_t perf\_mon\_counter;

// The number of GPS fixes we have had

static int16\_t gps\_fix\_count;

// System Timers

// --------------

// Time in microseconds of main control loop

static uint32\_t fast\_loopTimer;

// Time in microseconds of 50hz control loop

static uint32\_t fiftyhz\_loopTimer = 0;

// Counters for branching from 10 hz control loop

static byte medium\_loopCounter;

// Counters for branching from 3 1/3hz control loop

static byte slow\_loopCounter;

// Counters for branching at 1 hz

static byte counter\_one\_herz;

// Counter of main loop executions. Used for performance monitoring and failsafe processing

static uint16\_t mainLoop\_count;

// Delta Time in milliseconds for navigation computations, updated with every good GPS read

static float dTnav;

// Counters for branching from 4 minute control loop used to save Compass offsets

static int16\_t superslow\_loopCounter;

// Loiter timer - Records how long we have been in loiter

static uint32\_t rtl\_loiter\_start\_time;

// disarms the copter while in Acro or Stabilize mode after 30 seconds of no flight

static uint8\_t auto\_disarming\_counter;

// prevents duplicate GPS messages from entering system

static uint32\_t last\_gps\_time;

// Used to exit the roll and pitch auto trim function

static uint8\_t auto\_trim\_counter;

// Reference to the relay object (APM1 -> PORTL 2) (APM2 -> PORTB 7)

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

AP\_Relay\_APM2 relay;

#else

AP\_Relay\_APM1 relay;

#endif

//Reference to the camera object (it uses the relay object inside it)

#if CAMERA == ENABLED

AP\_Camera camera(&relay);

#endif

// a pin for reading the receiver RSSI voltage. The scaling by 0.25

// is to take the 0 to 1024 range down to an 8 bit range for MAVLink

AP\_AnalogSource\_Arduino RSSI\_pin(-1, 0.25);

#if CLI\_ENABLED == ENABLED

static int8\_t setup\_show (uint8\_t argc, const Menu::arg \*argv);

#endif

// Camera/Antenna mount tracking and stabilisation stuff

// --------------------------------------

#if MOUNT == ENABLED

// current\_loc uses the baro/gps soloution for altitude rather than gps only.

// mabe one could use current\_loc for lat/lon too and eliminate g\_gps alltogether?

AP\_Mount camera\_mount(&current\_loc, g\_gps, &ahrs, 0);

#endif

#if MOUNT2 == ENABLED

// current\_loc uses the baro/gps soloution for altitude rather than gps only.

// mabe one could use current\_loc for lat/lon too and eliminate g\_gps alltogether?

AP\_Mount camera\_mount2(&current\_loc, g\_gps, &ahrs, 1);

#endif

////////////////////////////////////////////////////////////////////////////////

// Experimental AP\_Limits library - set constraints, limits, fences, minima, maxima on various parameters

////////////////////////////////////////////////////////////////////////////////

#if AP\_LIMITS == ENABLED

AP\_Limits limits;

AP\_Limit\_GPSLock gpslock\_limit(g\_gps);

AP\_Limit\_Geofence geofence\_limit(FENCE\_START\_BYTE, FENCE\_WP\_SIZE, MAX\_FENCEPOINTS, g\_gps, &home, &current\_loc);

AP\_Limit\_Altitude altitude\_limit(&current\_loc);

#endif

////////////////////////////////////////////////////////////////////////////////

// function definitions to keep compiler from complaining about undeclared functions

////////////////////////////////////////////////////////////////////////////////

void get\_throttle\_althold(int32\_t target\_alt, int16\_t min\_climb\_rate, int16\_t max\_climb\_rate);

////////////////////////////////////////////////////////////////////////////////

// Top-level logic

////////////////////////////////////////////////////////////////////////////////

void setup() {

memcheck\_init();

init\_ardupilot();

}

void loop()

{

uint32\_t timer = micros();

uint16\_t num\_samples;

// We want this to execute fast

// ----------------------------

num\_samples = ins.num\_samples\_available();

if (num\_samples >= 2) {

#if DEBUG\_FAST\_LOOP == ENABLED

Log\_Write\_Data(DATA\_FAST\_LOOP, (int32\_t)(timer - fast\_loopTimer));

#endif

// check loop time

perf\_info\_check\_loop\_time(timer - fast\_loopTimer);

G\_Dt = (float)(timer - fast\_loopTimer) / 1000000.f; // used by PI Loops

fast\_loopTimer = timer;

// for mainloop failure monitoring

mainLoop\_count++;

// Execute the fast loop

// ---------------------

fast\_loop();

// run the 50hz loop 1/2 the time

ap\_system.run\_50hz\_loop = !ap\_system.run\_50hz\_loop;

if(ap\_system.run\_50hz\_loop) {

#if DEBUG\_MED\_LOOP == ENABLED

Log\_Write\_Data(DATA\_MED\_LOOP, (int32\_t)(timer - fiftyhz\_loopTimer));

#endif

// store the micros for the 50 hz timer

fiftyhz\_loopTimer = timer;

// check for new GPS messages

// --------------------------

update\_GPS();

// run navigation routines

update\_navigation();

// perform 10hz tasks

// ------------------

medium\_loop();

// Stuff to run at full 50hz, but after the med loops

// --------------------------------------------------

fifty\_hz\_loop();

counter\_one\_herz++;

// trgger our 1 hz loop

if(counter\_one\_herz >= 50) {

super\_slow\_loop();

counter\_one\_herz = 0;

}

perf\_mon\_counter++;

if (perf\_mon\_counter >= 500 ) { // 500 iterations at 50hz = 10 seconds

if (g.log\_bitmask & MASK\_LOG\_PM)

Log\_Write\_Performance();

perf\_info\_reset();

gps\_fix\_count = 0;

perf\_mon\_counter = 0;

}

}else{

// process communications with the GCS

gcs\_check();

}

} else {

#ifdef DESKTOP\_BUILD

usleep(1000);

#endif

if (timer - fast\_loopTimer < 9000) {

// we have some spare cycles available

// less than 10ms has passed. We have at least one millisecond

// of free time. The most useful thing to do with that time is

// to accumulate some sensor readings, specifically the

// compass, which is often very noisy but is not interrupt

// driven, so it can't accumulate readings by itself

if (g.compass\_enabled) {

compass.accumulate();

}

}

}

}

// Main loop - 100hz

static void fast\_loop()

{

// IMU DCM Algorithm

// --------------------

read\_AHRS();

// reads all of the necessary trig functions for cameras, throttle, etc.

// --------------------------------------------------------------------

update\_trig();

// run low level rate controllers that only require IMU data

run\_rate\_controllers();

// write out the servo PWM values

// ------------------------------

set\_servos\_4();

// Inertial Nav

// --------------------

read\_inertia();

// optical flow

// --------------------

#if OPTFLOW == ENABLED

if(g.optflow\_enabled) {

update\_optical\_flow();

}

#endif // OPTFLOW == ENABLED

// Read radio and 3-position switch on radio

// -----------------------------------------

read\_radio();

read\_control\_switch();

// custom code/exceptions for flight modes

// ---------------------------------------

update\_yaw\_mode();

update\_roll\_pitch\_mode();

// update targets to rate controllers

update\_rate\_contoller\_targets();

// agmatthews - USERHOOKS

#ifdef USERHOOK\_FASTLOOP

USERHOOK\_FASTLOOP

#endif

}

static void medium\_loop()

{

// This is the start of the medium (10 Hz) loop pieces

// -----------------------------------------

switch(medium\_loopCounter) {

// This case deals with the GPS and Compass

//-----------------------------------------

case 0:

medium\_loopCounter++;

#if HIL\_MODE != HIL\_MODE\_ATTITUDE // don't execute in HIL mode

if(g.compass\_enabled) {

if (compass.read()) {

compass.null\_offsets();

}

}

#endif

// auto\_trim - stores roll and pitch radio inputs to ahrs

auto\_trim();

// record throttle output

// ------------------------------

throttle\_integrator += g.rc\_3.servo\_out;

break;

// This case performs some navigation computations

//------------------------------------------------

case 1:

medium\_loopCounter++;

read\_receiver\_rssi();

break;

// command processing

//-------------------

case 2:

medium\_loopCounter++;

if(control\_mode == TOY\_A) {

update\_toy\_throttle();

if(throttle\_mode == THROTTLE\_AUTO) {

update\_toy\_altitude();

}

}

ap\_system.alt\_sensor\_flag = true;

break;

// This case deals with sending high rate telemetry

//-------------------------------------------------

case 3:

medium\_loopCounter++;

// perform next command

// --------------------

if(control\_mode == AUTO) {

if(ap.home\_is\_set && g.command\_total > 1) {

update\_commands();

}

}

if(motors.armed()) {

if (g.log\_bitmask & MASK\_LOG\_ATTITUDE\_MED) {

Log\_Write\_Attitude();

#if SECONDARY\_DMP\_ENABLED == ENABLED

Log\_Write\_DMP();

#endif

}

if (g.log\_bitmask & MASK\_LOG\_MOTORS)

Log\_Write\_Motors();

}

break;

// This case controls the slow loop

//---------------------------------

case 4:

medium\_loopCounter = 0;

if (g.battery\_monitoring != 0) {

read\_battery();

}

// Accel trims = hold > 2 seconds

// Throttle cruise = switch less than 1 second

// --------------------------------------------

read\_trim\_switch();

// Check for engine arming

// -----------------------

arm\_motors();

// agmatthews - USERHOOKS

#ifdef USERHOOK\_MEDIUMLOOP

USERHOOK\_MEDIUMLOOP

#endif

#if COPTER\_LEDS == ENABLED

update\_copter\_leds();

#endif

slow\_loop();

break;

default:

// this is just a catch all

// ------------------------

medium\_loopCounter = 0;

break;

}

}

// stuff that happens at 50 hz

// ---------------------------

static void fifty\_hz\_loop()

{

// read altitude sensors or estimate altitude

// ------------------------------------------

update\_altitude\_est();

// Update the throttle ouput

// -------------------------

update\_throttle\_mode();

#if TOY\_EDF == ENABLED

edf\_toy();

#endif

#ifdef USERHOOK\_50HZLOOP

USERHOOK\_50HZLOOP

#endif

#if HIL\_MODE != HIL\_MODE\_DISABLED && FRAME\_CONFIG != HELI\_FRAME

// HIL for a copter needs very fast update of the servo values

gcs\_send\_message(MSG\_RADIO\_OUT);

#endif

#if MOUNT == ENABLED

// update camera mount's position

camera\_mount.update\_mount\_position();

#endif

#if MOUNT2 == ENABLED

// update camera mount's position

camera\_mount2.update\_mount\_position();

#endif

#if CAMERA == ENABLED

camera.trigger\_pic\_cleanup();

#endif

# if HIL\_MODE == HIL\_MODE\_DISABLED

if (g.log\_bitmask & MASK\_LOG\_ATTITUDE\_FAST && motors.armed()) {

Log\_Write\_Attitude();

#if SECONDARY\_DMP\_ENABLED == ENABLED

Log\_Write\_DMP();

#endif

}

if (g.log\_bitmask & MASK\_LOG\_RAW && motors.armed())

Log\_Write\_Raw();

#endif

}

static void slow\_loop()

{

#if AP\_LIMITS == ENABLED

// Run the AP\_Limits main loop

limits\_loop();

#endif // AP\_LIMITS\_ENABLED

// This is the slow (3 1/3 Hz) loop pieces

//----------------------------------------

switch (slow\_loopCounter) {

case 0:

slow\_loopCounter++;

superslow\_loopCounter++;

// record if the compass is healthy

set\_compass\_healthy(compass.healthy);

if(superslow\_loopCounter > 1200) {

#if HIL\_MODE != HIL\_MODE\_ATTITUDE

if(g.rc\_3.control\_in == 0 && control\_mode == STABILIZE && g.compass\_enabled) {

compass.save\_offsets();

superslow\_loopCounter = 0;

}

#endif

}

if(motors.armed()) {

if (g.log\_bitmask & MASK\_LOG\_ITERM)

Log\_Write\_Iterm();

}else{

// check the user hasn't updated the frame orientation

motors.set\_frame\_orientation(g.frame\_orientation);

}

break;

case 1:

slow\_loopCounter++;

#if MOUNT == ENABLED

update\_aux\_servo\_function(&g.rc\_5, &g.rc\_6, &g.rc\_7, &g.rc\_8, &g.rc\_10, &g.rc\_11);

#endif

enable\_aux\_servos();

#if MOUNT == ENABLED

camera\_mount.update\_mount\_type();

#endif

#if MOUNT2 == ENABLED

camera\_mount2.update\_mount\_type();

#endif

// agmatthews - USERHOOKS

#ifdef USERHOOK\_SLOWLOOP

USERHOOK\_SLOWLOOP

#endif

break;

case 2:

slow\_loopCounter = 0;

update\_events();

// blink if we are armed

update\_lights();

if(g.radio\_tuning > 0)

tuning();

#if USB\_MUX\_PIN > 0

check\_usb\_mux();

#endif

break;

default:

slow\_loopCounter = 0;

break;

}

}

#define AUTO\_DISARMING\_DELAY 25

// 1Hz loop

static void super\_slow\_loop()

{

Log\_Write\_Data(DATA\_AP\_STATE, ap.value);

if (g.log\_bitmask & MASK\_LOG\_CUR && motors.armed())

Log\_Write\_Current();

// this function disarms the copter if it has been sitting on the ground for any moment of time greater than 25 seconds

// but only of the control mode is manual

if((control\_mode <= ACRO) && (g.rc\_3.control\_in == 0)) {

auto\_disarming\_counter++;

if(auto\_disarming\_counter == AUTO\_DISARMING\_DELAY) {

init\_disarm\_motors();

}else if (auto\_disarming\_counter > AUTO\_DISARMING\_DELAY) {

auto\_disarming\_counter = AUTO\_DISARMING\_DELAY + 1;

}

}else{

auto\_disarming\_counter = 0;

}

// agmatthews - USERHOOKS

#ifdef USERHOOK\_SUPERSLOWLOOP

USERHOOK\_SUPERSLOWLOOP

#endif

}

// called at 100hz but data from sensor only arrives at 20 Hz

#if OPTFLOW == ENABLED

static void update\_optical\_flow(void)

{

static uint32\_t last\_of\_update = 0;

static uint8\_t of\_log\_counter = 0;

// if new data has arrived, process it

if( optflow.last\_update != last\_of\_update ) {

last\_of\_update = optflow.last\_update;

optflow.update\_position(ahrs.roll, ahrs.pitch, cos\_yaw\_x, sin\_yaw\_y, current\_loc.alt); // updates internal lon and lat with estimation based on optical flow

// write to log at 5hz

of\_log\_counter++;

if( of\_log\_counter >= 4 ) {

of\_log\_counter = 0;

if (g.log\_bitmask & MASK\_LOG\_OPTFLOW) {

Log\_Write\_Optflow();

}

}

}

}

#endif // OPTFLOW == ENABLED

// called at 50hz

static void update\_GPS(void)

{

// A counter that is used to grab at least 10 reads before commiting the Home location

static byte ground\_start\_count = 10;

g\_gps->update();

update\_GPS\_light();

set\_gps\_healthy(g\_gps->status() == g\_gps->GPS\_OK);

if (g\_gps->new\_data && g\_gps->fix) {

// clear new data flag

g\_gps->new\_data = false;

// check for duiplicate GPS messages

if(last\_gps\_time != g\_gps->time) {

// for performance monitoring

// --------------------------

gps\_fix\_count++;

if(ground\_start\_count > 1) {

ground\_start\_count--;

} else if (ground\_start\_count == 1) {

// We countdown N number of good GPS fixes

// so that the altitude is more accurate

// -------------------------------------

if (current\_loc.lat == 0) {

ground\_start\_count = 5;

}else{

if (g.compass\_enabled) {

// Set compass declination automatically

compass.set\_initial\_location(g\_gps->latitude, g\_gps->longitude);

}

// save home to eeprom (we must have a good fix to have reached this point)

init\_home();

ground\_start\_count = 0;

}

}

if (g.log\_bitmask & MASK\_LOG\_GPS && motors.armed()) {

Log\_Write\_GPS();

}

#if HIL\_MODE == HIL\_MODE\_ATTITUDE // only execute in HIL mode

ap\_system.alt\_sensor\_flag = true;

#endif

}

// save GPS time so we don't get duplicate reads

last\_gps\_time = g\_gps->time;

}

}

// set\_yaw\_mode - update yaw mode and initialise any variables required

bool set\_yaw\_mode(uint8\_t new\_yaw\_mode)

{

// boolean to ensure proper initialisation of throttle modes

bool yaw\_initialised = false;

// return immediately if no change

if( new\_yaw\_mode == yaw\_mode ) {

return true;

}

switch( new\_yaw\_mode ) {

case YAW\_HOLD:

case YAW\_ACRO:

yaw\_initialised = true;

break;

case YAW\_LOOK\_AT\_NEXT\_WP:

if( ap.home\_is\_set ) {

yaw\_initialised = true;

}

break;

case YAW\_LOOK\_AT\_LOCATION:

if( ap.home\_is\_set ) {

// update bearing - assumes yaw\_look\_at\_WP has been intialised before set\_yaw\_mode was called

yaw\_look\_at\_WP\_bearing = get\_bearing\_cd(&current\_loc, &yaw\_look\_at\_WP);

yaw\_initialised = true;

}

break;

case YAW\_LOOK\_AT\_HEADING:

yaw\_initialised = true;

break;

case YAW\_LOOK\_AT\_HOME:

if( ap.home\_is\_set ) {

yaw\_initialised = true;

}

break;

case YAW\_TOY:

yaw\_initialised = true;

break;

case YAW\_LOOK\_AHEAD:

if( ap.home\_is\_set ) {

yaw\_initialised = true;

}

break;

}

// if initialisation has been successful update the yaw mode

if( yaw\_initialised ) {

yaw\_mode = new\_yaw\_mode;

}

// return success or failure

return yaw\_initialised;

}

// update\_yaw\_mode - run high level yaw controllers

// 100hz update rate

void update\_yaw\_mode(void)

{

switch(yaw\_mode) {

case YAW\_HOLD:

// heading hold at heading held in nav\_yaw but allow input from pilot

get\_yaw\_rate\_stabilized\_ef(g.rc\_4.control\_in);

break;

case YAW\_ACRO:

// pilot controlled yaw using rate controller

if(g.axis\_enabled) {

get\_yaw\_rate\_stabilized\_ef(g.rc\_4.control\_in);

}else{

get\_acro\_yaw(g.rc\_4.control\_in);

}

break;

case YAW\_LOOK\_AT\_NEXT\_WP:

// point towards next waypoint (no pilot input accepted)

// we don't use wp\_bearing because we don't want the copter to turn too much during flight

nav\_yaw = get\_yaw\_slew(nav\_yaw, original\_wp\_bearing, AUTO\_YAW\_SLEW\_RATE);

get\_stabilize\_yaw(nav\_yaw);

// if there is any pilot input, switch to YAW\_HOLD mode for the next iteration

if( g.rc\_4.control\_in != 0 ) {

set\_yaw\_mode(YAW\_HOLD);

}

break;

case YAW\_LOOK\_AT\_LOCATION:

// point towards a location held in yaw\_look\_at\_WP (no pilot input accepted)

nav\_yaw = get\_yaw\_slew(nav\_yaw, yaw\_look\_at\_WP\_bearing, AUTO\_YAW\_SLEW\_RATE);

get\_stabilize\_yaw(nav\_yaw);

// if there is any pilot input, switch to YAW\_HOLD mode for the next iteration

if( g.rc\_4.control\_in != 0 ) {

set\_yaw\_mode(YAW\_HOLD);

}

break;

case YAW\_LOOK\_AT\_HOME:

// keep heading always pointing at home with no pilot input allowed

nav\_yaw = get\_yaw\_slew(nav\_yaw, home\_bearing, AUTO\_YAW\_SLEW\_RATE);

get\_stabilize\_yaw(nav\_yaw);

// if there is any pilot input, switch to YAW\_HOLD mode for the next iteration

if( g.rc\_4.control\_in != 0 ) {

set\_yaw\_mode(YAW\_HOLD);

}

break;

case YAW\_LOOK\_AT\_HEADING:

// keep heading pointing in the direction held in yaw\_look\_at\_heading with no pilot input allowed

nav\_yaw = get\_yaw\_slew(nav\_yaw, yaw\_look\_at\_heading, yaw\_look\_at\_heading\_slew);

get\_stabilize\_yaw(nav\_yaw);

break;

case YAW\_LOOK\_AHEAD:

// Commanded Yaw to automatically look ahead.

get\_look\_ahead\_yaw(g.rc\_4.control\_in);

break;

#if TOY\_LOOKUP == TOY\_EXTERNAL\_MIXER

case YAW\_TOY:

// update to allow external roll/yaw mixing

// keep heading always pointing at home with no pilot input allowed

nav\_yaw = get\_yaw\_slew(nav\_yaw, home\_bearing, AUTO\_YAW\_SLEW\_RATE);

get\_stabilize\_yaw(nav\_yaw);

break;

#endif

}

}

// set\_roll\_pitch\_mode - update roll/pitch mode and initialise any variables as required

bool set\_roll\_pitch\_mode(uint8\_t new\_roll\_pitch\_mode)

{

// boolean to ensure proper initialisation of throttle modes

bool roll\_pitch\_initialised = false;

// return immediately if no change

if( new\_roll\_pitch\_mode == roll\_pitch\_mode ) {

return true;

}

switch( new\_roll\_pitch\_mode ) {

case ROLL\_PITCH\_STABLE:

case ROLL\_PITCH\_ACRO:

case ROLL\_PITCH\_AUTO:

case ROLL\_PITCH\_STABLE\_OF:

case ROLL\_PITCH\_TOY:

case ROLL\_PITCH\_LOITER\_PR:

roll\_pitch\_initialised = true;

break;

}

// if initialisation has been successful update the yaw mode

if( roll\_pitch\_initialised ) {

roll\_pitch\_mode = new\_roll\_pitch\_mode;

}

// return success or failure

return roll\_pitch\_initialised;

}

// update\_roll\_pitch\_mode - run high level roll and pitch controllers

// 100hz update rate

void update\_roll\_pitch\_mode(void)

{

if (ap.do\_flip) {

if(abs(g.rc\_1.control\_in) < 4000) {

roll\_flip();

return;

}else{

// force an exit from the loop if we are not hands off sticks.

ap.do\_flip = false;

Log\_Write\_Event(DATA\_EXIT\_FLIP);

}

}

switch(roll\_pitch\_mode) {

case ROLL\_PITCH\_ACRO:

#if FRAME\_CONFIG == HELI\_FRAME

if(g.axis\_enabled) {

get\_roll\_rate\_stabilized\_ef(g.rc\_1.control\_in);

get\_pitch\_rate\_stabilized\_ef(g.rc\_2.control\_in);

}else{

// ACRO does not get SIMPLE mode ability

if (motors.flybar\_mode == 1) {

g.rc\_1.servo\_out = g.rc\_1.control\_in;

g.rc\_2.servo\_out = g.rc\_2.control\_in;

} else {

get\_acro\_roll(g.rc\_1.control\_in);

get\_acro\_pitch(g.rc\_2.control\_in);

}

}

#else // !HELI\_FRAME

if(g.axis\_enabled) {

get\_roll\_rate\_stabilized\_ef(g.rc\_1.control\_in);

get\_pitch\_rate\_stabilized\_ef(g.rc\_2.control\_in);

}else{

// ACRO does not get SIMPLE mode ability

get\_acro\_roll(g.rc\_1.control\_in);

get\_acro\_pitch(g.rc\_2.control\_in);

}

#endif // HELI\_FRAME

break;

case ROLL\_PITCH\_STABLE:

// apply SIMPLE mode transform

if(ap.simple\_mode && ap\_system.new\_radio\_frame) {

update\_simple\_mode();

}

control\_roll = g.rc\_1.control\_in;

control\_pitch = g.rc\_2.control\_in;

get\_stabilize\_roll(control\_roll);

get\_stabilize\_pitch(control\_pitch);

break;

case ROLL\_PITCH\_AUTO:

// apply SIMPLE mode transform

if(ap.simple\_mode && ap\_system.new\_radio\_frame) {

update\_simple\_mode();

}

// mix in user control with Nav control

nav\_roll += constrain(wrap\_180(auto\_roll - nav\_roll), -g.auto\_slew\_rate.get(), g.auto\_slew\_rate.get()); // 40 deg a second

nav\_pitch += constrain(wrap\_180(auto\_pitch - nav\_pitch), -g.auto\_slew\_rate.get(), g.auto\_slew\_rate.get()); // 40 deg a second

control\_roll = g.rc\_1.control\_mix(nav\_roll);

control\_pitch = g.rc\_2.control\_mix(nav\_pitch);

get\_stabilize\_roll(control\_roll);

get\_stabilize\_pitch(control\_pitch);

break;

case ROLL\_PITCH\_STABLE\_OF:

// apply SIMPLE mode transform

if(ap.simple\_mode && ap\_system.new\_radio\_frame) {

update\_simple\_mode();

}

control\_roll = g.rc\_1.control\_in;

control\_pitch = g.rc\_2.control\_in;

// mix in user control with optical flow

get\_stabilize\_roll(get\_of\_roll(control\_roll));

get\_stabilize\_pitch(get\_of\_pitch(control\_pitch));

break;

// THOR

// a call out to the main toy logic

case ROLL\_PITCH\_TOY:

roll\_pitch\_toy();

break;

case ROLL\_PITCH\_LOITER\_PR:

// LOITER does not get SIMPLE mode ability

nav\_roll += constrain(wrap\_180(auto\_roll - nav\_roll), -g.auto\_slew\_rate.get(), g.auto\_slew\_rate.get()); // 40 deg a second

nav\_pitch += constrain(wrap\_180(auto\_pitch - nav\_pitch), -g.auto\_slew\_rate.get(), g.auto\_slew\_rate.get()); // 40 deg a second

get\_stabilize\_roll(nav\_roll);

get\_stabilize\_pitch(nav\_pitch);

break;

}

#if FRAME\_CONFIG != HELI\_FRAME

if(g.rc\_3.control\_in == 0 && control\_mode <= ACRO) {

reset\_rate\_I();

reset\_stability\_I();

}

#endif //HELI\_FRAME

if(ap\_system.new\_radio\_frame) {

// clear new radio frame info

ap\_system.new\_radio\_frame = false;

}

}

// new radio frame is used to make sure we only call this at 50hz

void update\_simple\_mode(void)

{

static byte simple\_counter = 0; // State machine counter for Simple Mode

static float simple\_sin\_y=0, simple\_cos\_x=0;

// used to manage state machine

// which improves speed of function

simple\_counter++;

int16\_t delta = wrap\_360(ahrs.yaw\_sensor - initial\_simple\_bearing)/100;

if (simple\_counter == 1) {

// roll

simple\_cos\_x = sin(radians(90 - delta));

}else if (simple\_counter > 2) {

// pitch

simple\_sin\_y = cos(radians(90 - delta));

simple\_counter = 0;

}

// Rotate input by the initial bearing

int16\_t \_roll = g.rc\_1.control\_in \* simple\_cos\_x + g.rc\_2.control\_in \* simple\_sin\_y;

int16\_t \_pitch = -(g.rc\_1.control\_in \* simple\_sin\_y - g.rc\_2.control\_in \* simple\_cos\_x);

g.rc\_1.control\_in = \_roll;

g.rc\_2.control\_in = \_pitch;

}

// set\_throttle\_mode - sets the throttle mode and initialises any variables as required

bool set\_throttle\_mode( uint8\_t new\_throttle\_mode )

{

// boolean to ensure proper initialisation of throttle modes

bool throttle\_initialised = false;

// return immediately if no change

if( new\_throttle\_mode == throttle\_mode ) {

return true;

}

// initialise any variables required for the new throttle mode

switch(new\_throttle\_mode) {

case THROTTLE\_MANUAL:

case THROTTLE\_MANUAL\_TILT\_COMPENSATED:

throttle\_accel\_deactivate(); // this controller does not use accel based throttle controller

altitude\_error = 0; // clear altitude error reported to GCS

throttle\_initialised = true;

break;

case THROTTLE\_ACCELERATION: // pilot inputs the desired acceleration

if( g.throttle\_accel\_enabled ) { // this throttle mode requires use of the accel based throttle controller

altitude\_error = 0; // clear altitude error reported to GCS

throttle\_initialised = true;

}

break;

case THROTTLE\_RATE:

altitude\_error = 0; // clear altitude error reported to GCS

throttle\_initialised = true;

break;

case THROTTLE\_STABILIZED\_RATE:

case THROTTLE\_DIRECT\_ALT:

controller\_desired\_alt = current\_loc.alt; // reset controller desired altitude to current altitude

throttle\_initialised = true;

break;

case THROTTLE\_HOLD:

case THROTTLE\_AUTO:

controller\_desired\_alt = current\_loc.alt; // reset controller desired altitude to current altitude

set\_new\_altitude(current\_loc.alt); // by default hold the current altitude

if ( throttle\_mode <= THROTTLE\_MANUAL\_TILT\_COMPENSATED ) { // reset the alt hold I terms if previous throttle mode was manual

reset\_throttle\_I();

set\_accel\_throttle\_I\_from\_pilot\_throttle(get\_pilot\_desired\_throttle(g.rc\_3.control\_in));

}

throttle\_initialised = true;

break;

case THROTTLE\_LAND:

set\_land\_complete(false); // mark landing as incomplete

land\_detector = 0; // A counter that goes up if our climb rate stalls out.

controller\_desired\_alt = current\_loc.alt; // reset controller desired altitude to current altitude

// Set target altitude to LAND\_START\_ALT if we are high, below this altitude the get\_throttle\_rate\_stabilized will take care of setting the next\_WP.alt

if (current\_loc.alt >= LAND\_START\_ALT) {

set\_new\_altitude(LAND\_START\_ALT);

}

throttle\_initialised = true;

break;

default:

// To-Do: log an error message to the dataflash or tlogs instead of printing to the serial port

cliSerial->printf\_P(PSTR("Unsupported throttle mode: %d!!"),new\_throttle\_mode);

break;

}

// update the throttle mode

if( throttle\_initialised ) {

throttle\_mode = new\_throttle\_mode;

// reset some variables used for logging

desired\_climb\_rate = 0;

nav\_throttle = 0;

}

// return success or failure

return throttle\_initialised;

}

// update\_throttle\_mode - run high level throttle controllers

// 50 hz update rate

void update\_throttle\_mode(void)

{

int16\_t pilot\_climb\_rate;

int16\_t pilot\_throttle\_scaled;

if(ap.do\_flip) // this is pretty bad but needed to flip in AP modes.

return;

// do not run throttle controllers if motors disarmed

if( !motors.armed() ) {

set\_throttle\_out(0, false);

throttle\_accel\_deactivate(); // do not allow the accel based throttle to override our command

return;

}

#if FRAME\_CONFIG == HELI\_FRAME

if (control\_mode == STABILIZE){

motors.stab\_throttle = true;

} else {

motors.stab\_throttle = false;

}

#endif // HELI\_FRAME

switch(throttle\_mode) {

case THROTTLE\_MANUAL:

// completely manual throttle

if(g.rc\_3.control\_in <= 0){

set\_throttle\_out(0, false);

}else{

// send pilot's output directly to motors

pilot\_throttle\_scaled = get\_pilot\_desired\_throttle(g.rc\_3.control\_in);

set\_throttle\_out(pilot\_throttle\_scaled, false);

// update estimate of throttle cruise

#if FRAME\_CONFIG == HELI\_FRAME

update\_throttle\_cruise(motors.coll\_out);

#else

update\_throttle\_cruise(pilot\_throttle\_scaled);

#endif //HELI\_FRAME

// check if we've taken off yet

if (!ap.takeoff\_complete && motors.armed()) {

if (pilot\_throttle\_scaled > g.throttle\_cruise) {

// we must be in the air by now

set\_takeoff\_complete(true);

}

}

}

break;

case THROTTLE\_MANUAL\_TILT\_COMPENSATED:

// manual throttle but with angle boost

if (g.rc\_3.control\_in <= 0) {

set\_throttle\_out(0, false); // no need for angle boost with zero throttle

}else{

pilot\_throttle\_scaled = get\_pilot\_desired\_throttle(g.rc\_3.control\_in);

set\_throttle\_out(pilot\_throttle\_scaled, true);

// update estimate of throttle cruise

#if FRAME\_CONFIG == HELI\_FRAME

update\_throttle\_cruise(motors.coll\_out);

#else

update\_throttle\_cruise(pilot\_throttle\_scaled);

#endif //HELI\_FRAME

if (!ap.takeoff\_complete && motors.armed()) {

if (pilot\_throttle\_scaled > g.throttle\_cruise) {

// we must be in the air by now

set\_takeoff\_complete(true);

}

}

}

break;

case THROTTLE\_ACCELERATION:

// pilot inputs the desired acceleration

if(g.rc\_3.control\_in <= 0){

set\_throttle\_out(0, false);

throttle\_accel\_deactivate(); // do not allow the accel based throttle to override our command

}else{

int16\_t desired\_acceleration = get\_pilot\_desired\_acceleration(g.rc\_3.control\_in);

set\_throttle\_accel\_target(desired\_acceleration);

}

break;

case THROTTLE\_RATE:

// pilot inputs the desired climb rate. Note this is the unstabilized rate controller

if(g.rc\_3.control\_in <= 0){

set\_throttle\_out(0, false);

throttle\_accel\_deactivate(); // do not allow the accel based throttle to override our command

}else{

pilot\_climb\_rate = get\_pilot\_desired\_climb\_rate(g.rc\_3.control\_in);

get\_throttle\_rate(pilot\_climb\_rate);

}

break;

case THROTTLE\_STABILIZED\_RATE:

// pilot inputs the desired climb rate. Note this is the stabilized rate controller

if(g.rc\_3.control\_in <= 0){

set\_throttle\_out(0, false);

throttle\_accel\_deactivate(); // do not allow the accel based throttle to override our command

altitude\_error = 0; // clear altitude error reported to GCS - normally underlying alt hold controller updates altitude error reported to GCS

}else{

pilot\_climb\_rate = get\_pilot\_desired\_climb\_rate(g.rc\_3.control\_in);

get\_throttle\_rate\_stabilized(pilot\_climb\_rate);

}

break;

case THROTTLE\_DIRECT\_ALT:

// pilot inputs a desired altitude from 0 ~ 10 meters

if(g.rc\_3.control\_in <= 0){

set\_throttle\_out(0, false);

throttle\_accel\_deactivate(); // do not allow the accel based throttle to override our command

altitude\_error = 0; // clear altitude error reported to GCS - normally underlying alt hold controller updates altitude error reported to GCS

}else{

int32\_t desired\_alt = get\_pilot\_desired\_direct\_alt(g.rc\_3.control\_in);

get\_throttle\_althold\_with\_slew(desired\_alt, g.auto\_velocity\_z\_min, g.auto\_velocity\_z\_max);

}

break;

case THROTTLE\_HOLD:

// alt hold plus pilot input of climb rate

pilot\_climb\_rate = get\_pilot\_desired\_climb\_rate(g.rc\_3.control\_in);

if( sonar\_alt\_health >= SONAR\_ALT\_HEALTH\_MAX ) {

// if sonar is ok, use surface tracking

get\_throttle\_surface\_tracking(pilot\_climb\_rate);

}else{

// if no sonar fall back stabilize rate controller

get\_throttle\_rate\_stabilized(pilot\_climb\_rate);

}

break;

case THROTTLE\_AUTO:

// auto pilot altitude controller with target altitude held in next\_WP.alt

if(motors.auto\_armed() == true) {

get\_throttle\_althold\_with\_slew(next\_WP.alt, g.auto\_velocity\_z\_min, g.auto\_velocity\_z\_max);

}

break;

case THROTTLE\_LAND:

// landing throttle controller

get\_throttle\_land();

break;

}

}

static void read\_AHRS(void)

{

// Perform IMU calculations and get attitude info

//-----------------------------------------------

#if HIL\_MODE != HIL\_MODE\_DISABLED

// update hil before ahrs update

gcs\_update();

#endif

ahrs.update();

omega = ins.get\_gyro();

#if SECONDARY\_DMP\_ENABLED == ENABLED

ahrs2.update();

#endif

}

static void update\_trig(void){

Vector2f yawvector;

Matrix3f temp = ahrs.get\_dcm\_matrix();

yawvector.x = temp.a.x; // sin

yawvector.y = temp.b.x; // cos

yawvector.normalize();

cos\_pitch\_x = safe\_sqrt(1 - (temp.c.x \* temp.c.x)); // level = 1

cos\_roll\_x = temp.c.z / cos\_pitch\_x; // level = 1

cos\_pitch\_x = constrain(cos\_pitch\_x, 0, 1.0);

// this relies on constrain() of infinity doing the right thing,

// which it does do in avr-libc

cos\_roll\_x = constrain(cos\_roll\_x, -1.0, 1.0);

sin\_yaw\_y = yawvector.x; // 1y = north

cos\_yaw\_x = yawvector.y; // 0x = north

// added to convert earth frame to body frame for rate controllers

sin\_pitch = -temp.c.x;

sin\_roll = temp.c.y / cos\_pitch\_x;

//flat:

// 0 ° = cos\_yaw: 0.00, sin\_yaw: 1.00,

// 90° = cos\_yaw: 1.00, sin\_yaw: 0.00,

// 180 = cos\_yaw: 0.00, sin\_yaw: -1.00,

// 270 = cos\_yaw: -1.00, sin\_yaw: 0.00,

}

// updated at 10hz

static void update\_altitude()

{

int32\_t old\_baro\_alt = baro\_alt;

int16\_t old\_sonar\_alt = sonar\_alt;

#if HIL\_MODE == HIL\_MODE\_ATTITUDE

// we are in the SIM, fake out the baro and Sonar

int16\_t fake\_relative\_alt = g\_gps->altitude - gps\_base\_alt;

baro\_alt = fake\_relative\_alt;

baro\_rate = (baro\_alt - old\_baro\_alt) \* 5; // 5hz

if(g.sonar\_enabled) {

sonar\_alt = fake\_relative\_alt;

sonar\_rate = baro\_rate;

}

current\_loc.alt = baro\_alt;

climb\_rate\_actual = baro\_rate;

#else

// read in actual baro altitude

baro\_alt = read\_barometer();

// calc baro based vertical velocity

int16\_t temp = (baro\_alt - old\_baro\_alt) \* 10;

baro\_rate = (temp + baro\_rate) >> 1;

baro\_rate = constrain(baro\_rate, -500, 500);

// read in sonar altitude and calculate sonar rate

sonar\_alt = read\_sonar();

// start calculating the sonar\_rate as soon as valid sonar readings start coming in so that we are ready when the sonar\_alt\_health becomes 3

// Note: post 2.9.1 release we will remove the sonar\_rate variable completely

if(sonar\_alt\_health > 1) {

sonar\_rate = (sonar\_alt - old\_sonar\_alt) \* 10;

sonar\_rate = constrain(sonar\_rate, -150, 150);

}

// Note: with inertial nav, alt and rate are pulled from the inav lib at 50hz in update\_altitude\_est function

// so none of the below is required

# if INERTIAL\_NAV\_Z != ENABLED

// if no sonar set current alt to baro alt

if(!g.sonar\_enabled) {

// NO Sonar case

current\_loc.alt = baro\_alt;

climb\_rate\_actual = baro\_rate;

}else{

// Blend barometer and sonar data together

float scale;

if(baro\_alt < 800) {

scale = (float)(sonar\_alt - 400) / 200.0;

scale = constrain(scale, 0.0, 1.0);

// solve for a blended altitude

current\_loc.alt = ((float)sonar\_alt \* (1.0 - scale)) + ((float)baro\_alt \* scale);

// solve for a blended climb\_rate

climb\_rate\_actual = ((float)sonar\_rate \* (1.0 - scale)) + (float)baro\_rate \* scale;

}else{

// we must be higher than sonar (>800), don't get tricked by bad sonar reads

current\_loc.alt = baro\_alt;

// dont blend, go straight baro

climb\_rate\_actual = baro\_rate;

}

}

// climb\_rate\_error is used to spread the change in climb rate across the next 5 samples

climb\_rate\_error = (climb\_rate\_actual - climb\_rate) / 5;

# endif // INERTIAL\_NAV\_Z != ENABLED

#endif // HIL\_MODE == HIL\_MODE\_ATTITUDE

// update the target altitude

verify\_altitude();

}

static void update\_altitude\_est()

{

#if INERTIAL\_NAV\_Z == ENABLED

// with inertial nav we can update the altitude and climb rate at 50hz

current\_loc.alt = inertial\_nav.get\_altitude();

climb\_rate = inertial\_nav.get\_velocity\_z();

// update baro and sonar alt and climb rate just for logging purposes

// To-Do: remove alt\_sensor\_flag and move update\_altitude to be called from 10hz loop

if(ap\_system.alt\_sensor\_flag) {

ap\_system.alt\_sensor\_flag = false;

update\_altitude();

if(g.log\_bitmask & MASK\_LOG\_CTUN && motors.armed()) {

Log\_Write\_Control\_Tuning();

}

}

#else

if(ap\_system.alt\_sensor\_flag) {

update\_altitude();

ap\_system.alt\_sensor\_flag = false;

if(g.log\_bitmask & MASK\_LOG\_CTUN && motors.armed()) {

Log\_Write\_Control\_Tuning();

}

}else{

// simple dithering of climb rate

climb\_rate += climb\_rate\_error;

current\_loc.alt += (climb\_rate / 50);

}

#endif

}

static void tuning(){

tuning\_value = (float)g.rc\_6.control\_in / 1000.0;

g.rc\_6.set\_range(g.radio\_tuning\_low,g.radio\_tuning\_high); // 0 to 1

switch(g.radio\_tuning) {

case CH6\_RATE\_KD:

g.pid\_rate\_roll.kD(tuning\_value);

g.pid\_rate\_pitch.kD(tuning\_value);

break;

case CH6\_STABILIZE\_KP:

g.pi\_stabilize\_roll.kP(tuning\_value);

g.pi\_stabilize\_pitch.kP(tuning\_value);

break;

case CH6\_STABILIZE\_KI:

g.pi\_stabilize\_roll.kI(tuning\_value);

g.pi\_stabilize\_pitch.kI(tuning\_value);

break;

case CH6\_ACRO\_KP:

g.acro\_p = tuning\_value;

break;

case CH6\_RATE\_KP:

g.pid\_rate\_roll.kP(tuning\_value);

g.pid\_rate\_pitch.kP(tuning\_value);

break;

case CH6\_RATE\_KI:

g.pid\_rate\_roll.kI(tuning\_value);

g.pid\_rate\_pitch.kI(tuning\_value);

break;

case CH6\_YAW\_KP:

g.pi\_stabilize\_yaw.kP(tuning\_value);

break;

case CH6\_YAW\_KI:

g.pi\_stabilize\_yaw.kI(tuning\_value);

break;

case CH6\_YAW\_RATE\_KP:

g.pid\_rate\_yaw.kP(tuning\_value);

break;

case CH6\_YAW\_RATE\_KD:

g.pid\_rate\_yaw.kD(tuning\_value);

break;

case CH6\_THROTTLE\_KP:

g.pid\_throttle.kP(tuning\_value);

break;

case CH6\_THROTTLE\_KI:

g.pid\_throttle.kI(tuning\_value);

break;

case CH6\_THROTTLE\_KD:

g.pid\_throttle.kD(tuning\_value);

break;

case CH6\_TOP\_BOTTOM\_RATIO:

motors.top\_bottom\_ratio = tuning\_value;

break;

case CH6\_RELAY:

if (g.rc\_6.control\_in > 525) relay.on();

if (g.rc\_6.control\_in < 475) relay.off();

break;

case CH6\_TRAVERSE\_SPEED:

g.waypoint\_speed\_max = g.rc\_6.control\_in;

break;

case CH6\_LOITER\_KP:

g.pi\_loiter\_lat.kP(tuning\_value);

g.pi\_loiter\_lon.kP(tuning\_value);

break;

case CH6\_LOITER\_KI:

g.pi\_loiter\_lat.kI(tuning\_value);

g.pi\_loiter\_lon.kI(tuning\_value);

break;

case CH6\_NAV\_KP:

g.pid\_nav\_lat.kP(tuning\_value);

g.pid\_nav\_lon.kP(tuning\_value);

break;

case CH6\_LOITER\_RATE\_KP:

g.pid\_loiter\_rate\_lon.kP(tuning\_value);

g.pid\_loiter\_rate\_lat.kP(tuning\_value);

break;

case CH6\_LOITER\_RATE\_KI:

g.pid\_loiter\_rate\_lon.kI(tuning\_value);

g.pid\_loiter\_rate\_lat.kI(tuning\_value);

break;

case CH6\_LOITER\_RATE\_KD:

g.pid\_loiter\_rate\_lon.kD(tuning\_value);

g.pid\_loiter\_rate\_lat.kD(tuning\_value);

break;

case CH6\_NAV\_KI:

g.pid\_nav\_lat.kI(tuning\_value);

g.pid\_nav\_lon.kI(tuning\_value);

break;

#if FRAME\_CONFIG == HELI\_FRAME

case CH6\_HELI\_EXTERNAL\_GYRO:

motors.ext\_gyro\_gain = tuning\_value;

break;

#endif

case CH6\_THR\_HOLD\_KP:

g.pi\_alt\_hold.kP(tuning\_value);

break;

case CH6\_OPTFLOW\_KP:

g.pid\_optflow\_roll.kP(tuning\_value);

g.pid\_optflow\_pitch.kP(tuning\_value);

break;

case CH6\_OPTFLOW\_KI:

g.pid\_optflow\_roll.kI(tuning\_value);

g.pid\_optflow\_pitch.kI(tuning\_value);

break;

case CH6\_OPTFLOW\_KD:

g.pid\_optflow\_roll.kD(tuning\_value);

g.pid\_optflow\_pitch.kD(tuning\_value);

break;

#if HIL\_MODE != HIL\_MODE\_ATTITUDE // do not allow modifying \_kp or \_kp\_yaw gains in HIL mode

case CH6\_AHRS\_YAW\_KP:

ahrs.\_kp\_yaw.set(tuning\_value);

break;

case CH6\_AHRS\_KP:

ahrs.\_kp.set(tuning\_value);

break;

#endif

case CH6\_INAV\_TC:

#if INERTIAL\_NAV\_XY == ENABLED

inertial\_nav.set\_time\_constant\_xy(tuning\_value);

#endif

#if INERTIAL\_NAV\_Z == ENABLED

inertial\_nav.set\_time\_constant\_z(tuning\_value);

#endif

break;

case CH6\_THR\_ACCEL\_KP:

g.pid\_throttle\_accel.kP(tuning\_value);

break;

case CH6\_THR\_ACCEL\_KI:

g.pid\_throttle\_accel.kI(tuning\_value);

break;

case CH6\_THR\_ACCEL\_KD:

g.pid\_throttle\_accel.kD(tuning\_value);

break;

}

}

## APM\_Config.h

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

// Example config file. Take a look at config.h. Any term define there can be overridden by defining it here.

//#define CONFIG\_APM\_HARDWARE APM\_HARDWARE\_APM2

// Ordinary users should please ignore the following define.

// APM2\_BETA\_HARDWARE is used to support early (September-October 2011) APM2

// hardware which had the BMP085 barometer onboard. Only a handful of

// developers have these boards.

//#define APM2\_BETA\_HARDWARE

//#define MAG\_ORIENTATION AP\_COMPASS\_COMPONENTS\_DOWN\_PINS\_FORWARD

//#define HIL\_MODE HIL\_MODE\_ATTITUDE

//#define DMP\_ENABLED ENABLED

//#define SECONDARY\_DMP\_ENABLED ENABLED // allows running DMP in parallel with DCM for testing purposes

//#define FRAME\_CONFIG QUAD\_FRAME

/\*

\* options:

\* QUAD\_FRAME

\* TRI\_FRAME

\* HEXA\_FRAME

\* Y6\_FRAME

\* OCTA\_FRAME

\* OCTA\_QUAD\_FRAME

\* HELI\_FRAME

\*/

//#define FRAME\_ORIENTATION X\_FRAME

/\*

\* PLUS\_FRAME

\* X\_FRAME

\* V\_FRAME

\*/

//#define CH7\_OPTION CH7\_SAVE\_WP

/\*

\* CH7\_DO\_NOTHING

\* CH7\_FLIP

\* CH7\_SIMPLE\_MODE

\* CH7\_RTL

\* CH7\_SAVE\_TRIM

\* CH7\_SAVE\_WP

\* CH7\_CAMERA\_TRIGGER

\*/

// Inertia based contollers

//#define INERTIAL\_NAV\_XY ENABLED

#define INERTIAL\_NAV\_Z ENABLED

//#define MOTORS\_JD880

//#define MOTORS\_JD850

// agmatthews USERHOOKS

// the choice of function names is up to the user and does not have to match these

// uncomment these hooks and ensure there is a matching function on your "UserCode.pde" file

//#define USERHOOK\_FASTLOOP userhook\_FastLoop();

#define USERHOOK\_50HZLOOP userhook\_50Hz();

//#define USERHOOK\_MEDIUMLOOP userhook\_MediumLoop();

//#define USERHOOK\_SLOWLOOP userhook\_SlowLoop();

//#define USERHOOK\_SUPERSLOWLOOP userhook\_SuperSlowLoop();

#define USERHOOK\_INIT userhook\_init();

// the choice of included variables file (\*.h) is up to the user and does not have to match this one

// Ensure the defined file exists and is in the arducopter directory

#define USERHOOK\_VARIABLES "UserVariables.h"

//#define LOGGING\_ENABLED DISABLED

// #define LOITER\_REPOSITIONING ENABLED // Experimental Do Not Use

// #define LOITER\_RP ROLL\_PITCH\_LOITER\_PR

## APM\_Config\_mavlink\_hil.h

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

// Example config file. Take a look at config.h. Any term define there can be overridden by defining it here.

//#define CONFIG\_APM\_HARDWARE APM\_HARDWARE\_APM2

// Ordinary users should please ignore the following define.

// APM2\_BETA\_HARDWARE is used to support early (September-October 2011) APM2

// hardware which had the BMP085 barometer onboard. Only a handful of

// developers have these boards.

//#define APM2\_BETA\_HARDWARE

//#define MAG\_ORIENTATION AP\_COMPASS\_COMPONENTS\_DOWN\_PINS\_FORWARD

//#define HIL\_MODE HIL\_MODE\_ATTITUDE

//#define DMP\_ENABLED ENABLED

//#define SECONDARY\_DMP\_ENABLED ENABLED // allows running DMP in parallel with DCM for testing purposes

//#define FRAME\_CONFIG QUAD\_FRAME

/\*

\* options:

\* QUAD\_FRAME

\* TRI\_FRAME

\* HEXA\_FRAME

\* Y6\_FRAME

\* OCTA\_FRAME

\* OCTA\_QUAD\_FRAME

\* HELI\_FRAME

\*/

//#define FRAME\_ORIENTATION X\_FRAME

/\*

\* PLUS\_FRAME

\* X\_FRAME

\* V\_FRAME

\*/

//#define CH7\_OPTION CH7\_SAVE\_WP

/\*

\* CH7\_DO\_NOTHING

\* CH7\_FLIP

\* CH7\_SIMPLE\_MODE

\* CH7\_RTL

\* CH7\_SAVE\_TRIM

\* CH7\_SAVE\_WP

\* CH7\_CAMERA\_TRIGGER

\*/

// Inertia based contollers

//#define INERTIAL\_NAV\_XY ENABLED

#define INERTIAL\_NAV\_Z ENABLED

//#define MOTORS\_JD880

//#define MOTORS\_JD850

// agmatthews USERHOOKS

// the choice of function names is up to the user and does not have to match these

// uncomment these hooks and ensure there is a matching function on your "UserCode.pde" file

//#define USERHOOK\_FASTLOOP userhook\_FastLoop();

#define USERHOOK\_50HZLOOP userhook\_50Hz();

//#define USERHOOK\_MEDIUMLOOP userhook\_MediumLoop();

//#define USERHOOK\_SLOWLOOP userhook\_SlowLoop();

//#define USERHOOK\_SUPERSLOWLOOP userhook\_SuperSlowLoop();

#define USERHOOK\_INIT userhook\_init();

// the choice of included variables file (\*.h) is up to the user and does not have to match this one

// Ensure the defined file exists and is in the arducopter directory

#define USERHOOK\_VARIABLES "UserVariables.h"

//#define LOGGING\_ENABLED DISABLED

// #define LOITER\_REPOSITIONING ENABLED // Experimental Do Not Use

// #define LOITER\_RP ROLL\_PITCH\_LOITER\_PR

## AP\_State

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

void set\_home\_is\_set(bool b)

{

ap.home\_is\_set = b;

if(b) Log\_Write\_Event(DATA\_SET\_HOME);

}

// ---------------------------------------------

void set\_armed(bool b)

{

ap.armed = b;

if(b){

Log\_Write\_Event(DATA\_ARMED);

}else{

Log\_Write\_Event(DATA\_DISARMED);

}

}

// ---------------------------------------------

void set\_auto\_armed(bool b)

{

ap.auto\_armed = b;

if(b){

Log\_Write\_Event(DATA\_AUTO\_ARMED);

}

}

// ---------------------------------------------

void set\_simple\_mode(bool b)

{

if(ap.simple\_mode != b){

if(b){

Log\_Write\_Event(DATA\_SET\_SIMPLE\_ON);

}else{

Log\_Write\_Event(DATA\_SET\_SIMPLE\_OFF);

}

}

ap.simple\_mode = b;

}

// ---------------------------------------------

static void set\_failsafe(bool mode)

{

// only act on changes

// -------------------

if(ap.failsafe != mode) {

// store the value so we don't trip the gate twice

// -----------------------------------------------

ap.failsafe = mode;

if (ap.failsafe == false) {

// We've regained radio contact

// ----------------------------

failsafe\_off\_event();

}else{

// We've lost radio contact

// ------------------------

failsafe\_on\_event();

}

}

}

// ---------------------------------------------

void set\_low\_battery(bool b)

{

ap.low\_battery = b;

}

// ---------------------------------------------

void set\_takeoff\_complete(bool b)

{

if(b){

Log\_Write\_Event(DATA\_TAKEOFF);

}

ap.takeoff\_complete = b;

}

// ---------------------------------------------

void set\_land\_complete(bool b)

{

if(b){

Log\_Write\_Event(DATA\_LAND\_COMPLETE);

}

ap.land\_complete = b;

}

// ---------------------------------------------

void set\_alt\_change(uint8\_t flag){

// if no change, exit immediately

if( alt\_change\_flag == flag ) {

return;

}

// update flag

alt\_change\_flag = flag;

if(flag == REACHED\_ALT){

Log\_Write\_Event(DATA\_REACHED\_ALT);

}else if(flag == ASCENDING){

Log\_Write\_Event(DATA\_ASCENDING);

}else if(flag == DESCENDING){

Log\_Write\_Event(DATA\_DESCENDING);

}

}

void set\_compass\_healthy(bool b)

{

if(ap.compass\_status != b){

if(false == b){

Log\_Write\_Event(DATA\_LOST\_COMPASS);

}

}

ap.compass\_status = b;

}

void set\_gps\_healthy(bool b)

{

if(ap.gps\_status != b){

if(false == b){

Log\_Write\_Event(DATA\_LOST\_GPS);

}

}

ap.gps\_status = b;

}

void dump\_state()

{

cliSerial->printf("st: %u\n",ap.value);

//cliSerial->printf("%u\n", \*(uint16\_t\*)&ap);

}

## Attitude

/// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

static void

get\_stabilize\_roll(int32\_t target\_angle)

{

// angle error

target\_angle = wrap\_180(target\_angle - ahrs.roll\_sensor);

// limit the error we're feeding to the PID

target\_angle = constrain(target\_angle, -4500, 4500);

// convert to desired Rate:

int32\_t target\_rate = g.pi\_stabilize\_roll.get\_p(target\_angle);

int16\_t i\_stab;

if(labs(ahrs.roll\_sensor) < 500) {

target\_angle = constrain(target\_angle, -500, 500);

i\_stab = g.pi\_stabilize\_roll.get\_i(target\_angle, G\_Dt);

}else{

i\_stab = g.pi\_stabilize\_roll.get\_integrator();

}

// set targets for rate controller

set\_roll\_rate\_target(target\_rate+i\_stab, EARTH\_FRAME);

}

static void

get\_stabilize\_pitch(int32\_t target\_angle)

{

// angle error

target\_angle = wrap\_180(target\_angle - ahrs.pitch\_sensor);

// limit the error we're feeding to the PID

target\_angle = constrain(target\_angle, -4500, 4500);

// convert to desired Rate:

int32\_t target\_rate = g.pi\_stabilize\_pitch.get\_p(target\_angle);

int16\_t i\_stab;

if(labs(ahrs.pitch\_sensor) < 500) {

target\_angle = constrain(target\_angle, -500, 500);

i\_stab = g.pi\_stabilize\_pitch.get\_i(target\_angle, G\_Dt);

}else{

i\_stab = g.pi\_stabilize\_pitch.get\_integrator();

}

// set targets for rate controller

set\_pitch\_rate\_target(target\_rate + i\_stab, EARTH\_FRAME);

}

static void

get\_stabilize\_yaw(int32\_t target\_angle)

{

int32\_t target\_rate,i\_term;

int32\_t angle\_error;

int32\_t output = 0;

// angle error

angle\_error = wrap\_180(target\_angle - ahrs.yaw\_sensor);

// limit the error we're feeding to the PID

angle\_error = constrain(angle\_error, -4500, 4500);

// convert angle error to desired Rate:

target\_rate = g.pi\_stabilize\_yaw.get\_p(angle\_error);

i\_term = g.pi\_stabilize\_yaw.get\_i(angle\_error, G\_Dt);

// do not use rate controllers for helicotpers with external gyros

#if FRAME\_CONFIG == HELI\_FRAME

if(motors.ext\_gyro\_enabled) {

g.rc\_4.servo\_out = constrain((target\_rate + i\_term), -4500, 4500);

}

#endif

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && g.radio\_tuning == CH6\_YAW\_KP ) {

pid\_log\_counter++;

if( pid\_log\_counter >= 10 ) { // (update rate / desired output rate) = (100hz / 10hz) = 10

pid\_log\_counter = 0;

Log\_Write\_PID(CH6\_YAW\_KP, angle\_error, target\_rate, i\_term, 0, output, tuning\_value);

}

}

#endif

// set targets for rate controller

set\_yaw\_rate\_target(target\_rate+i\_term, EARTH\_FRAME);

}

static void

get\_acro\_roll(int32\_t target\_rate)

{

target\_rate = target\_rate \* g.acro\_p;

// set targets for rate controller

set\_roll\_rate\_target(target\_rate, BODY\_FRAME);

}

static void

get\_acro\_pitch(int32\_t target\_rate)

{

target\_rate = target\_rate \* g.acro\_p;

// set targets for rate controller

set\_pitch\_rate\_target(target\_rate, BODY\_FRAME);

}

static void

get\_acro\_yaw(int32\_t target\_rate)

{

target\_rate = target\_rate \* g.acro\_p;

// set targets for rate controller

set\_yaw\_rate\_target(target\_rate, BODY\_FRAME);

}

// Roll with rate input and stabilized in the earth frame

static void

get\_roll\_rate\_stabilized\_ef(int32\_t stick\_angle)

{

int32\_t angle\_error = 0;

// convert the input to the desired roll rate

int32\_t target\_rate = stick\_angle \* g.acro\_p - (roll\_axis \* g.acro\_balance\_roll)/100;

// convert the input to the desired roll rate

roll\_axis += target\_rate \* G\_Dt;

roll\_axis = wrap\_180(roll\_axis);

// ensure that we don't reach gimbal lock

if (labs(roll\_axis > 4500) && g.acro\_trainer\_enabled) {

roll\_axis = constrain(roll\_axis, -4500, 4500);

angle\_error = wrap\_180(roll\_axis - ahrs.roll\_sensor);

} else {

// angle error with maximum of +- max\_angle\_overshoot

angle\_error = wrap\_180(roll\_axis - ahrs.roll\_sensor);

angle\_error = constrain(angle\_error, -MAX\_ROLL\_OVERSHOOT, MAX\_ROLL\_OVERSHOOT);

}

if (motors.armed() == false || ((g.rc\_3.control\_in == 0) && !ap.failsafe)) {

angle\_error = 0;

}

// update roll\_axis to be within max\_angle\_overshoot of our current heading

roll\_axis = wrap\_180(angle\_error + ahrs.roll\_sensor);

// set earth frame targets for rate controller

// set earth frame targets for rate controller

set\_roll\_rate\_target(g.pi\_stabilize\_roll.get\_p(angle\_error) + target\_rate, EARTH\_FRAME);

}

// Pitch with rate input and stabilized in the earth frame

static void

get\_pitch\_rate\_stabilized\_ef(int32\_t stick\_angle)

{

int32\_t angle\_error = 0;

// convert the input to the desired pitch rate

int32\_t target\_rate = stick\_angle \* g.acro\_p - (pitch\_axis \* g.acro\_balance\_pitch)/100;

// convert the input to the desired pitch rate

pitch\_axis += target\_rate \* G\_Dt;

pitch\_axis = wrap\_180(pitch\_axis);

// ensure that we don't reach gimbal lock

if (labs(pitch\_axis) > 4500) {

pitch\_axis = constrain(pitch\_axis, -4500, 4500);

angle\_error = wrap\_180(pitch\_axis - ahrs.pitch\_sensor);

} else {

// angle error with maximum of +- max\_angle\_overshoot

angle\_error = wrap\_180(pitch\_axis - ahrs.pitch\_sensor);

angle\_error = constrain(angle\_error, -MAX\_PITCH\_OVERSHOOT, MAX\_PITCH\_OVERSHOOT);

}

if (motors.armed() == false || ((g.rc\_3.control\_in == 0) && !ap.failsafe)) {

angle\_error = 0;

}

// update pitch\_axis to be within max\_angle\_overshoot of our current heading

pitch\_axis = wrap\_180(angle\_error + ahrs.pitch\_sensor);

// set earth frame targets for rate controller

set\_pitch\_rate\_target(g.pi\_stabilize\_pitch.get\_p(angle\_error) + target\_rate, EARTH\_FRAME);

}

// Yaw with rate input and stabilized in the earth frame

static void

get\_yaw\_rate\_stabilized\_ef(int32\_t stick\_angle)

{

int32\_t angle\_error = 0;

// convert the input to the desired yaw rate

int32\_t target\_rate = stick\_angle \* g.acro\_p;

// convert the input to the desired yaw rate

nav\_yaw += target\_rate \* G\_Dt;

nav\_yaw = wrap\_360(nav\_yaw);

// calculate difference between desired heading and current heading

angle\_error = wrap\_180(nav\_yaw - ahrs.yaw\_sensor);

// limit the maximum overshoot

angle\_error = constrain(angle\_error, -MAX\_YAW\_OVERSHOOT, MAX\_YAW\_OVERSHOOT);

if (motors.armed() == false || ((g.rc\_3.control\_in == 0) && !ap.failsafe)) {

angle\_error = 0;

}

// update nav\_yaw to be within max\_angle\_overshoot of our current heading

nav\_yaw = wrap\_360(angle\_error + ahrs.yaw\_sensor);

// set earth frame targets for rate controller

set\_yaw\_rate\_target(g.pi\_stabilize\_yaw.get\_p(angle\_error)+target\_rate, EARTH\_FRAME);

}

// set\_roll\_rate\_target - to be called by upper controllers to set roll rate targets in the earth frame

void set\_roll\_rate\_target( int32\_t desired\_rate, uint8\_t earth\_or\_body\_frame ) {

rate\_targets\_frame = earth\_or\_body\_frame;

if( earth\_or\_body\_frame == BODY\_FRAME ) {

roll\_rate\_target\_bf = desired\_rate;

}else{

roll\_rate\_target\_ef = desired\_rate;

}

}

// set\_pitch\_rate\_target - to be called by upper controllers to set pitch rate targets in the earth frame

void set\_pitch\_rate\_target( int32\_t desired\_rate, uint8\_t earth\_or\_body\_frame ) {

rate\_targets\_frame = earth\_or\_body\_frame;

if( earth\_or\_body\_frame == BODY\_FRAME ) {

pitch\_rate\_target\_bf = desired\_rate;

}else{

pitch\_rate\_target\_ef = desired\_rate;

}

}

// set\_yaw\_rate\_target - to be called by upper controllers to set yaw rate targets in the earth frame

void set\_yaw\_rate\_target( int32\_t desired\_rate, uint8\_t earth\_or\_body\_frame ) {

rate\_targets\_frame = earth\_or\_body\_frame;

if( earth\_or\_body\_frame == BODY\_FRAME ) {

yaw\_rate\_target\_bf = desired\_rate;

}else{

yaw\_rate\_target\_ef = desired\_rate;

}

}

// update\_rate\_contoller\_targets - converts earth frame rates to body frame rates for rate controllers

void

update\_rate\_contoller\_targets()

{

if( rate\_targets\_frame == EARTH\_FRAME ) {

// convert earth frame rates to body frame rates

roll\_rate\_target\_bf = roll\_rate\_target\_ef - sin\_pitch \* yaw\_rate\_target\_ef;

pitch\_rate\_target\_bf = cos\_roll\_x \* pitch\_rate\_target\_ef + sin\_roll \* cos\_pitch\_x \* yaw\_rate\_target\_ef;

yaw\_rate\_target\_bf = cos\_pitch\_x \* cos\_roll\_x \* yaw\_rate\_target\_ef - sin\_roll \* pitch\_rate\_target\_ef;

}

}

// run roll, pitch and yaw rate controllers and send output to motors

// targets for these controllers comes from stabilize controllers

void

run\_rate\_controllers()

{

#if FRAME\_CONFIG == HELI\_FRAME // helicopters only use rate controllers for yaw and only when not using an external gyro

if(!motors.ext\_gyro\_enabled) {

g.rc\_1.servo\_out = get\_heli\_rate\_roll(roll\_rate\_target\_bf);

g.rc\_2.servo\_out = get\_heli\_rate\_pitch(pitch\_rate\_target\_bf);

g.rc\_4.servo\_out = get\_heli\_rate\_yaw(yaw\_rate\_target\_bf);

}

#else

// call rate controllers

g.rc\_1.servo\_out = get\_rate\_roll(roll\_rate\_target\_bf);

g.rc\_2.servo\_out = get\_rate\_pitch(pitch\_rate\_target\_bf);

g.rc\_4.servo\_out = get\_rate\_yaw(yaw\_rate\_target\_bf);

#endif

// run throttle controller if accel based throttle controller is enabled and active (active means it has been given a target)

if( g.throttle\_accel\_enabled && throttle\_accel\_controller\_active ) {

set\_throttle\_out(get\_throttle\_accel(throttle\_accel\_target\_ef), true);

}

}

#if FRAME\_CONFIG == HELI\_FRAME

// init\_rate\_controllers - set-up filters for rate controller inputs

void init\_rate\_controllers()

{

// initalise low pass filters on rate controller inputs

// 1st parameter is time\_step, 2nd parameter is time\_constant

rate\_roll\_filter.set\_cutoff\_frequency(0.01, 2.0);

rate\_pitch\_filter.set\_cutoff\_frequency(0.01, 2.0);

// rate\_yaw\_filter.set\_cutoff\_frequency(0.01, 2.0);

// other option for initialisation is rate\_roll\_filter.set\_cutoff\_frequency(<time\_step>,<cutoff\_freq>);

}

static int16\_t

get\_heli\_rate\_roll(int32\_t target\_rate)

{

int32\_t p,i,d,ff; // used to capture pid values for logging

int32\_t current\_rate; // this iteration's rate

int32\_t rate\_error; // simply target\_rate - current\_rate

int32\_t output; // output from pid controller

// get current rate

current\_rate = (omega.x \* DEGX100);

// filter input

current\_rate = rate\_roll\_filter.apply(current\_rate);

// call pid controller

rate\_error = target\_rate - current\_rate;

p = g.pid\_rate\_roll.get\_p(rate\_error);

if (motors.flybar\_mode == 1) { // Mechanical Flybars get regular integral for rate auto trim

if (target\_rate > -50 && target\_rate < 50){ // Frozen at high rates

i = g.pid\_rate\_roll.get\_i(rate\_error, G\_Dt);

} else {

i = g.pid\_rate\_roll.get\_integrator();

}

} else {

i = g.pid\_rate\_roll.get\_leaky\_i(rate\_error, G\_Dt, RATE\_INTEGRATOR\_LEAK\_RATE); // Flybarless Helis get huge I-terms. I-term controls much of the rate

}

d = g.pid\_rate\_roll.get\_d(rate\_error, G\_Dt);

ff = g.heli\_roll\_ff \* target\_rate;

output = p + i + d + ff;

// constrain output

output = constrain(output, -4500, 4500);

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the rate P, I or D gains

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_RATE\_KP || g.radio\_tuning == CH6\_RATE\_KI || g.radio\_tuning == CH6\_RATE\_KD) ) {

pid\_log\_counter++;

if( pid\_log\_counter >= 10 ) { // (update rate / desired output rate) = (100hz / 10hz) = 10

pid\_log\_counter = 0;

Log\_Write\_PID(CH6\_RATE\_KP, rate\_error, p, i, d, output, tuning\_value);

}

}

#endif

// output control

return output;

}

static int16\_t

get\_heli\_rate\_pitch(int32\_t target\_rate)

{

int32\_t p,i,d,ff; // used to capture pid values for logging

int32\_t current\_rate; // this iteration's rate

int32\_t rate\_error; // simply target\_rate - current\_rate

int32\_t output; // output from pid controller

// get current rate

current\_rate = (omega.y \* DEGX100);

// filter input

current\_rate = rate\_pitch\_filter.apply(current\_rate);

// call pid controller

rate\_error = target\_rate - current\_rate;

p = g.pid\_rate\_pitch.get\_p(rate\_error); // Helicopters get huge feed-forward

if (motors.flybar\_mode == 1) { // Mechanical Flybars get regular integral for rate auto trim

if (target\_rate > -50 && target\_rate < 50){ // Frozen at high rates

i = g.pid\_rate\_pitch.get\_i(rate\_error, G\_Dt);

} else {

i = g.pid\_rate\_pitch.get\_integrator();

}

} else {

i = g.pid\_rate\_pitch.get\_leaky\_i(rate\_error, G\_Dt, RATE\_INTEGRATOR\_LEAK\_RATE); // Flybarless Helis get huge I-terms. I-term controls much of the rate

}

d = g.pid\_rate\_pitch.get\_d(rate\_error, G\_Dt);

ff = g.heli\_pitch\_ff\*target\_rate;

output = p + i + d + ff;

// constrain output

output = constrain(output, -4500, 4500);

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the rate P, I or D gains

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_RATE\_KP || g.radio\_tuning == CH6\_RATE\_KI || g.radio\_tuning == CH6\_RATE\_KD) ) {

if( pid\_log\_counter == 0 ) { // relies on get\_heli\_rate\_roll to update pid\_log\_counter

Log\_Write\_PID(CH6\_RATE\_KP+100, rate\_error, p, i, 0, output, tuning\_value);

}

}

#endif

// output control

return output;

}

static int16\_t

get\_heli\_rate\_yaw(int32\_t target\_rate)

{

int32\_t p,i,d,ff; // used to capture pid values for logging

int32\_t current\_rate; // this iteration's rate

int32\_t rate\_error;

int32\_t output;

// get current rate

current\_rate = (omega.z \* DEGX100);

// filter input

// current\_rate = rate\_yaw\_filter.apply(current\_rate);

// rate control

rate\_error = target\_rate - current\_rate;

// separately calculate p, i, d values for logging

p = g.pid\_rate\_yaw.get\_p(rate\_error);

i = g.pid\_rate\_yaw.get\_i(rate\_error, G\_Dt);

d = g.pid\_rate\_yaw.get\_d(rate\_error, G\_Dt);

ff = g.heli\_yaw\_ff\*target\_rate;

output = p + i + d + ff;

output = constrain(output, -4500, 4500);

#if LOGGING\_ENABLED == ENABLED

// log output if PID loggins is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_YAW\_RATE\_KP || g.radio\_tuning == CH6\_YAW\_RATE\_KD) ) {

pid\_log\_counter++;

if( pid\_log\_counter >= 10 ) { // (update rate / desired output rate) = (100hz / 10hz) = 10

pid\_log\_counter = 0;

Log\_Write\_PID(CH6\_YAW\_RATE\_KP, rate\_error, p, i, d, output, tuning\_value);

}

}

#endif

// output control

return output;

}

#endif // HELI\_FRAME

#if FRAME\_CONFIG != HELI\_FRAME

static int16\_t

get\_rate\_roll(int32\_t target\_rate)

{

int32\_t p,i,d; // used to capture pid values for logging

int32\_t current\_rate; // this iteration's rate

int32\_t rate\_error; // simply target\_rate - current\_rate

int32\_t output; // output from pid controller

// get current rate

current\_rate = (omega.x \* DEGX100);

// call pid controller

rate\_error = target\_rate - current\_rate;

p = g.pid\_rate\_roll.get\_p(rate\_error);

// freeze I term if we've breached roll-pitch limits

if( motors.reached\_limit(AP\_MOTOR\_ROLLPITCH\_LIMIT) ) {

i = g.pid\_rate\_roll.get\_integrator();

}else{

i = g.pid\_rate\_roll.get\_i(rate\_error, G\_Dt);

}

d = g.pid\_rate\_roll.get\_d(rate\_error, G\_Dt);

output = p + i + d;

// constrain output

output = constrain(output, -5000, 5000);

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the rate P, I or D gains

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_RATE\_KP || g.radio\_tuning == CH6\_RATE\_KI || g.radio\_tuning == CH6\_RATE\_KD) ) {

pid\_log\_counter++; // Note: get\_rate\_pitch pid logging relies on this function to update pid\_log\_counter so if you change the line above you must change the equivalent line in get\_rate\_pitch

if( pid\_log\_counter >= 10 ) { // (update rate / desired output rate) = (100hz / 10hz) = 10

pid\_log\_counter = 0;

Log\_Write\_PID(CH6\_RATE\_KP, rate\_error, p, i, d, output, tuning\_value);

}

}

#endif

// output control

return output;

}

static int16\_t

get\_rate\_pitch(int32\_t target\_rate)

{

int32\_t p,i,d; // used to capture pid values for logging

int32\_t current\_rate; // this iteration's rate

int32\_t rate\_error; // simply target\_rate - current\_rate

int32\_t output; // output from pid controller

// get current rate

current\_rate = (omega.y \* DEGX100);

// call pid controller

rate\_error = target\_rate - current\_rate;

p = g.pid\_rate\_pitch.get\_p(rate\_error);

// freeze I term if we've breached roll-pitch limits

if( motors.reached\_limit(AP\_MOTOR\_ROLLPITCH\_LIMIT) ) {

i = g.pid\_rate\_pitch.get\_integrator();

}else{

i = g.pid\_rate\_pitch.get\_i(rate\_error, G\_Dt);

}

d = g.pid\_rate\_pitch.get\_d(rate\_error, G\_Dt);

output = p + i + d;

// constrain output

output = constrain(output, -5000, 5000);

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the rate P, I or D gains

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_RATE\_KP || g.radio\_tuning == CH6\_RATE\_KI || g.radio\_tuning == CH6\_RATE\_KD) ) {

if( pid\_log\_counter == 0 ) { // relies on get\_rate\_roll having updated pid\_log\_counter

Log\_Write\_PID(CH6\_RATE\_KP+100, rate\_error, p, i, d, output, tuning\_value);

}

}

#endif

// output control

return output;

}

static int16\_t

get\_rate\_yaw(int32\_t target\_rate)

{

int32\_t p,i,d; // used to capture pid values for logging

int32\_t rate\_error;

int32\_t output;

// rate control

rate\_error = target\_rate - (omega.z \* DEGX100);

// separately calculate p, i, d values for logging

p = g.pid\_rate\_yaw.get\_p(rate\_error);

// freeze I term if we've breached yaw limits

if( motors.reached\_limit(AP\_MOTOR\_YAW\_LIMIT) ) {

i = g.pid\_rate\_yaw.get\_integrator();

}else{

i = g.pid\_rate\_yaw.get\_i(rate\_error, G\_Dt);

}

d = g.pid\_rate\_yaw.get\_d(rate\_error, G\_Dt);

output = p+i+d;

output = constrain(output, -4500, 4500);

#if LOGGING\_ENABLED == ENABLED

// log output if PID loggins is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && g.radio\_tuning == CH6\_YAW\_RATE\_KP ) {

pid\_log\_counter++;

if( pid\_log\_counter >= 10 ) { // (update rate / desired output rate) = (100hz / 10hz) = 10

pid\_log\_counter = 0;

Log\_Write\_PID(CH6\_YAW\_RATE\_KP, rate\_error, p, i, d, output, tuning\_value);

}

}

#endif

#if FRAME\_CONFIG == TRI\_FRAME

// constrain output

return output;

#else // !TRI\_FRAME

// output control:

int16\_t yaw\_limit = 2200 + abs(g.rc\_4.control\_in);

// smoother Yaw control:

return constrain(output, -yaw\_limit, yaw\_limit);

#endif // TRI\_FRAME

}

#endif // !HELI\_FRAME

// calculate modified roll/pitch depending upon optical flow calculated position

static int32\_t

get\_of\_roll(int32\_t input\_roll)

{

#if OPTFLOW == ENABLED

static float tot\_x\_cm = 0; // total distance from target

static uint32\_t last\_of\_roll\_update = 0;

int32\_t new\_roll = 0;

int32\_t p,i,d;

// check if new optflow data available

if( optflow.last\_update != last\_of\_roll\_update) {

last\_of\_roll\_update = optflow.last\_update;

// add new distance moved

tot\_x\_cm += optflow.x\_cm;

// only stop roll if caller isn't modifying roll

if( input\_roll == 0 && current\_loc.alt < 1500) {

p = g.pid\_optflow\_roll.get\_p(-tot\_x\_cm);

i = g.pid\_optflow\_roll.get\_i(-tot\_x\_cm,1.0); // we could use the last update time to calculate the time change

d = g.pid\_optflow\_roll.get\_d(-tot\_x\_cm,1.0);

new\_roll = p+i+d;

}else{

g.pid\_optflow\_roll.reset\_I();

tot\_x\_cm = 0;

p = 0; // for logging

i = 0;

d = 0;

}

// limit amount of change and maximum angle

of\_roll = constrain(new\_roll, (of\_roll-20), (of\_roll+20));

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the rate P, I or D gains

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_OPTFLOW\_KP || g.radio\_tuning == CH6\_OPTFLOW\_KI || g.radio\_tuning == CH6\_OPTFLOW\_KD) ) {

pid\_log\_counter++; // Note: get\_of\_pitch pid logging relies on this function updating pid\_log\_counter so if you change the line above you must change the equivalent line in get\_of\_pitch

if( pid\_log\_counter >= 5 ) { // (update rate / desired output rate) = (100hz / 10hz) = 10

pid\_log\_counter = 0;

Log\_Write\_PID(CH6\_OPTFLOW\_KP, tot\_x\_cm, p, i, d, of\_roll, tuning\_value);

}

}

#endif // LOGGING\_ENABLED == ENABLED

}

// limit max angle

of\_roll = constrain(of\_roll, -1000, 1000);

return input\_roll+of\_roll;

#else

return input\_roll;

#endif

}

static int32\_t

get\_of\_pitch(int32\_t input\_pitch)

{

#if OPTFLOW == ENABLED

static float tot\_y\_cm = 0; // total distance from target

static uint32\_t last\_of\_pitch\_update = 0;

int32\_t new\_pitch = 0;

int32\_t p,i,d;

// check if new optflow data available

if( optflow.last\_update != last\_of\_pitch\_update ) {

last\_of\_pitch\_update = optflow.last\_update;

// add new distance moved

tot\_y\_cm += optflow.y\_cm;

// only stop roll if caller isn't modifying pitch

if( input\_pitch == 0 && current\_loc.alt < 1500 ) {

p = g.pid\_optflow\_pitch.get\_p(tot\_y\_cm);

i = g.pid\_optflow\_pitch.get\_i(tot\_y\_cm, 1.0); // we could use the last update time to calculate the time change

d = g.pid\_optflow\_pitch.get\_d(tot\_y\_cm, 1.0);

new\_pitch = p + i + d;

}else{

tot\_y\_cm = 0;

g.pid\_optflow\_pitch.reset\_I();

p = 0; // for logging

i = 0;

d = 0;

}

// limit amount of change

of\_pitch = constrain(new\_pitch, (of\_pitch-20), (of\_pitch+20));

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the rate P, I or D gains

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_OPTFLOW\_KP || g.radio\_tuning == CH6\_OPTFLOW\_KI || g.radio\_tuning == CH6\_OPTFLOW\_KD) ) {

if( pid\_log\_counter == 0 ) { // relies on get\_of\_roll having updated the pid\_log\_counter

Log\_Write\_PID(CH6\_OPTFLOW\_KP+100, tot\_y\_cm, p, i, d, of\_pitch, tuning\_value);

}

}

#endif // LOGGING\_ENABLED == ENABLED

}

// limit max angle

of\_pitch = constrain(of\_pitch, -1000, 1000);

return input\_pitch+of\_pitch;

#else

return input\_pitch;

#endif

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* yaw controllers

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void get\_look\_ahead\_yaw(int16\_t pilot\_yaw)

{

// Commanded Yaw to automatically look ahead.

if (g\_gps->fix && g\_gps->ground\_course > YAW\_LOOK\_AHEAD\_MIN\_SPEED) {

nav\_yaw = get\_yaw\_slew(nav\_yaw, g\_gps->ground\_course, AUTO\_YAW\_SLEW\_RATE);

get\_stabilize\_yaw(wrap\_360(nav\_yaw + pilot\_yaw)); // Allow pilot to "skid" around corners up to 45 degrees

}else{

nav\_yaw += pilot\_yaw \* g.acro\_p \* G\_Dt;

nav\_yaw = wrap\_360(nav\_yaw);

get\_stabilize\_yaw(nav\_yaw);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* throttle control

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// update\_throttle\_cruise - update throttle cruise if necessary

static void update\_throttle\_cruise(int16\_t throttle)

{

// ensure throttle\_avg has been initialised

if( throttle\_avg == 0 ) {

throttle\_avg = g.throttle\_cruise;

}

// calc average throttle if we are in a level hover

if (throttle > g.throttle\_min && abs(climb\_rate) < 60 && labs(ahrs.roll\_sensor) < 500 && labs(ahrs.pitch\_sensor) < 500) {

throttle\_avg = throttle\_avg \* .99 + (float)throttle \* .01;

g.throttle\_cruise = throttle\_avg;

}

}

#if FRAME\_CONFIG == HELI\_FRAME

// get\_angle\_boost - returns a throttle including compensation for roll/pitch angle

// throttle value should be 0 ~ 1000

// for traditional helicopters

static int16\_t get\_angle\_boost(int16\_t throttle)

{

float angle\_boost\_factor = cos\_pitch\_x \* cos\_roll\_x;

angle\_boost\_factor = 1.0 - constrain(angle\_boost\_factor, .5, 1.0);

int16\_t throttle\_above\_mid = max(throttle - motors.throttle\_mid,0);

// to allow logging of angle boost

angle\_boost = throttle\_above\_mid\*angle\_boost\_factor;

return throttle + angle\_boost;

}

#else // all multicopters

// get\_angle\_boost - returns a throttle including compensation for roll/pitch angle

// throttle value should be 0 ~ 1000

static int16\_t get\_angle\_boost(int16\_t throttle)

{

float temp = cos\_pitch\_x \* cos\_roll\_x;

int16\_t throttle\_out;

temp = constrain(temp, .5, 1.0);

temp = constrain(9000-max(labs(roll\_axis),labs(pitch\_axis)), 0, 3000) / (3000 \* temp);

throttle\_out = constrain((float)(throttle-g.throttle\_min) \* temp + g.throttle\_min, g.throttle\_min, 1000);

//Serial.printf("Thin:%4.2f sincos:%4.2f temp:%4.2f roll\_axis:%4.2f Out:%4.2f \n", 1.0\*throttle, 1.0\*cos\_pitch\_x \* cos\_roll\_x, 1.0\*temp, 1.0\*roll\_axis, 1.0\*constrain((float)value \* temp, 0, 1000));

// to allow logging of angle boost

angle\_boost = throttle\_out - throttle;

return throttle\_out;

}

#endif // FRAME\_CONFIG == HELI\_FRAME

// set\_throttle\_out - to be called by upper throttle controllers when they wish to provide throttle output directly to motors

// provide 0 to cut motors

void set\_throttle\_out( int16\_t throttle\_out, bool apply\_angle\_boost )

{

if( apply\_angle\_boost ) {

g.rc\_3.servo\_out = get\_angle\_boost(throttle\_out);

}else{

g.rc\_3.servo\_out = throttle\_out;

// clear angle\_boost for logging purposes

angle\_boost = 0;

}

}

// set\_throttle\_accel\_target - to be called by upper throttle controllers to set desired vertical acceleration in earth frame

void set\_throttle\_accel\_target( int16\_t desired\_acceleration )

{

if( g.throttle\_accel\_enabled ) {

throttle\_accel\_target\_ef = desired\_acceleration;

throttle\_accel\_controller\_active = true;

}else{

// To-Do log dataflash or tlog error

cliSerial->print\_P(PSTR("Err: target sent to inactive acc thr controller!\n"));

}

}

// disable\_throttle\_accel - disables the accel based throttle controller

// it will be re-enasbled on the next set\_throttle\_accel\_target

// required when we wish to set motors to zero when pilot inputs zero throttle

void throttle\_accel\_deactivate()

{

throttle\_accel\_controller\_active = false;

}

// get\_throttle\_accel - accelerometer based throttle controller

// returns an actual throttle output (0 ~ 1000) to be sent to the motors

static int16\_t

get\_throttle\_accel(int16\_t z\_target\_accel)

{

static float z\_accel\_error = 0; // The acceleration error in cm.

static uint32\_t last\_call\_ms = 0; // the last time this controller was called

int32\_t p,i,d; // used to capture pid values for logging

int16\_t output;

float z\_accel\_meas;

uint32\_t now = millis();

// Calculate Earth Frame Z acceleration

z\_accel\_meas = -(ahrs.get\_accel\_ef().z + gravity) \* 100;

// reset target altitude if this controller has just been engaged

if( now - last\_call\_ms > 100 ) {

// Reset Filter

z\_accel\_error = 0;

} else {

// calculate accel error and Filter with fc = 2 Hz

z\_accel\_error = z\_accel\_error + 0.11164 \* (constrain(z\_target\_accel - z\_accel\_meas, -32000, 32000) - z\_accel\_error);

}

last\_call\_ms = now;

// separately calculate p, i, d values for logging

p = g.pid\_throttle\_accel.get\_p(z\_accel\_error);

// freeze I term if we've breached throttle limits

if( motors.reached\_limit(AP\_MOTOR\_THROTTLE\_LIMIT) ) {

i = g.pid\_throttle\_accel.get\_integrator();

}else{

i = g.pid\_throttle\_accel.get\_i(z\_accel\_error, .01);

}

d = g.pid\_throttle\_accel.get\_d(z\_accel\_error, .01);

//

// limit the rate

output = constrain(p+i+d+g.throttle\_cruise, g.throttle\_min, g.throttle\_max);

#if LOGGING\_ENABLED == ENABLED

// log output if PID loggins is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_THR\_ACCEL\_KP || g.radio\_tuning == CH6\_THR\_ACCEL\_KI || g.radio\_tuning == CH6\_THR\_ACCEL\_KD) ) {

pid\_log\_counter++;

if( pid\_log\_counter >= 10 ) { // (update rate / desired output rate) = (50hz / 10hz) = 5hz

pid\_log\_counter = 0;

Log\_Write\_PID(CH6\_THR\_ACCEL\_KP, z\_accel\_error, p, i, d, output, tuning\_value);

}

}

#endif

return output;

}

// get\_pilot\_desired\_throttle - transform pilot's throttle input to make cruise throttle mid stick

// used only for manual throttle modes

// returns throttle output 0 to 1000

#define THROTTLE\_IN\_MIDDLE 500 // the throttle mid point

static int16\_t get\_pilot\_desired\_throttle(int16\_t throttle\_control)

{

int16\_t throttle\_out;

// exit immediately in the simple cases

if( throttle\_control == 0 || g.throttle\_mid == 500) {

return throttle\_control;

}

// ensure reasonable throttle values

throttle\_control = constrain(throttle\_control,0,1000);

g.throttle\_mid = constrain(g.throttle\_mid,300,700);

// check throttle is above, below or in the deadband

if (throttle\_control < THROTTLE\_IN\_MIDDLE) {

// below the deadband

throttle\_out = g.throttle\_min + ((float)(throttle\_control-g.throttle\_min))\*((float)(g.throttle\_mid - g.throttle\_min))/((float)(500-g.throttle\_min));

}else if(throttle\_control > THROTTLE\_IN\_MIDDLE) {

// above the deadband

throttle\_out = g.throttle\_mid + ((float)(throttle\_control-500))\*(float)(1000-g.throttle\_mid)/500.0f;

}else{

// must be in the deadband

throttle\_out = g.throttle\_mid;

}

return throttle\_out;

}

// get\_pilot\_desired\_climb\_rate - transform pilot's throttle input to

// climb rate in cm/s. we use radio\_in instead of control\_in to get the full range

// without any deadzone at the bottom

#define THROTTLE\_IN\_DEADBAND 100 // the throttle input channel's deadband in PWM

#define THROTTLE\_IN\_DEADBAND\_TOP (THROTTLE\_IN\_MIDDLE+THROTTLE\_IN\_DEADBAND) // top of the deadband

#define THROTTLE\_IN\_DEADBAND\_BOTTOM (THROTTLE\_IN\_MIDDLE-THROTTLE\_IN\_DEADBAND) // bottom of the deadband

static int16\_t get\_pilot\_desired\_climb\_rate(int16\_t throttle\_control)

{

int16\_t desired\_rate = 0;

// throttle failsafe check

if( ap.failsafe ) {

return 0;

}

// ensure a reasonable throttle value

throttle\_control = constrain(throttle\_control,0,1000);

// check throttle is above, below or in the deadband

if (throttle\_control < THROTTLE\_IN\_DEADBAND\_BOTTOM) {

// below the deadband

desired\_rate = (int32\_t)g.pilot\_velocity\_z\_max \* (throttle\_control-THROTTLE\_IN\_DEADBAND\_BOTTOM) / (THROTTLE\_IN\_MIDDLE - THROTTLE\_IN\_DEADBAND);

}else if (throttle\_control > THROTTLE\_IN\_DEADBAND\_TOP) {

// above the deadband

desired\_rate = (int32\_t)g.pilot\_velocity\_z\_max \* (throttle\_control-THROTTLE\_IN\_DEADBAND\_TOP) / (THROTTLE\_IN\_MIDDLE - THROTTLE\_IN\_DEADBAND);

}else{

// must be in the deadband

desired\_rate = 0;

}

// desired climb rate for logging

desired\_climb\_rate = desired\_rate;

return desired\_rate;

}

// get\_pilot\_desired\_acceleration - transform pilot's throttle input to a desired acceleration

// default upper and lower bounds are 500 cm/s/s (roughly 1/2 a G)

// returns acceleration in cm/s/s

static int16\_t get\_pilot\_desired\_acceleration(int16\_t throttle\_control)

{

int32\_t desired\_accel = 0;

// throttle failsafe check

if( ap.failsafe ) {

return 0;

}

// ensure a reasonable throttle value

throttle\_control = constrain(throttle\_control,0,1000);

// check throttle is above, below or in the deadband

if (throttle\_control < THROTTLE\_IN\_DEADBAND\_BOTTOM) {

// below the deadband

desired\_accel = (int32\_t)ACCELERATION\_MAX\_Z \* (throttle\_control-THROTTLE\_IN\_DEADBAND\_BOTTOM) / (THROTTLE\_IN\_MIDDLE - THROTTLE\_IN\_DEADBAND);

}else if(throttle\_control > THROTTLE\_IN\_DEADBAND\_TOP) {

// above the deadband

desired\_accel = (int32\_t)ACCELERATION\_MAX\_Z \* (throttle\_control-THROTTLE\_IN\_DEADBAND\_TOP) / (THROTTLE\_IN\_MIDDLE - THROTTLE\_IN\_DEADBAND);

}else{

// must be in the deadband

desired\_accel = 0;

}

return desired\_accel;

}

// get\_pilot\_desired\_direct\_alt - transform pilot's throttle input to a desired altitude

// return altitude in cm between 0 to 10m

static int32\_t get\_pilot\_desired\_direct\_alt(int16\_t throttle\_control)

{

int32\_t desired\_alt = 0;

// throttle failsafe check

if( ap.failsafe ) {

return 0;

}

// ensure a reasonable throttle value

throttle\_control = constrain(throttle\_control,0,1000);

desired\_alt = throttle\_control;

return desired\_alt;

}

// get\_throttle\_rate - calculates desired accel required to achieve desired z\_target\_speed

// sets accel based throttle controller target

static void

get\_throttle\_rate(int16\_t z\_target\_speed)

{

static uint32\_t last\_call\_ms = 0;

static float z\_rate\_error = 0; // The velocity error in cm.

int32\_t p,i,d; // used to capture pid values for logging

int16\_t output; // the target acceleration if the accel based throttle is enabled, otherwise the output to be sent to the motors

uint32\_t now = millis();

// reset target altitude if this controller has just been engaged

if( now - last\_call\_ms > 100 ) {

// Reset Filter

z\_rate\_error = 0;

} else {

// calculate rate error and filter with cut off frequency of 2 Hz

z\_rate\_error = z\_rate\_error + 0.20085 \* ((z\_target\_speed - climb\_rate) - z\_rate\_error);

}

last\_call\_ms = now;

// separately calculate p, i, d values for logging

p = g.pid\_throttle.get\_p(z\_rate\_error);

// freeze I term if we've breached throttle limits

if(motors.reached\_limit(AP\_MOTOR\_THROTTLE\_LIMIT)) {

i = g.pid\_throttle.get\_integrator();

}else{

i = g.pid\_throttle.get\_i(z\_rate\_error, .02);

}

d = g.pid\_throttle.get\_d(z\_rate\_error, .02);

// consolidate target acceleration

output = p+i+d;

#if LOGGING\_ENABLED == ENABLED

// log output if PID loggins is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_THROTTLE\_KP || g.radio\_tuning == CH6\_THROTTLE\_KI || g.radio\_tuning == CH6\_THROTTLE\_KD) ) {

pid\_log\_counter++;

if( pid\_log\_counter >= 10 ) { // (update rate / desired output rate) = (50hz / 10hz) = 5hz

pid\_log\_counter = 0;

Log\_Write\_PID(CH6\_THROTTLE\_KP, z\_rate\_error, p, i, d, output, tuning\_value);

}

}

#endif

// send output to accelerometer based throttle controller if enabled otherwise send directly to motors

if( g.throttle\_accel\_enabled ) {

// set target for accel based throttle controller

set\_throttle\_accel\_target(output);

}else{

set\_throttle\_out(g.throttle\_cruise+output, true);

}

// update throttle cruise

// TO-DO: this may not be correct because g.rc\_3.servo\_out has not been updated for this iteration

if( z\_target\_speed == 0 ) {

update\_throttle\_cruise(g.rc\_3.servo\_out);

}

}

// get\_throttle\_althold - hold at the desired altitude in cm

// updates accel based throttle controller targets

// Note: max\_climb\_rate is an optional parameter to allow reuse of this function by landing controller

static void

get\_throttle\_althold(int32\_t target\_alt, int16\_t min\_climb\_rate, int16\_t max\_climb\_rate)

{

int32\_t alt\_error;

int16\_t desired\_rate;

int32\_t linear\_distance; // the distace we swap between linear and sqrt.

// calculate altitude error

alt\_error = target\_alt - current\_loc.alt;

// check kP to avoid division by zero

if( g.pi\_alt\_hold.kP() != 0 ) {

linear\_distance = 250/(2\*g.pi\_alt\_hold.kP()\*g.pi\_alt\_hold.kP());

if( alt\_error > 2\*linear\_distance ) {

desired\_rate = safe\_sqrt(2\*250\*(alt\_error-linear\_distance));

}else if( alt\_error < -2\*linear\_distance ) {

desired\_rate = -safe\_sqrt(2\*250\*(-alt\_error-linear\_distance));

}else{

desired\_rate = g.pi\_alt\_hold.get\_p(alt\_error);

}

}else{

desired\_rate = 0;

}

desired\_rate = constrain(desired\_rate, min\_climb\_rate, max\_climb\_rate);

// call rate based throttle controller which will update accel based throttle controller targets

get\_throttle\_rate(desired\_rate);

// update altitude error reported to GCS

altitude\_error = alt\_error;

// TO-DO: enabled PID logging for this controller

}

// get\_throttle\_althold\_with\_slew - altitude controller with slew to avoid step changes in altitude target

// calls normal althold controller which updates accel based throttle controller targets

static void

get\_throttle\_althold\_with\_slew(int16\_t target\_alt, int16\_t min\_climb\_rate, int16\_t max\_climb\_rate)

{

// limit target altitude change

controller\_desired\_alt += constrain(target\_alt-controller\_desired\_alt, min\_climb\_rate\*0.02, max\_climb\_rate\*0.02);

// do not let target altitude get too far from current altitude

controller\_desired\_alt = constrain(controller\_desired\_alt,current\_loc.alt-750,current\_loc.alt+750);

get\_throttle\_althold(controller\_desired\_alt, min\_climb\_rate-250, max\_climb\_rate+250); // 250 is added to give head room to alt hold controller

}

// get\_throttle\_rate\_stabilized - rate controller with additional 'stabilizer'

// 'stabilizer' ensure desired rate is being met

// calls normal throttle rate controller which updates accel based throttle controller targets

static void

get\_throttle\_rate\_stabilized(int16\_t target\_rate)

{

controller\_desired\_alt += target\_rate \* 0.02;

// do not let target altitude get too far from current altitude

controller\_desired\_alt = constrain(controller\_desired\_alt,current\_loc.alt-750,current\_loc.alt+750);

set\_new\_altitude(controller\_desired\_alt);

get\_throttle\_althold(controller\_desired\_alt, -g.pilot\_velocity\_z\_max-250, g.pilot\_velocity\_z\_max+250); // 250 is added to give head room to alt hold controller

}

// get\_throttle\_land - high level landing logic

// sends the desired acceleration in the accel based throttle controller

// called at 50hz

static void

get\_throttle\_land()

{

// if we are above 10m and the sonar does not sense anything perform regular alt hold descent

if (current\_loc.alt >= LAND\_START\_ALT && !(g.sonar\_enabled && sonar\_alt\_health >= SONAR\_ALT\_HEALTH\_MAX)) {

get\_throttle\_althold\_with\_slew(LAND\_START\_ALT, g.auto\_velocity\_z\_min, -abs(g.land\_speed));

}else{

get\_throttle\_rate\_stabilized(-abs(g.land\_speed));

// detect whether we have landed by watching for minimum throttle and now movement

if (abs(climb\_rate) < 20 && (g.rc\_3.servo\_out <= get\_angle\_boost(g.throttle\_min) || g.pid\_throttle\_accel.get\_integrator() <= -150)) {

if( land\_detector < LAND\_DETECTOR\_TRIGGER ) {

land\_detector++;

}else{

set\_land\_complete(true);

if( g.rc\_3.control\_in == 0 || ap.failsafe ) {

init\_disarm\_motors();

}

}

}else{

// we've sensed movement up or down so decrease land\_detector

if (land\_detector > 0 ) {

land\_detector--;

}

}

}

}

// get\_throttle\_surface\_tracking - hold copter at the desired distance above the ground

// updates accel based throttle controller targets

static void

get\_throttle\_surface\_tracking(int16\_t target\_rate)

{

static float target\_sonar\_alt = 0; // The desired altitude in cm above the ground

static uint32\_t last\_call\_ms = 0;

float distance\_error;

float sonar\_induced\_slew\_rate;

uint32\_t now = millis();

// reset target altitude if this controller has just been engaged

if( now - last\_call\_ms > 200 ) {

target\_sonar\_alt = sonar\_alt + controller\_desired\_alt - current\_loc.alt;

}

last\_call\_ms = now;

target\_sonar\_alt += target\_rate \* 0.02;

distance\_error = (target\_sonar\_alt-sonar\_alt);

sonar\_induced\_slew\_rate = constrain(fabs(THR\_SURFACE\_TRACKING\_P \* distance\_error),0,THR\_SURFACE\_TRACKING\_VELZ\_MAX);

// do not let target altitude get too far from current altitude above ground

// Note: the 750cm limit is perhaps too wide but is consistent with the regular althold limits and helps ensure a smooth transition

target\_sonar\_alt = constrain(target\_sonar\_alt,sonar\_alt-750,sonar\_alt+750);

controller\_desired\_alt = current\_loc.alt+(target\_sonar\_alt-sonar\_alt);

set\_new\_altitude(controller\_desired\_alt);

get\_throttle\_althold\_with\_slew(controller\_desired\_alt, target\_rate-sonar\_induced\_slew\_rate, target\_rate+sonar\_induced\_slew\_rate); // VELZ\_MAX limits how quickly we react

}

/\*

\* reset all I integrators

\*/

static void reset\_I\_all(void)

{

reset\_rate\_I();

reset\_stability\_I();

reset\_wind\_I();

reset\_throttle\_I();

reset\_optflow\_I();

// This is the only place we reset Yaw

g.pi\_stabilize\_yaw.reset\_I();

}

static void reset\_rate\_I()

{

g.pid\_rate\_roll.reset\_I();

g.pid\_rate\_pitch.reset\_I();

g.pid\_rate\_yaw.reset\_I();

}

static void reset\_optflow\_I(void)

{

g.pid\_optflow\_roll.reset\_I();

g.pid\_optflow\_pitch.reset\_I();

of\_roll = 0;

of\_pitch = 0;

}

static void reset\_wind\_I(void)

{

// Wind Compensation

// this i is not currently being used, but we reset it anyway

// because someone may modify it and not realize it, causing a bug

g.pi\_loiter\_lat.reset\_I();

g.pi\_loiter\_lon.reset\_I();

g.pid\_loiter\_rate\_lat.reset\_I();

g.pid\_loiter\_rate\_lon.reset\_I();

g.pid\_nav\_lat.reset\_I();

g.pid\_nav\_lon.reset\_I();

}

static void reset\_throttle\_I(void)

{

// For Altitude Hold

g.pi\_alt\_hold.reset\_I();

g.pid\_throttle.reset\_I();

g.pid\_throttle\_accel.reset\_I();

}

static void set\_accel\_throttle\_I\_from\_pilot\_throttle(int16\_t pilot\_throttle)

{

// shift difference between pilot's throttle and hover throttle into accelerometer I

g.pid\_throttle\_accel.set\_integrator(pilot\_throttle-g.throttle\_cruise);

}

static void reset\_stability\_I(void)

{

// Used to balance a quad

// This only needs to be reset during Auto-leveling in flight

g.pi\_stabilize\_roll.reset\_I();

g.pi\_stabilize\_pitch.reset\_I();

}

## GCS.h

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

/// @file GCS.h

/// @brief Interface definition for the various Ground Control System

// protocols.

#ifndef \_\_GCS\_H

#define \_\_GCS\_H

#include <FastSerial.h>

#include <AP\_Common.h>

#include <GPS.h>

#include <Stream.h>

#include <stdint.h>

///

/// @class GCS

/// @brief Class describing the interface between the APM code

/// proper and the GCS implementation.

///

/// GCS' are currently implemented inside the sketch and as such have

/// access to all global state. The sketch should not, however, call GCS

/// internal functions - all calls to the GCS should be routed through

/// this interface (or functions explicitly exposed by a subclass).

///

class GCS\_Class

{

public:

/// Startup initialisation.

///

/// This routine performs any one-off initialisation required before

/// GCS messages are exchanged.

///

/// @note The stream is expected to be set up and configured for the

/// correct bitrate before ::init is called.

///

/// @note The stream is currently BetterStream so that we can use the \_P

/// methods; this may change if Arduino adds them to Print.

///

/// @param port The stream over which messages are exchanged.

///

void init(FastSerial \*port) {

\_port = port;

initialised = true;

}

/// Update GCS state.

///

/// This may involve checking for received bytes on the stream,

/// or sending additional periodic messages.

void update(void) {

}

/// Send a message with a single numeric parameter.

///

/// This may be a standalone message, or the GCS driver may

/// have its own way of locating additional parameters to send.

///

/// @param id ID of the message to send.

/// @param param Explicit message parameter.

///

void send\_message(enum ap\_message id) {

}

/// Send a text message.

///

/// @param severity A value describing the importance of the message.

/// @param str The text to be sent.

///

void send\_text(gcs\_severity severity, const char \*str) {

}

/// Send a text message with a PSTR()

///

/// @param severity A value describing the importance of the message.

/// @param str The text to be sent.

///

void send\_text(gcs\_severity severity, const prog\_char\_t \*str) {

}

// send streams which match frequency range

void data\_stream\_send(void);

// set to true if this GCS link is active

bool initialised;

protected:

/// The stream we are communicating over

FastSerial \* \_port;

};

//

// GCS class definitions.

//

// These are here so that we can declare the GCS object early in the sketch

// and then reference it statically rather than via a pointer.

//

///

/// @class GCS\_MAVLINK

/// @brief The mavlink protocol for qgroundcontrol

///

class GCS\_MAVLINK : public GCS\_Class

{

public:

GCS\_MAVLINK();

void update(void);

void init(FastSerial \*port);

void send\_message(enum ap\_message id);

void send\_text(gcs\_severity severity, const char \*str);

void send\_text(gcs\_severity severity, const prog\_char\_t \*str);

void data\_stream\_send(void);

void queued\_param\_send();

void queued\_waypoint\_send();

static const struct AP\_Param::GroupInfo var\_info[];

// NOTE! The streams enum below and the

// set of AP\_Int16 stream rates \_must\_ be

// kept in the same order

enum streams {STREAM\_RAW\_SENSORS,

STREAM\_EXTENDED\_STATUS,

STREAM\_RC\_CHANNELS,

STREAM\_RAW\_CONTROLLER,

STREAM\_POSITION,

STREAM\_EXTRA1,

STREAM\_EXTRA2,

STREAM\_EXTRA3,

STREAM\_PARAMS,

NUM\_STREAMS};

// see if we should send a stream now. Called at 50Hz

bool stream\_trigger(enum streams stream\_num);

private:

void handleMessage(mavlink\_message\_t \* msg);

/// Perform queued sending operations

///

AP\_Param \* \_queued\_parameter; ///< next parameter to

// be sent in queue

enum ap\_var\_type \_queued\_parameter\_type; ///< type of the next

// parameter

AP\_Param::ParamToken \_queued\_parameter\_token; ///AP\_Param token for

// next() call

uint16\_t \_queued\_parameter\_index; ///< next queued

// parameter's index

uint16\_t \_queued\_parameter\_count; ///< saved count of

// parameters for

// queued send

/// Count the number of reportable parameters.

///

/// Not all parameters can be reported via MAVlink. We count the number

// that are

/// so that we can report to a GCS the number of parameters it should

// expect when it

/// requests the full set.

///

/// @return The number of reportable parameters.

///

uint16\_t \_count\_parameters(); ///< count reportable

// parameters

uint16\_t \_parameter\_count; ///< cache of reportable

// parameters

mavlink\_channel\_t chan;

uint16\_t packet\_drops;

#if CLI\_ENABLED == ENABLED

// this allows us to detect the user wanting the CLI to start

uint8\_t crlf\_count;

#endif

// waypoints

uint16\_t waypoint\_request\_i; // request index

uint16\_t waypoint\_dest\_sysid; // where to send requests

uint16\_t waypoint\_dest\_compid; // "

bool waypoint\_sending; // currently in send process

bool waypoint\_receiving; // currently receiving

uint16\_t waypoint\_count;

uint32\_t waypoint\_timelast\_send; // milliseconds

uint32\_t waypoint\_timelast\_receive; // milliseconds

uint32\_t waypoint\_timelast\_request; // milliseconds

uint16\_t waypoint\_send\_timeout; // milliseconds

uint16\_t waypoint\_receive\_timeout; // milliseconds

// data stream rates. The code assumes that

// streamRateRawSensors is the first

AP\_Int16 streamRateRawSensors;

AP\_Int16 streamRateExtendedStatus;

AP\_Int16 streamRateRCChannels;

AP\_Int16 streamRateRawController;

AP\_Int16 streamRatePosition;

AP\_Int16 streamRateExtra1;

AP\_Int16 streamRateExtra2;

AP\_Int16 streamRateExtra3;

AP\_Int16 streamRateParams;

// number of 50Hz ticks until we next send this stream

uint8\_t stream\_ticks[NUM\_STREAMS];

// number of extra ticks to add to slow things down for the radio

uint8\_t stream\_slowdown;

};

#endif // \_\_GCS\_H

## GCS

/// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

/\*

\* GCS Protocol

\*

\* 4 Ardupilot Header

\* D

\* 5 Payload length

\* 1 Message ID

\* 1 Message Version

\* 9 Payload byte 1

\* 8 Payload byte 2

\* 7 Payload byte 3

\* A Checksum byte 1

\* B Checksum byte 2

\*

\*/

/\*

\* #if GCS\_PORT == 3

# define SendSerw Serial3.write

# define SendSer Serial3.print

##else

# define SendSerw Serial.write

# define SendSer Serial.print

##endif

#

# byte mess\_buffer[60];

# byte buff\_pointer;

#

# // Unions for getting byte values

# union f\_out{

# byte bytes[4];

# float value;

# } floatOut;

#

# union i\_out {

# byte bytes[2];

# int16\_t value;

# } intOut;

#

# union l\_out{

# byte bytes[4];

# int32\_t value;

# } longOut;

#

# // Add binary values to the buffer

# void write\_byte(byte val)

# {

# mess\_buffer[buff\_pointer++] = val;

# }

#

# void write\_int(int16\_t val )

# {

# intOut.value = val;

# mess\_buffer[buff\_pointer++] = intOut.bytes[0];

# mess\_buffer[buff\_pointer++] = intOut.bytes[1];

# }

#

# void write\_float(float val)

# {

# floatOut.value = val;

# mess\_buffer[buff\_pointer++] = floatOut.bytes[0];

# mess\_buffer[buff\_pointer++] = floatOut.bytes[1];

# mess\_buffer[buff\_pointer++] = floatOut.bytes[2];

# mess\_buffer[buff\_pointer++] = floatOut.bytes[3];

# }

#

# void write\_long(int32\_t val)

# {

# longOut.value = val;

# mess\_buffer[buff\_pointer++] = longOut.bytes[0];

# mess\_buffer[buff\_pointer++] = longOut.bytes[1];

# mess\_buffer[buff\_pointer++] = longOut.bytes[2];

# mess\_buffer[buff\_pointer++] = longOut.bytes[3];

# }

#

# void flush(byte id)

# {

# byte mess\_ck\_a = 0;

# byte mess\_ck\_b = 0;

# byte i;

#

# SendSer("4D"); // This is the message preamble

# SendSerw(buff\_pointer); // Length

# SendSerw(2); // id

#

# for (i = 0; i < buff\_pointer; i++) {

# SendSerw(mess\_buffer[i]);

# }

#

# buff\_pointer = 0;

# }

\*/

## GCS\_Mavlink

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

// use this to prevent recursion during sensor init

static bool in\_mavlink\_delay;

// this costs us 51 bytes, but means that low priority

// messages don't block the CPU

static mavlink\_statustext\_t pending\_status;

// true when we have received at least 1 MAVLink packet

static bool mavlink\_active;

// check if a message will fit in the payload space available

#define CHECK\_PAYLOAD\_SIZE(id) if (payload\_space < MAVLINK\_MSG\_ID\_ ## id ## \_LEN) return false

// prototype this for use inside the GCS class

static void gcs\_send\_text\_fmt(const prog\_char\_t \*fmt, ...);

// gcs\_check - throttles communication with ground station.

// should be called regularly

// returns true if it has sent a message to the ground station

static bool gcs\_check()

{

static uint32\_t last\_1hz, last\_50hz;

bool sent\_message = false;

uint32\_t tnow = millis();

if (tnow - last\_1hz > 1000) {

last\_1hz = tnow;

gcs\_send\_message(MSG\_HEARTBEAT);

sent\_message = true;

}

if (tnow - last\_50hz > 20 && !sent\_message) {

last\_50hz = tnow;

gcs\_update();

gcs\_data\_stream\_send();

sent\_message = true;

}

gcs\_send\_message(MSG\_RETRY\_DEFERRED);

return sent\_message;

}

/\*

\* !!NOTE!!

\*

\* the use of NOINLINE separate functions for each message type avoids

\* a compiler bug in gcc that would cause it to use far more stack

\* space than is needed. Without the NOINLINE we use the sum of the

\* stack needed for each message type. Please be careful to follow the

\* pattern below when adding any new messages

\*/

static NOINLINE void send\_heartbeat(mavlink\_channel\_t chan)

{

uint8\_t base\_mode = MAV\_MODE\_FLAG\_CUSTOM\_MODE\_ENABLED;

uint8\_t system\_status = MAV\_STATE\_ACTIVE;

uint32\_t custom\_mode = control\_mode;

// work out the base\_mode. This value is not very useful

// for APM, but we calculate it as best we can so a generic

// MAVLink enabled ground station can work out something about

// what the MAV is up to. The actual bit values are highly

// ambiguous for most of the APM flight modes. In practice, you

// only get useful information from the custom\_mode, which maps to

// the APM flight mode and has a well defined meaning in the

// ArduPlane documentation

base\_mode = MAV\_MODE\_FLAG\_STABILIZE\_ENABLED;

switch (control\_mode) {

case AUTO:

case RTL:

case LOITER:

case GUIDED:

case CIRCLE:

base\_mode |= MAV\_MODE\_FLAG\_GUIDED\_ENABLED;

// note that MAV\_MODE\_FLAG\_AUTO\_ENABLED does not match what

// APM does in any mode, as that is defined as "system finds its own goal

// positions", which APM does not currently do

break;

}

// all modes except INITIALISING have some form of manual

// override if stick mixing is enabled

base\_mode |= MAV\_MODE\_FLAG\_MANUAL\_INPUT\_ENABLED;

#if HIL\_MODE != HIL\_MODE\_DISABLED

base\_mode |= MAV\_MODE\_FLAG\_HIL\_ENABLED;

#endif

// we are armed if we are not initialising

if (motors.armed()) {

base\_mode |= MAV\_MODE\_FLAG\_SAFETY\_ARMED;

}

// indicate we have set a custom mode

base\_mode |= MAV\_MODE\_FLAG\_CUSTOM\_MODE\_ENABLED;

mavlink\_msg\_heartbeat\_send(

chan,

MAV\_TYPE\_QUADROTOR,

MAV\_AUTOPILOT\_ARDUPILOTMEGA,

base\_mode,

custom\_mode,

system\_status);

}

static NOINLINE void send\_attitude(mavlink\_channel\_t chan)

{

mavlink\_msg\_attitude\_send(

chan,

millis(),

ahrs.roll,

ahrs.pitch,

ahrs.yaw,

omega.x,

omega.y,

omega.z);

}

#if AP\_LIMITS == ENABLED

static NOINLINE void send\_limits\_status(mavlink\_channel\_t chan)

{

limits\_send\_mavlink\_status(chan);

}

#endif

static NOINLINE void send\_extended\_status1(mavlink\_channel\_t chan, uint16\_t packet\_drops)

{

uint32\_t control\_sensors\_present = 0;

uint32\_t control\_sensors\_enabled;

uint32\_t control\_sensors\_health;

// first what sensors/controllers we have

control\_sensors\_present |= (1<<0); // 3D gyro present

control\_sensors\_present |= (1<<1); // 3D accelerometer present

if (g.compass\_enabled) {

control\_sensors\_present |= (1<<2); // compass present

}

control\_sensors\_present |= (1<<3); // absolute pressure sensor present

if (g\_gps != NULL && g\_gps->status() == GPS::GPS\_OK) {

control\_sensors\_present |= (1<<5); // GPS present

}

control\_sensors\_present |= (1<<10); // 3D angular rate control

control\_sensors\_present |= (1<<11); // attitude stabilisation

control\_sensors\_present |= (1<<12); // yaw position

control\_sensors\_present |= (1<<13); // altitude control

control\_sensors\_present |= (1<<14); // X/Y position control

control\_sensors\_present |= (1<<15); // motor control

// now what sensors/controllers are enabled

// first the sensors

control\_sensors\_enabled = control\_sensors\_present & 0x1FF;

// now the controllers

control\_sensors\_enabled = control\_sensors\_present & 0x1FF;

control\_sensors\_enabled |= (1<<10); // 3D angular rate control

control\_sensors\_enabled |= (1<<11); // attitude stabilisation

control\_sensors\_enabled |= (1<<13); // altitude control

control\_sensors\_enabled |= (1<<15); // motor control

switch (control\_mode) {

case AUTO:

case RTL:

case LOITER:

case GUIDED:

case CIRCLE:

case POSITION:

control\_sensors\_enabled |= (1<<12); // yaw position

control\_sensors\_enabled |= (1<<14); // X/Y position control

break;

}

// at the moment all sensors/controllers are assumed healthy

control\_sensors\_health = control\_sensors\_present;

if (!compass.healthy) {

control\_sensors\_health &= ~(1<<2); // compass

}

if (!compass.use\_for\_yaw()) {

control\_sensors\_enabled &= ~(1<<2); // compass

}

uint16\_t battery\_current = -1;

uint8\_t battery\_remaining = -1;

if (current\_total1 != 0 && g.pack\_capacity != 0) {

battery\_remaining = (100.0 \* (g.pack\_capacity - current\_total1) / g.pack\_capacity);

}

if (current\_total1 != 0) {

battery\_current = current\_amps1 \* 100;

}

if (g.battery\_monitoring == 3) {

/\*setting a out-of-range value.

\* It informs to external devices that

\* it cannot be calculated properly just by voltage\*/

battery\_remaining = 150;

}

mavlink\_msg\_sys\_status\_send(

chan,

control\_sensors\_present,

control\_sensors\_enabled,

control\_sensors\_health,

0, // CPU Load not supported in AC yet

battery\_voltage1 \* 1000, // mV

battery\_current, // in 10mA units

battery\_remaining, // in %

0, // comm drops %,

0, // comm drops in pkts,

0, 0, 0, 0);

}

static void NOINLINE send\_meminfo(mavlink\_channel\_t chan)

{

extern unsigned \_\_brkval;

mavlink\_msg\_meminfo\_send(chan, \_\_brkval, memcheck\_available\_memory());

}

static void NOINLINE send\_location(mavlink\_channel\_t chan)

{

uint32\_t fix\_time;

// if we have a GPS fix, take the time as the last fix time. That

// allows us to correctly calculate velocities and extrapolate

// positions.

// If we don't have a GPS fix then we are dead reckoning, and will

// use the current boot time as the fix time.

if (g\_gps->status() == GPS::GPS\_OK) {

fix\_time = g\_gps->last\_fix\_time;

} else {

fix\_time = millis();

}

mavlink\_msg\_global\_position\_int\_send(

chan,

fix\_time,

current\_loc.lat, // in 1E7 degrees

current\_loc.lng, // in 1E7 degrees

g\_gps->altitude \* 10, // millimeters above sea level

(current\_loc.alt - home.alt) \* 10, // millimeters above ground

g\_gps->velocity\_north() \* 100, // X speed cm/s (+ve North)

g\_gps->velocity\_east() \* 100, // Y speed cm/s (+ve East)

g\_gps->velocity\_down() \* -100, // Z speed cm/s (+ve up)

g\_gps->ground\_course); // course in 1/100 degree

}

static void NOINLINE send\_nav\_controller\_output(mavlink\_channel\_t chan)

{

mavlink\_msg\_nav\_controller\_output\_send(

chan,

nav\_roll / 1.0e2,

nav\_pitch / 1.0e2,

wp\_bearing / 1.0e2,

wp\_bearing / 1.0e2,

wp\_distance / 1.0e2,

altitude\_error / 1.0e2,

0,

crosstrack\_error); // was 0

}

static void NOINLINE send\_ahrs(mavlink\_channel\_t chan)

{

Vector3f omega\_I = ahrs.get\_gyro\_drift();

mavlink\_msg\_ahrs\_send(

chan,

omega\_I.x,

omega\_I.y,

omega\_I.z,

1,

0,

ahrs.get\_error\_rp(),

ahrs.get\_error\_yaw());

}

#ifdef DESKTOP\_BUILD

// report simulator state

static void NOINLINE send\_simstate(mavlink\_channel\_t chan)

{

sitl.simstate\_send(chan);

}

#endif

#ifndef DESKTOP\_BUILD

static void NOINLINE send\_hwstatus(mavlink\_channel\_t chan)

{

mavlink\_msg\_hwstatus\_send(

chan,

board\_voltage(),

I2c.lockup\_count());

}

#endif

static void NOINLINE send\_gps\_raw(mavlink\_channel\_t chan)

{

uint8\_t fix = g\_gps->status();

if (fix == GPS::GPS\_OK) {

fix = 3;

}

mavlink\_msg\_gps\_raw\_int\_send(

chan,

g\_gps->last\_fix\_time\*(uint64\_t)1000,

fix,

g\_gps->latitude, // in 1E7 degrees

g\_gps->longitude, // in 1E7 degrees

g\_gps->altitude \* 10, // in mm

g\_gps->hdop,

65535,

g\_gps->ground\_speed, // cm/s

g\_gps->ground\_course, // 1/100 degrees,

g\_gps->num\_sats);

}

static void NOINLINE send\_servo\_out(mavlink\_channel\_t chan)

{

// normalized values scaled to -10000 to 10000

// This is used for HIL. Do not change without discussing with HIL maintainers

#if FRAME\_CONFIG == HELI\_FRAME

mavlink\_msg\_rc\_channels\_scaled\_send(

chan,

millis(),

0, // port 0

g.rc\_1.servo\_out,

g.rc\_2.servo\_out,

g.rc\_3.radio\_out,

g.rc\_4.servo\_out,

0,

0,

0,

0,

receiver\_rssi);

#else

#if X\_PLANE == ENABLED

/\* update by JLN for X-Plane HIL \*/

if(motors.armed() && motors.auto\_armed()) {

mavlink\_msg\_rc\_channels\_scaled\_send(

chan,

millis(),

0, // port 0

g.rc\_1.servo\_out,

g.rc\_2.servo\_out,

10000 \* g.rc\_3.norm\_output(),

g.rc\_4.servo\_out,

10000 \* g.rc\_1.norm\_output(),

10000 \* g.rc\_2.norm\_output(),

10000 \* g.rc\_3.norm\_output(),

10000 \* g.rc\_4.norm\_output(),

receiver\_rssi);

}else{

mavlink\_msg\_rc\_channels\_scaled\_send(

chan,

millis(),

0, // port 0

0,

0,

-10000,

0,

10000 \* g.rc\_1.norm\_output(),

10000 \* g.rc\_2.norm\_output(),

10000 \* g.rc\_3.norm\_output(),

10000 \* g.rc\_4.norm\_output(),

receiver\_rssi);

}

#else

mavlink\_msg\_rc\_channels\_scaled\_send(

chan,

millis(),

0, // port 0

g.rc\_1.servo\_out,

g.rc\_2.servo\_out,

g.rc\_3.radio\_out,

g.rc\_4.servo\_out,

10000 \* g.rc\_1.norm\_output(),

10000 \* g.rc\_2.norm\_output(),

10000 \* g.rc\_3.norm\_output(),

10000 \* g.rc\_4.norm\_output(),

receiver\_rssi);

#endif

#endif

}

static void NOINLINE send\_radio\_in(mavlink\_channel\_t chan)

{

mavlink\_msg\_rc\_channels\_raw\_send(

chan,

millis(),

0, // port

g.rc\_1.radio\_in,

g.rc\_2.radio\_in,

g.rc\_3.radio\_in,

g.rc\_4.radio\_in,

g.rc\_5.radio\_in,

g.rc\_6.radio\_in,

g.rc\_7.radio\_in,

g.rc\_8.radio\_in,

receiver\_rssi);

}

static void NOINLINE send\_radio\_out(mavlink\_channel\_t chan)

{

mavlink\_msg\_servo\_output\_raw\_send(

chan,

micros(),

0, // port

motors.motor\_out[AP\_MOTORS\_MOT\_1],

motors.motor\_out[AP\_MOTORS\_MOT\_2],

motors.motor\_out[AP\_MOTORS\_MOT\_3],

motors.motor\_out[AP\_MOTORS\_MOT\_4],

motors.motor\_out[AP\_MOTORS\_MOT\_5],

motors.motor\_out[AP\_MOTORS\_MOT\_6],

motors.motor\_out[AP\_MOTORS\_MOT\_7],

motors.motor\_out[AP\_MOTORS\_MOT\_8]);

}

static void NOINLINE send\_vfr\_hud(mavlink\_channel\_t chan)

{

mavlink\_msg\_vfr\_hud\_send(

chan,

(float)g\_gps->ground\_speed / 100.0,

(float)g\_gps->ground\_speed / 100.0,

(ahrs.yaw\_sensor / 100) % 360,

g.rc\_3.servo\_out/10,

current\_loc.alt / 100.0,

climb\_rate / 100.0);

}

static void NOINLINE send\_raw\_imu1(mavlink\_channel\_t chan)

{

Vector3f accel = ins.get\_accel();

Vector3f gyro = ins.get\_gyro();

mavlink\_msg\_raw\_imu\_send(

chan,

micros(),

accel.x \* 1000.0 / gravity,

accel.y \* 1000.0 / gravity,

accel.z \* 1000.0 / gravity,

gyro.x \* 1000.0,

gyro.y \* 1000.0,

gyro.z \* 1000.0,

compass.mag\_x,

compass.mag\_y,

compass.mag\_z);

}

static void NOINLINE send\_raw\_imu2(mavlink\_channel\_t chan)

{

mavlink\_msg\_scaled\_pressure\_send(

chan,

millis(),

(float)barometer.get\_pressure()/100.0,

(float)(barometer.get\_pressure() - barometer.get\_ground\_pressure())/100.0,

(int)(barometer.get\_temperature()\*10));

}

static void NOINLINE send\_raw\_imu3(mavlink\_channel\_t chan)

{

Vector3f mag\_offsets = compass.get\_offsets();

Vector3f accel\_offsets = ins.get\_accel\_offsets();

Vector3f gyro\_offsets = ins.get\_gyro\_offsets();

mavlink\_msg\_sensor\_offsets\_send(chan,

mag\_offsets.x,

mag\_offsets.y,

mag\_offsets.z,

compass.get\_declination(),

barometer.get\_raw\_pressure(),

barometer.get\_raw\_temp(),

gyro\_offsets.x,

gyro\_offsets.y,

gyro\_offsets.z,

accel\_offsets.x,

accel\_offsets.y,

accel\_offsets.z);

}

static void NOINLINE send\_current\_waypoint(mavlink\_channel\_t chan)

{

mavlink\_msg\_mission\_current\_send(

chan,

(uint16\_t)g.command\_index);

}

static void NOINLINE send\_statustext(mavlink\_channel\_t chan)

{

mavlink\_msg\_statustext\_send(

chan,

pending\_status.severity,

pending\_status.text);

}

// are we still delaying telemetry to try to avoid Xbee bricking?

static bool telemetry\_delayed(mavlink\_channel\_t chan)

{

uint32\_t tnow = millis() >> 10;

if (tnow > (uint8\_t)g.telem\_delay) {

return false;

}

#if USB\_MUX\_PIN > 0

if (chan == MAVLINK\_COMM\_0 && ap\_system.usb\_connected) {

// this is an APM2 with USB telemetry

return false;

}

// we're either on the 2nd UART, or no USB cable is connected

// we need to delay telemetry

return true;

#else

if (chan == MAVLINK\_COMM\_0) {

// we're on the USB port

return false;

}

// don't send telemetry yet

return true;

#endif

}

// try to send a message, return false if it won't fit in the serial tx buffer

static bool mavlink\_try\_send\_message(mavlink\_channel\_t chan, enum ap\_message id, uint16\_t packet\_drops)

{

int16\_t payload\_space = comm\_get\_txspace(chan) - MAVLINK\_NUM\_NON\_PAYLOAD\_BYTES;

if (telemetry\_delayed(chan)) {

return false;

}

switch(id) {

case MSG\_HEARTBEAT:

CHECK\_PAYLOAD\_SIZE(HEARTBEAT);

send\_heartbeat(chan);

break;

case MSG\_EXTENDED\_STATUS1:

CHECK\_PAYLOAD\_SIZE(SYS\_STATUS);

send\_extended\_status1(chan, packet\_drops);

break;

case MSG\_EXTENDED\_STATUS2:

CHECK\_PAYLOAD\_SIZE(MEMINFO);

send\_meminfo(chan);

break;

case MSG\_ATTITUDE:

CHECK\_PAYLOAD\_SIZE(ATTITUDE);

send\_attitude(chan);

break;

case MSG\_LOCATION:

CHECK\_PAYLOAD\_SIZE(GLOBAL\_POSITION\_INT);

send\_location(chan);

break;

case MSG\_NAV\_CONTROLLER\_OUTPUT:

CHECK\_PAYLOAD\_SIZE(NAV\_CONTROLLER\_OUTPUT);

send\_nav\_controller\_output(chan);

break;

case MSG\_GPS\_RAW:

CHECK\_PAYLOAD\_SIZE(GPS\_RAW\_INT);

send\_gps\_raw(chan);

break;

case MSG\_SERVO\_OUT:

CHECK\_PAYLOAD\_SIZE(RC\_CHANNELS\_SCALED);

send\_servo\_out(chan);

break;

case MSG\_RADIO\_IN:

CHECK\_PAYLOAD\_SIZE(RC\_CHANNELS\_RAW);

send\_radio\_in(chan);

break;

case MSG\_RADIO\_OUT:

CHECK\_PAYLOAD\_SIZE(SERVO\_OUTPUT\_RAW);

send\_radio\_out(chan);

break;

case MSG\_VFR\_HUD:

CHECK\_PAYLOAD\_SIZE(VFR\_HUD);

send\_vfr\_hud(chan);

break;

case MSG\_RAW\_IMU1:

CHECK\_PAYLOAD\_SIZE(RAW\_IMU);

send\_raw\_imu1(chan);

break;

case MSG\_RAW\_IMU2:

CHECK\_PAYLOAD\_SIZE(SCALED\_PRESSURE);

send\_raw\_imu2(chan);

break;

case MSG\_RAW\_IMU3:

CHECK\_PAYLOAD\_SIZE(SENSOR\_OFFSETS);

send\_raw\_imu3(chan);

break;

case MSG\_CURRENT\_WAYPOINT:

CHECK\_PAYLOAD\_SIZE(MISSION\_CURRENT);

send\_current\_waypoint(chan);

break;

case MSG\_NEXT\_PARAM:

CHECK\_PAYLOAD\_SIZE(PARAM\_VALUE);

if (chan == MAVLINK\_COMM\_0) {

gcs0.queued\_param\_send();

} else if (gcs3.initialised) {

gcs3.queued\_param\_send();

}

break;

case MSG\_NEXT\_WAYPOINT:

CHECK\_PAYLOAD\_SIZE(MISSION\_REQUEST);

if (chan == MAVLINK\_COMM\_0) {

gcs0.queued\_waypoint\_send();

} else {

gcs3.queued\_waypoint\_send();

}

break;

case MSG\_STATUSTEXT:

CHECK\_PAYLOAD\_SIZE(STATUSTEXT);

send\_statustext(chan);

break;

#if AP\_LIMITS == ENABLED

case MSG\_LIMITS\_STATUS:

CHECK\_PAYLOAD\_SIZE(LIMITS\_STATUS);

send\_limits\_status(chan);

break;

#endif

case MSG\_AHRS:

CHECK\_PAYLOAD\_SIZE(AHRS);

send\_ahrs(chan);

break;

case MSG\_SIMSTATE:

#ifdef DESKTOP\_BUILD

CHECK\_PAYLOAD\_SIZE(SIMSTATE);

send\_simstate(chan);

#endif

break;

case MSG\_HWSTATUS:

#ifndef DESKTOP\_BUILD

CHECK\_PAYLOAD\_SIZE(HWSTATUS);

send\_hwstatus(chan);

#endif

break;

case MSG\_RETRY\_DEFERRED:

break; // just here to prevent a warning

}

return true;

}

#define MAX\_DEFERRED\_MESSAGES MSG\_RETRY\_DEFERRED

static struct mavlink\_queue {

enum ap\_message deferred\_messages[MAX\_DEFERRED\_MESSAGES];

uint8\_t next\_deferred\_message;

uint8\_t num\_deferred\_messages;

} mavlink\_queue[2];

// send a message using mavlink

static void mavlink\_send\_message(mavlink\_channel\_t chan, enum ap\_message id, uint16\_t packet\_drops)

{

uint8\_t i, nextid;

struct mavlink\_queue \*q = &mavlink\_queue[(uint8\_t)chan];

// see if we can send the deferred messages, if any

while (q->num\_deferred\_messages != 0) {

if (!mavlink\_try\_send\_message(chan,

q->deferred\_messages[q->next\_deferred\_message],

packet\_drops)) {

break;

}

q->next\_deferred\_message++;

if (q->next\_deferred\_message == MAX\_DEFERRED\_MESSAGES) {

q->next\_deferred\_message = 0;

}

q->num\_deferred\_messages--;

}

if (id == MSG\_RETRY\_DEFERRED) {

return;

}

// this message id might already be deferred

for (i=0, nextid = q->next\_deferred\_message; i < q->num\_deferred\_messages; i++) {

if (q->deferred\_messages[nextid] == id) {

// its already deferred, discard

return;

}

nextid++;

if (nextid == MAX\_DEFERRED\_MESSAGES) {

nextid = 0;

}

}

if (q->num\_deferred\_messages != 0 ||

!mavlink\_try\_send\_message(chan, id, packet\_drops)) {

// can't send it now, so defer it

if (q->num\_deferred\_messages == MAX\_DEFERRED\_MESSAGES) {

// the defer buffer is full, discard

return;

}

nextid = q->next\_deferred\_message + q->num\_deferred\_messages;

if (nextid >= MAX\_DEFERRED\_MESSAGES) {

nextid -= MAX\_DEFERRED\_MESSAGES;

}

q->deferred\_messages[nextid] = id;

q->num\_deferred\_messages++;

}

}

void mavlink\_send\_text(mavlink\_channel\_t chan, gcs\_severity severity, const char \*str)

{

if (telemetry\_delayed(chan)) {

return;

}

if (severity == SEVERITY\_LOW) {

// send via the deferred queuing system

pending\_status.severity = (uint8\_t)severity;

strncpy((char \*)pending\_status.text, str, sizeof(pending\_status.text));

mavlink\_send\_message(chan, MSG\_STATUSTEXT, 0);

} else {

// send immediately

mavlink\_msg\_statustext\_send(

chan,

severity,

str);

}

}

const AP\_Param::GroupInfo GCS\_MAVLINK::var\_info[] PROGMEM = {

AP\_GROUPINFO("RAW\_SENS", 0, GCS\_MAVLINK, streamRateRawSensors, 0),

AP\_GROUPINFO("EXT\_STAT", 1, GCS\_MAVLINK, streamRateExtendedStatus, 0),

AP\_GROUPINFO("RC\_CHAN", 2, GCS\_MAVLINK, streamRateRCChannels, 0),

AP\_GROUPINFO("RAW\_CTRL", 3, GCS\_MAVLINK, streamRateRawController, 0),

AP\_GROUPINFO("POSITION", 4, GCS\_MAVLINK, streamRatePosition, 0),

AP\_GROUPINFO("EXTRA1", 5, GCS\_MAVLINK, streamRateExtra1, 0),

AP\_GROUPINFO("EXTRA2", 6, GCS\_MAVLINK, streamRateExtra2, 0),

AP\_GROUPINFO("EXTRA3", 7, GCS\_MAVLINK, streamRateExtra3, 0),

AP\_GROUPINFO("PARAMS", 8, GCS\_MAVLINK, streamRateParams, 0),

AP\_GROUPEND

};

GCS\_MAVLINK::GCS\_MAVLINK() :

packet\_drops(0),

waypoint\_send\_timeout(1000), // 1 second

waypoint\_receive\_timeout(1000) // 1 second

{

}

void

GCS\_MAVLINK::init(FastSerial \* port)

{

GCS\_Class::init(port);

if (port == &Serial) {

mavlink\_comm\_0\_port = port;

chan = MAVLINK\_COMM\_0;

}else{

mavlink\_comm\_1\_port = port;

chan = MAVLINK\_COMM\_1;

}

\_queued\_parameter = NULL;

}

void

GCS\_MAVLINK::update(void)

{

// receive new packets

mavlink\_message\_t msg;

mavlink\_status\_t status;

status.packet\_rx\_drop\_count = 0;

// process received bytes

while(comm\_get\_available(chan))

{

uint8\_t c = comm\_receive\_ch(chan);

#if CLI\_ENABLED == ENABLED

/\* allow CLI to be started by hitting enter 3 times, if no

\* heartbeat packets have been received \*/

if (mavlink\_active == false) {

if (c == '\n' || c == '\r') {

crlf\_count++;

} else {

crlf\_count = 0;

}

if (crlf\_count == 3) {

run\_cli(\_port);

}

}

#endif

// Try to get a new message

if (mavlink\_parse\_char(chan, c, &msg, &status)) {

// we exclude radio packets to make it possible to use the

// CLI over the radio

if (msg.msgid != MAVLINK\_MSG\_ID\_RADIO) {

mavlink\_active = true;

}

handleMessage(&msg);

}

}

// Update packet drops counter

packet\_drops += status.packet\_rx\_drop\_count;

if (!waypoint\_receiving && !waypoint\_sending) {

return;

}

uint32\_t tnow = millis();

if (waypoint\_receiving &&

waypoint\_request\_i <= (unsigned)g.command\_total &&

tnow > waypoint\_timelast\_request + 500 + (stream\_slowdown\*20)) {

waypoint\_timelast\_request = tnow;

send\_message(MSG\_NEXT\_WAYPOINT);

}

// stop waypoint sending if timeout

if (waypoint\_sending && (tnow - waypoint\_timelast\_send) > waypoint\_send\_timeout) {

waypoint\_sending = false;

}

// stop waypoint receiving if timeout

if (waypoint\_receiving && (tnow - waypoint\_timelast\_receive) > waypoint\_receive\_timeout) {

waypoint\_receiving = false;

}

}

// see if we should send a stream now. Called at 50Hz

bool GCS\_MAVLINK::stream\_trigger(enum streams stream\_num)

{

AP\_Int16 \*stream\_rates = &streamRateRawSensors;

uint8\_t rate = (uint8\_t)stream\_rates[stream\_num].get();

if (rate == 0) {

return false;

}

if (stream\_ticks[stream\_num] == 0) {

// we're triggering now, setup the next trigger point

if (rate > 50) {

rate = 50;

}

stream\_ticks[stream\_num] = (50 / rate) + stream\_slowdown;

return true;

}

// count down at 50Hz

stream\_ticks[stream\_num]--;

return false;

}

void

GCS\_MAVLINK::data\_stream\_send(void)

{

if (waypoint\_receiving || waypoint\_sending) {

// don't interfere with mission transfer

return;

}

if (\_queued\_parameter != NULL) {

if (streamRateParams.get() <= 0) {

streamRateParams.set(50);

}

if (stream\_trigger(STREAM\_PARAMS)) {

send\_message(MSG\_NEXT\_PARAM);

}

// don't send anything else at the same time as parameters

return;

}

if (in\_mavlink\_delay) {

// don't send any other stream types while in the delay callback

return;

}

if (stream\_trigger(STREAM\_RAW\_SENSORS)) {

send\_message(MSG\_RAW\_IMU1);

send\_message(MSG\_RAW\_IMU2);

send\_message(MSG\_RAW\_IMU3);

//cliSerial->printf("mav1 %d\n", (int)streamRateRawSensors.get());

}

if (stream\_trigger(STREAM\_EXTENDED\_STATUS)) {

send\_message(MSG\_EXTENDED\_STATUS1);

send\_message(MSG\_EXTENDED\_STATUS2);

send\_message(MSG\_CURRENT\_WAYPOINT);

send\_message(MSG\_GPS\_RAW);

send\_message(MSG\_NAV\_CONTROLLER\_OUTPUT);

send\_message(MSG\_LIMITS\_STATUS);

}

if (stream\_trigger(STREAM\_POSITION)) {

send\_message(MSG\_LOCATION);

}

if (stream\_trigger(STREAM\_RAW\_CONTROLLER)) {

send\_message(MSG\_SERVO\_OUT);

//cliSerial->printf("mav4 %d\n", (int)streamRateRawController.get());

}

if (stream\_trigger(STREAM\_RC\_CHANNELS)) {

send\_message(MSG\_RADIO\_OUT);

send\_message(MSG\_RADIO\_IN);

//cliSerial->printf("mav5 %d\n", (int)streamRateRCChannels.get());

}

if (stream\_trigger(STREAM\_EXTRA1)) {

send\_message(MSG\_ATTITUDE);

send\_message(MSG\_SIMSTATE);

//cliSerial->printf("mav6 %d\n", (int)streamRateExtra1.get());

}

if (stream\_trigger(STREAM\_EXTRA2)) {

send\_message(MSG\_VFR\_HUD);

//cliSerial->printf("mav7 %d\n", (int)streamRateExtra2.get());

}

if (stream\_trigger(STREAM\_EXTRA3)) {

send\_message(MSG\_AHRS);

send\_message(MSG\_HWSTATUS);

}

}

void

GCS\_MAVLINK::send\_message(enum ap\_message id)

{

mavlink\_send\_message(chan,id, packet\_drops);

}

void

GCS\_MAVLINK::send\_text(gcs\_severity severity, const char \*str)

{

mavlink\_send\_text(chan,severity,str);

}

void

GCS\_MAVLINK::send\_text(gcs\_severity severity, const prog\_char\_t \*str)

{

mavlink\_statustext\_t m;

uint8\_t i;

for (i=0; i<sizeof(m.text); i++) {

m.text[i] = pgm\_read\_byte((const prog\_char \*)(str++));

}

if (i < sizeof(m.text)) m.text[i] = 0;

mavlink\_send\_text(chan, severity, (const char \*)m.text);

}

void GCS\_MAVLINK::handleMessage(mavlink\_message\_t\* msg)

{

struct Location tell\_command = {}; // command for telemetry

switch (msg->msgid) {

case MAVLINK\_MSG\_ID\_REQUEST\_DATA\_STREAM: //66

{

// decode

mavlink\_request\_data\_stream\_t packet;

mavlink\_msg\_request\_data\_stream\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system, packet.target\_component))

break;

int16\_t freq = 0; // packet frequency

if (packet.start\_stop == 0)

freq = 0; // stop sending

else if (packet.start\_stop == 1)

freq = packet.req\_message\_rate; // start sending

else

break;

switch(packet.req\_stream\_id) {

case MAV\_DATA\_STREAM\_ALL:

streamRateRawSensors = freq;

streamRateExtendedStatus = freq;

streamRateRCChannels = freq;

streamRateRawController = freq;

streamRatePosition = freq;

streamRateExtra1 = freq;

streamRateExtra2 = freq;

//streamRateExtra3.set\_and\_save(freq); // We just do set and save on the last as it takes care of the whole group.

streamRateExtra3 = freq; // Don't save!!

break;

case MAV\_DATA\_STREAM\_RAW\_SENSORS:

streamRateRawSensors = freq; // We do not set and save this one so that if HIL is shut down incorrectly

// we will not continue to broadcast raw sensor data at 50Hz.

break;

case MAV\_DATA\_STREAM\_EXTENDED\_STATUS:

//streamRateExtendedStatus.set\_and\_save(freq);

streamRateExtendedStatus = freq;

break;

case MAV\_DATA\_STREAM\_RC\_CHANNELS:

streamRateRCChannels = freq;

break;

case MAV\_DATA\_STREAM\_RAW\_CONTROLLER:

streamRateRawController = freq;

break;

//case MAV\_DATA\_STREAM\_RAW\_SENSOR\_FUSION:

// streamRateRawSensorFusion.set\_and\_save(freq);

// break;

case MAV\_DATA\_STREAM\_POSITION:

streamRatePosition = freq;

break;

case MAV\_DATA\_STREAM\_EXTRA1:

streamRateExtra1 = freq;

break;

case MAV\_DATA\_STREAM\_EXTRA2:

streamRateExtra2 = freq;

break;

case MAV\_DATA\_STREAM\_EXTRA3:

streamRateExtra3 = freq;

break;

default:

break;

}

break;

}

case MAVLINK\_MSG\_ID\_COMMAND\_LONG:

{

// decode

mavlink\_command\_long\_t packet;

mavlink\_msg\_command\_long\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system, packet.target\_component)) break;

uint8\_t result = MAV\_RESULT\_UNSUPPORTED;

// do command

send\_text(SEVERITY\_LOW,PSTR("command received: "));

switch(packet.command) {

case MAV\_CMD\_NAV\_LOITER\_UNLIM:

set\_mode(LOITER);

result = MAV\_RESULT\_ACCEPTED;

break;

case MAV\_CMD\_NAV\_RETURN\_TO\_LAUNCH:

set\_mode(RTL);

result = MAV\_RESULT\_ACCEPTED;

break;

case MAV\_CMD\_NAV\_LAND:

set\_mode(LAND);

result = MAV\_RESULT\_ACCEPTED;

break;

case MAV\_CMD\_MISSION\_START:

set\_mode(AUTO);

result = MAV\_RESULT\_ACCEPTED;

break;

case MAV\_CMD\_PREFLIGHT\_CALIBRATION:

if (packet.param1 == 1 ||

packet.param2 == 1 ||

packet.param3 == 1) {

ins.init\_accel(mavlink\_delay, flash\_leds); // level accelerometer values

ahrs.set\_trim(Vector3f(0,0,0)); // clear out saved trim

}

if (packet.param4 == 1) {

trim\_radio();

}

if (packet.param5 == 1) {

float trim\_roll, trim\_pitch;

// this blocks

ins.calibrate\_accel(mavlink\_delay, flash\_leds, gcs\_send\_text\_fmt, setup\_wait\_key, trim\_roll, trim\_pitch);

// reset ahrs's trim to suggested values from calibration routine

trim\_roll = constrain(trim\_roll, ToRad(-10.0f), ToRad(10.0f));

trim\_pitch = constrain(trim\_pitch, ToRad(-10.0f), ToRad(10.0f));

ahrs.set\_trim(Vector3f(trim\_roll, trim\_pitch, 0));

}

result = MAV\_RESULT\_ACCEPTED;

break;

case MAV\_CMD\_COMPONENT\_ARM\_DISARM:

if (packet.target\_component == MAV\_COMP\_ID\_SYSTEM\_CONTROL) {

if (packet.param1 == 1.0f) {

init\_arm\_motors();

result = MAV\_RESULT\_ACCEPTED;

} else if (packet.param1 == 0.0f) {

init\_disarm\_motors();

result = MAV\_RESULT\_ACCEPTED;

} else {

result = MAV\_RESULT\_UNSUPPORTED;

}

} else {

result = MAV\_RESULT\_UNSUPPORTED;

}

break;

case MAV\_CMD\_PREFLIGHT\_REBOOT\_SHUTDOWN:

if (packet.param1 == 1) {

reboot\_apm();

result = MAV\_RESULT\_ACCEPTED;

}

break;

default:

result = MAV\_RESULT\_UNSUPPORTED;

break;

}

mavlink\_msg\_command\_ack\_send(

chan,

packet.command,

result);

break;

}

case MAVLINK\_MSG\_ID\_SET\_MODE: //11

{

// decode

mavlink\_set\_mode\_t packet;

mavlink\_msg\_set\_mode\_decode(msg, &packet);

if (!(packet.base\_mode & MAV\_MODE\_FLAG\_CUSTOM\_MODE\_ENABLED)) {

// we ignore base\_mode as there is no sane way to map

// from that bitmap to a APM flight mode. We rely on

// custom\_mode instead.

break;

}

switch (packet.custom\_mode) {

case STABILIZE:

case ACRO:

case ALT\_HOLD:

case AUTO:

case GUIDED:

case LOITER:

case RTL:

case CIRCLE:

case POSITION:

case LAND:

case OF\_LOITER:

set\_mode(packet.custom\_mode);

break;

}

break;

}

/\*case MAVLINK\_MSG\_ID\_SET\_NAV\_MODE:

\* {

\* // decode

\* mavlink\_set\_nav\_mode\_t packet;

\* mavlink\_msg\_set\_nav\_mode\_decode(msg, &packet);

\* // To set some flight modes we must first receive a "set nav mode" message and then a "set mode" message

\* mav\_nav = packet.nav\_mode;

\* break;

\* }

\*/

case MAVLINK\_MSG\_ID\_MISSION\_REQUEST\_LIST: //43

{

//send\_text\_P(SEVERITY\_LOW,PSTR("waypoint request list"));

// decode

mavlink\_mission\_request\_list\_t packet;

mavlink\_msg\_mission\_request\_list\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system, packet.target\_component))

break;

// Start sending waypoints

mavlink\_msg\_mission\_count\_send(

chan,msg->sysid,

msg->compid,

g.command\_total); // includes home

waypoint\_timelast\_send = millis();

waypoint\_sending = true;

waypoint\_receiving = false;

waypoint\_dest\_sysid = msg->sysid;

waypoint\_dest\_compid = msg->compid;

break;

}

// XXX read a WP from EEPROM and send it to the GCS

case MAVLINK\_MSG\_ID\_MISSION\_REQUEST: // 40

{

//send\_text\_P(SEVERITY\_LOW,PSTR("waypoint request"));

// Check if sending waypiont

//if (!waypoint\_sending) break;

// 5/10/11 - We are trying out relaxing the requirement that we be in waypoint sending mode to respond to a waypoint request. DEW

// decode

mavlink\_mission\_request\_t packet;

mavlink\_msg\_mission\_request\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system, packet.target\_component))

break;

// send waypoint

tell\_command = get\_cmd\_with\_index(packet.seq);

// set frame of waypoint

uint8\_t frame;

if (tell\_command.options & MASK\_OPTIONS\_RELATIVE\_ALT) {

frame = MAV\_FRAME\_GLOBAL\_RELATIVE\_ALT; // reference frame

} else {

frame = MAV\_FRAME\_GLOBAL; // reference frame

}

float param1 = 0, param2 = 0, param3 = 0, param4 = 0;

// time that the mav should loiter in milliseconds

uint8\_t current = 0; // 1 (true), 0 (false)

if (packet.seq == (uint16\_t)g.command\_index)

current = 1;

uint8\_t autocontinue = 1; // 1 (true), 0 (false)

float x = 0, y = 0, z = 0;

if (tell\_command.id < MAV\_CMD\_NAV\_LAST) {

// command needs scaling

x = tell\_command.lat/1.0e7; // local (x), global (latitude)

y = tell\_command.lng/1.0e7; // local (y), global (longitude)

// ACM is processing alt inside each command. so we save and load raw values. - this is diffrent to APM

z = tell\_command.alt/1.0e2; // local (z), global/relative (altitude)

}

// Switch to map APM command fields into MAVLink command fields

switch (tell\_command.id) {

case MAV\_CMD\_NAV\_LOITER\_TURNS:

case MAV\_CMD\_CONDITION\_CHANGE\_ALT:

case MAV\_CMD\_DO\_SET\_HOME:

param1 = tell\_command.p1;

break;

case MAV\_CMD\_NAV\_ROI:

param1 = tell\_command.p1; // MAV\_ROI (aka roi mode) is held in wp's parameter but we actually do nothing with it because we only support pointing at a specific location provided by x,y and z parameters

break;

case MAV\_CMD\_CONDITION\_YAW:

param3 = tell\_command.p1;

param1 = tell\_command.alt;

param2 = tell\_command.lat;

param4 = tell\_command.lng;

break;

case MAV\_CMD\_NAV\_TAKEOFF:

param1 = 0;

break;

case MAV\_CMD\_NAV\_LOITER\_TIME:

param1 = tell\_command.p1; // ACM loiter time is in 1 second increments

break;

case MAV\_CMD\_CONDITION\_DELAY:

case MAV\_CMD\_CONDITION\_DISTANCE:

param1 = tell\_command.lat;

break;

case MAV\_CMD\_DO\_JUMP:

param2 = tell\_command.lat;

param1 = tell\_command.p1;

break;

case MAV\_CMD\_DO\_REPEAT\_SERVO:

param4 = tell\_command.lng;

case MAV\_CMD\_DO\_REPEAT\_RELAY:

case MAV\_CMD\_DO\_CHANGE\_SPEED:

param3 = tell\_command.lat;

param2 = tell\_command.alt;

param1 = tell\_command.p1;

break;

case MAV\_CMD\_NAV\_WAYPOINT:

param1 = tell\_command.p1;

break;

case MAV\_CMD\_DO\_SET\_PARAMETER:

case MAV\_CMD\_DO\_SET\_RELAY:

case MAV\_CMD\_DO\_SET\_SERVO:

param2 = tell\_command.alt;

param1 = tell\_command.p1;

break;

}

mavlink\_msg\_mission\_item\_send(chan,msg->sysid,

msg->compid,

packet.seq,

frame,

tell\_command.id,

current,

autocontinue,

param1,

param2,

param3,

param4,

x,

y,

z);

// update last waypoint comm stamp

waypoint\_timelast\_send = millis();

break;

}

case MAVLINK\_MSG\_ID\_MISSION\_ACK: //47

{

//send\_text\_P(SEVERITY\_LOW,PSTR("waypoint ack"));

// decode

mavlink\_mission\_ack\_t packet;

mavlink\_msg\_mission\_ack\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system,packet.target\_component)) break;

// turn off waypoint send

waypoint\_sending = false;

break;

}

case MAVLINK\_MSG\_ID\_PARAM\_REQUEST\_LIST: // 21

{

// gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("param request list"));

// decode

mavlink\_param\_request\_list\_t packet;

mavlink\_msg\_param\_request\_list\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system,packet.target\_component)) break;

// Start sending parameters - next call to ::update will kick the first one out

\_queued\_parameter = AP\_Param::first(&\_queued\_parameter\_token, &\_queued\_parameter\_type);

\_queued\_parameter\_index = 0;

\_queued\_parameter\_count = \_count\_parameters();

break;

}

case MAVLINK\_MSG\_ID\_PARAM\_REQUEST\_READ:

{

// decode

mavlink\_param\_request\_read\_t packet;

mavlink\_msg\_param\_request\_read\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system,packet.target\_component)) break;

enum ap\_var\_type p\_type;

AP\_Param \*vp;

char param\_name[AP\_MAX\_NAME\_SIZE];

if (packet.param\_index != -1) {

vp = AP\_Param::find\_by\_index(packet.param\_index, &p\_type);

if (vp == NULL) {

gcs\_send\_text\_fmt(PSTR("Unknown parameter index %d"), packet.param\_index);

break;

}

vp->copy\_name(param\_name, sizeof(param\_name), true);

} else {

vp = AP\_Param::find(packet.param\_id, &p\_type);

if (vp == NULL) {

gcs\_send\_text\_fmt(PSTR("Unknown parameter %.16s"), packet.param\_id);

break;

}

strncpy(param\_name, packet.param\_id, AP\_MAX\_NAME\_SIZE);

}

float value = vp->cast\_to\_float(p\_type);

mavlink\_msg\_param\_value\_send(

chan,

param\_name,

value,

mav\_var\_type(p\_type),

\_count\_parameters(),

packet.param\_index);

break;

}

case MAVLINK\_MSG\_ID\_MISSION\_CLEAR\_ALL: // 45

{

//send\_text\_P(SEVERITY\_LOW,PSTR("waypoint clear all"));

// decode

mavlink\_mission\_clear\_all\_t packet;

mavlink\_msg\_mission\_clear\_all\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system, packet.target\_component)) break;

// clear all waypoints

uint8\_t type = 0; // ok (0), error(1)

g.command\_total.set\_and\_save(1);

// send acknowledgement 3 times to makes sure it is received

for (int16\_t i=0; i<3; i++)

mavlink\_msg\_mission\_ack\_send(chan, msg->sysid, msg->compid, type);

break;

}

case MAVLINK\_MSG\_ID\_MISSION\_SET\_CURRENT: // 41

{

//send\_text\_P(SEVERITY\_LOW,PSTR("waypoint set current"));

// decode

mavlink\_mission\_set\_current\_t packet;

mavlink\_msg\_mission\_set\_current\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system,packet.target\_component)) break;

// set current command

change\_command(packet.seq);

mavlink\_msg\_mission\_current\_send(chan, g.command\_index);

break;

}

case MAVLINK\_MSG\_ID\_MISSION\_COUNT: // 44

{

//send\_text\_P(SEVERITY\_LOW,PSTR("waypoint count"));

// decode

mavlink\_mission\_count\_t packet;

mavlink\_msg\_mission\_count\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system,packet.target\_component)) break;

// start waypoint receiving

if (packet.count > MAX\_WAYPOINTS) {

packet.count = MAX\_WAYPOINTS;

}

g.command\_total.set\_and\_save(packet.count);

waypoint\_timelast\_receive = millis();

waypoint\_receiving = true;

waypoint\_sending = false;

waypoint\_request\_i = 0;

waypoint\_timelast\_request = 0;

break;

}

#ifdef MAVLINK\_MSG\_ID\_SET\_MAG\_OFFSETS

case MAVLINK\_MSG\_ID\_SET\_MAG\_OFFSETS:

{

mavlink\_set\_mag\_offsets\_t packet;

mavlink\_msg\_set\_mag\_offsets\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system,packet.target\_component)) break;

compass.set\_offsets(Vector3f(packet.mag\_ofs\_x, packet.mag\_ofs\_y, packet.mag\_ofs\_z));

break;

}

#endif

// XXX receive a WP from GCS and store in EEPROM

case MAVLINK\_MSG\_ID\_MISSION\_ITEM: //39

{

// decode

mavlink\_mission\_item\_t packet;

mavlink\_msg\_mission\_item\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system,packet.target\_component)) break;

// defaults

tell\_command.id = packet.command;

/\*

\* switch (packet.frame){

\*

\* case MAV\_FRAME\_MISSION:

\* case MAV\_FRAME\_GLOBAL:

\* {

\* tell\_command.lat = 1.0e7\*packet.x; // in as DD converted to \* t7

\* tell\_command.lng = 1.0e7\*packet.y; // in as DD converted to \* t7

\* tell\_command.alt = packet.z\*1.0e2; // in as m converted to cm

\* tell\_command.options = 0; // absolute altitude

\* break;

\* }

\*

\* case MAV\_FRAME\_LOCAL: // local (relative to home position)

\* {

\* tell\_command.lat = 1.0e7\*ToDeg(packet.x/

\* (radius\_of\_earth\*cos(ToRad(home.lat/1.0e7)))) + home.lat;

\* tell\_command.lng = 1.0e7\*ToDeg(packet.y/radius\_of\_earth) + home.lng;

\* tell\_command.alt = packet.z\*1.0e2;

\* tell\_command.options = MASK\_OPTIONS\_RELATIVE\_ALT;

\* break;

\* }

\* //case MAV\_FRAME\_GLOBAL\_RELATIVE\_ALT: // absolute lat/lng, relative altitude

\* default:

\* {

\* tell\_command.lat = 1.0e7 \* packet.x; // in as DD converted to \* t7

\* tell\_command.lng = 1.0e7 \* packet.y; // in as DD converted to \* t7

\* tell\_command.alt = packet.z \* 1.0e2;

\* tell\_command.options = MASK\_OPTIONS\_RELATIVE\_ALT; // store altitude relative!! Always!!

\* break;

\* }

\* }

\*/

// we only are supporting Abs position, relative Alt

tell\_command.lat = 1.0e7 \* packet.x; // in as DD converted to \* t7

tell\_command.lng = 1.0e7 \* packet.y; // in as DD converted to \* t7

tell\_command.alt = packet.z \* 1.0e2;

tell\_command.options = 1; // store altitude relative to home alt!! Always!!

switch (tell\_command.id) { // Switch to map APM command fields into MAVLink command fields

case MAV\_CMD\_NAV\_LOITER\_TURNS:

case MAV\_CMD\_DO\_SET\_HOME:

tell\_command.p1 = packet.param1;

break;

case MAV\_CMD\_NAV\_ROI:

tell\_command.p1 = packet.param1; // MAV\_ROI (aka roi mode) is held in wp's parameter but we actually do nothing with it because we only support pointing at a specific location provided by x,y and z parameters

break;

case MAV\_CMD\_CONDITION\_YAW:

tell\_command.p1 = packet.param3;

tell\_command.alt = packet.param1;

tell\_command.lat = packet.param2;

tell\_command.lng = packet.param4;

break;

case MAV\_CMD\_NAV\_TAKEOFF:

tell\_command.p1 = 0;

break;

case MAV\_CMD\_CONDITION\_CHANGE\_ALT:

tell\_command.p1 = packet.param1 \* 100;

break;

case MAV\_CMD\_NAV\_LOITER\_TIME:

tell\_command.p1 = packet.param1; // APM loiter time is in ten second increments

break;

case MAV\_CMD\_CONDITION\_DELAY:

case MAV\_CMD\_CONDITION\_DISTANCE:

tell\_command.lat = packet.param1;

break;

case MAV\_CMD\_DO\_JUMP:

tell\_command.lat = packet.param2;

tell\_command.p1 = packet.param1;

break;

case MAV\_CMD\_DO\_REPEAT\_SERVO:

tell\_command.lng = packet.param4;

case MAV\_CMD\_DO\_REPEAT\_RELAY:

case MAV\_CMD\_DO\_CHANGE\_SPEED:

tell\_command.lat = packet.param3;

tell\_command.alt = packet.param2;

tell\_command.p1 = packet.param1;

break;

case MAV\_CMD\_NAV\_WAYPOINT:

tell\_command.p1 = packet.param1;

break;

case MAV\_CMD\_DO\_SET\_PARAMETER:

case MAV\_CMD\_DO\_SET\_RELAY:

case MAV\_CMD\_DO\_SET\_SERVO:

tell\_command.alt = packet.param2;

tell\_command.p1 = packet.param1;

break;

}

if(packet.current == 2) { //current = 2 is a flag to tell us this is a "guided mode" waypoint and not for the mission

guided\_WP = tell\_command;

// add home alt if needed

if (guided\_WP.options & MASK\_OPTIONS\_RELATIVE\_ALT) {

guided\_WP.alt += home.alt;

}

set\_mode(GUIDED);

// verify we recevied the command

mavlink\_msg\_mission\_ack\_send(

chan,

msg->sysid,

msg->compid,

0);

} else if(packet.current == 3) { //current = 3 is a flag to tell us this is a alt change only

// add home alt if needed

if (tell\_command.options & MASK\_OPTIONS\_RELATIVE\_ALT) {

tell\_command.alt += home.alt;

}

set\_new\_altitude(tell\_command.alt);

// verify we recevied the command

mavlink\_msg\_mission\_ack\_send(

chan,

msg->sysid,

msg->compid,

0);

} else {

// Check if receiving waypoints (mission upload expected)

if (!waypoint\_receiving) break;

//cliSerial->printf("req: %d, seq: %d, total: %d\n", waypoint\_request\_i,packet.seq, g.command\_total.get());

// check if this is the requested waypoint

if (packet.seq != waypoint\_request\_i)

break;

if(packet.seq != 0)

set\_cmd\_with\_index(tell\_command, packet.seq);

// update waypoint receiving state machine

waypoint\_timelast\_receive = millis();

waypoint\_timelast\_request = 0;

waypoint\_request\_i++;

if (waypoint\_request\_i == (uint16\_t)g.command\_total) {

uint8\_t type = 0; // ok (0), error(1)

mavlink\_msg\_mission\_ack\_send(

chan,

msg->sysid,

msg->compid,

type);

send\_text(SEVERITY\_LOW,PSTR("flight plan received"));

waypoint\_receiving = false;

// XXX ignores waypoint radius for individual waypoints, can

// only set WP\_RADIUS parameter

}

}

break;

}

case MAVLINK\_MSG\_ID\_PARAM\_SET: // 23

{

AP\_Param \*vp;

enum ap\_var\_type var\_type;

// decode

mavlink\_param\_set\_t packet;

mavlink\_msg\_param\_set\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system, packet.target\_component))

break;

// set parameter

char key[AP\_MAX\_NAME\_SIZE+1];

strncpy(key, (char \*)packet.param\_id, AP\_MAX\_NAME\_SIZE);

key[AP\_MAX\_NAME\_SIZE] = 0;

// find the requested parameter

vp = AP\_Param::find(key, &var\_type);

if ((NULL != vp) && // exists

!isnan(packet.param\_value) && // not nan

!isinf(packet.param\_value)) { // not inf

// add a small amount before casting parameter values

// from float to integer to avoid truncating to the

// next lower integer value.

float rounding\_addition = 0.01;

// handle variables with standard type IDs

if (var\_type == AP\_PARAM\_FLOAT) {

((AP\_Float \*)vp)->set\_and\_save(packet.param\_value);

} else if (var\_type == AP\_PARAM\_INT32) {

#if LOGGING\_ENABLED == ENABLED

Log\_Write\_Data(1, ((AP\_Int32 \*)vp)->get());

#endif

if (packet.param\_value < 0) rounding\_addition = -rounding\_addition;

float v = packet.param\_value+rounding\_addition;

v = constrain(v, -2147483648.0, 2147483647.0);

((AP\_Int32 \*)vp)->set\_and\_save(v);

} else if (var\_type == AP\_PARAM\_INT16) {

#if LOGGING\_ENABLED == ENABLED

Log\_Write\_Data(3, (int32\_t)((AP\_Int16 \*)vp)->get());

#endif

if (packet.param\_value < 0) rounding\_addition = -rounding\_addition;

float v = packet.param\_value+rounding\_addition;

v = constrain(v, -32768, 32767);

((AP\_Int16 \*)vp)->set\_and\_save(v);

} else if (var\_type == AP\_PARAM\_INT8) {

#if LOGGING\_ENABLED == ENABLED

Log\_Write\_Data(4, (int32\_t)((AP\_Int8 \*)vp)->get());

#endif

if (packet.param\_value < 0) rounding\_addition = -rounding\_addition;

float v = packet.param\_value+rounding\_addition;

v = constrain(v, -128, 127);

((AP\_Int8 \*)vp)->set\_and\_save(v);

} else {

// we don't support mavlink set on this parameter

break;

}

// Report back the new value if we accepted the change

// we send the value we actually set, which could be

// different from the value sent, in case someone sent

// a fractional value to an integer type

mavlink\_msg\_param\_value\_send(

chan,

key,

vp->cast\_to\_float(var\_type),

mav\_var\_type(var\_type),

\_count\_parameters(),

-1); // XXX we don't actually know what its index is...

}

break;

} // end case

case MAVLINK\_MSG\_ID\_RC\_CHANNELS\_OVERRIDE: //70

{

// allow override of RC channel values for HIL

// or for complete GCS control of switch position

// and RC PWM values.

if(msg->sysid != g.sysid\_my\_gcs) break; // Only accept control from our gcs

mavlink\_rc\_channels\_override\_t packet;

int16\_t v[8];

mavlink\_msg\_rc\_channels\_override\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system,packet.target\_component))

break;

v[0] = packet.chan1\_raw;

v[1] = packet.chan2\_raw;

v[2] = packet.chan3\_raw;

v[3] = packet.chan4\_raw;

v[4] = packet.chan5\_raw;

v[5] = packet.chan6\_raw;

v[6] = packet.chan7\_raw;

v[7] = packet.chan8\_raw;

APM\_RC.setHIL(v);

break;

}

#if HIL\_MODE != HIL\_MODE\_DISABLED

case MAVLINK\_MSG\_ID\_HIL\_STATE:

{

mavlink\_hil\_state\_t packet;

mavlink\_msg\_hil\_state\_decode(msg, &packet);

float vel = sqrt((packet.vx \* (float)packet.vx) + (packet.vy \* (float)packet.vy));

float cog = wrap\_360(ToDeg(atan2(packet.vx, packet.vy)) \* 100);

// set gps hil sensor

g\_gps->setHIL(packet.time\_usec/1000,

packet.lat\*1.0e-7, packet.lon\*1.0e-7, packet.alt\*1.0e-3,

vel\*1.0e-2, cog\*1.0e-2, 0, 10);

if (gps\_base\_alt == 0) {

gps\_base\_alt = g\_gps->altitude;

}

current\_loc.lng = g\_gps->longitude;

current\_loc.lat = g\_gps->latitude;

current\_loc.alt = g\_gps->altitude - gps\_base\_alt;

if (!ap.home\_is\_set) {

init\_home();

}

// rad/sec

Vector3f gyros;

gyros.x = packet.rollspeed;

gyros.y = packet.pitchspeed;

gyros.z = packet.yawspeed;

// m/s/s

Vector3f accels;

accels.x = (float)packet.xacc / 1000.0;

accels.y = (float)packet.yacc / 1000.0;

accels.z = (float)packet.zacc / 1000.0;

ins.set\_gyro\_offsets(gyros);

ins.set\_accel\_offsets(accels);

// set AHRS hil sensor

ahrs.setHil(packet.roll,packet.pitch,packet.yaw,packet.rollspeed,

packet.pitchspeed,packet.yawspeed);

break;

}

#endif // HIL\_MODE != HIL\_MODE\_DISABLED

/\*

\* case MAVLINK\_MSG\_ID\_HEARTBEAT:

\* {

\* // We keep track of the last time we received a heartbeat from our GCS for failsafe purposes

\* if(msg->sysid != g.sysid\_my\_gcs) break;

\* rc\_override\_fs\_timer = millis();

\* break;

\* }

\*

\* #if HIL\_MODE != HIL\_MODE\_DISABLED

\* // This is used both as a sensor and to pass the location

\* // in HIL\_ATTITUDE mode.

\* case MAVLINK\_MSG\_ID\_GPS\_RAW:

\* {

\* // decode

\* mavlink\_gps\_raw\_t packet;

\* mavlink\_msg\_gps\_raw\_decode(msg, &packet);

\*

\* // set gps hil sensor

\* g\_gps->setHIL(packet.usec/1000,packet.lat,packet.lon,packet.alt,

\* packet.v,packet.hdg,0,0);

\* break;

\* }

\* #endif

\*/

#if HIL\_MODE == HIL\_MODE\_SENSORS

case MAVLINK\_MSG\_ID\_RAW\_IMU: // 28

{

// decode

mavlink\_raw\_imu\_t packet;

mavlink\_msg\_raw\_imu\_decode(msg, &packet);

// set imu hil sensors

// TODO: check scaling for temp/absPress

float temp = 70;

float absPress = 1;

// cliSerial->printf\_P(PSTR("accel:\t%d\t%d\t%d\n"), packet.xacc, packet.yacc, packet.zacc);

// cliSerial->printf\_P(PSTR("gyro:\t%d\t%d\t%d\n"), packet.xgyro, packet.ygyro, packet.zgyro);

// rad/sec

Vector3f gyros;

gyros.x = (float)packet.xgyro / 1000.0;

gyros.y = (float)packet.ygyro / 1000.0;

gyros.z = (float)packet.zgyro / 1000.0;

// m/s/s

Vector3f accels;

accels.x = (float)packet.xacc / 1000.0;

accels.y = (float)packet.yacc / 1000.0;

accels.z = (float)packet.zacc / 1000.0;

ins.set\_gyro\_offsets(gyros);

ins.set\_accel\_offsets(accels);

compass.setHIL(packet.xmag,packet.ymag,packet.zmag);

break;

}

case MAVLINK\_MSG\_ID\_RAW\_PRESSURE: //29

{

// decode

mavlink\_raw\_pressure\_t packet;

mavlink\_msg\_raw\_pressure\_decode(msg, &packet);

// set pressure hil sensor

// TODO: check scaling

float temp = 70;

barometer.setHIL(temp,packet.press\_diff1);

break;

}

#endif // HIL\_MODE

#if CAMERA == ENABLED

case MAVLINK\_MSG\_ID\_DIGICAM\_CONFIGURE:

{

camera.configure\_msg(msg);

break;

}

case MAVLINK\_MSG\_ID\_DIGICAM\_CONTROL:

{

camera.control\_msg(msg);

break;

}

#endif // CAMERA == ENABLED

#if MOUNT == ENABLED

case MAVLINK\_MSG\_ID\_MOUNT\_CONFIGURE:

{

camera\_mount.configure\_msg(msg);

break;

}

case MAVLINK\_MSG\_ID\_MOUNT\_CONTROL:

{

camera\_mount.control\_msg(msg);

break;

}

case MAVLINK\_MSG\_ID\_MOUNT\_STATUS:

{

camera\_mount.status\_msg(msg);

break;

}

#endif // MOUNT == ENABLED

case MAVLINK\_MSG\_ID\_RADIO:

{

mavlink\_radio\_t packet;

mavlink\_msg\_radio\_decode(msg, &packet);

// use the state of the transmit buffer in the radio to

// control the stream rate, giving us adaptive software

// flow control

if (packet.txbuf < 20 && stream\_slowdown < 100) {

// we are very low on space - slow down a lot

stream\_slowdown += 3;

} else if (packet.txbuf < 50 && stream\_slowdown < 100) {

// we are a bit low on space, slow down slightly

stream\_slowdown += 1;

} else if (packet.txbuf > 95 && stream\_slowdown > 10) {

// the buffer has plenty of space, speed up a lot

stream\_slowdown -= 2;

} else if (packet.txbuf > 90 && stream\_slowdown != 0) {

// the buffer has enough space, speed up a bit

stream\_slowdown--;

}

break;

}

#if AP\_LIMITS == ENABLED

// receive an AP\_Limits fence point from GCS and store in EEPROM

// receive a fence point from GCS and store in EEPROM

case MAVLINK\_MSG\_ID\_FENCE\_POINT: {

mavlink\_fence\_point\_t packet;

mavlink\_msg\_fence\_point\_decode(msg, &packet);

if (packet.count != geofence\_limit.fence\_total()) {

send\_text(SEVERITY\_LOW,PSTR("bad fence point"));

} else {

Vector2l point;

point.x = packet.lat\*1.0e7;

point.y = packet.lng\*1.0e7;

geofence\_limit.set\_fence\_point\_with\_index(point, packet.idx);

}

break;

}

// send a fence point to GCS

case MAVLINK\_MSG\_ID\_FENCE\_FETCH\_POINT: {

mavlink\_fence\_fetch\_point\_t packet;

mavlink\_msg\_fence\_fetch\_point\_decode(msg, &packet);

if (mavlink\_check\_target(packet.target\_system, packet.target\_component))

break;

if (packet.idx >= geofence\_limit.fence\_total()) {

send\_text(SEVERITY\_LOW,PSTR("bad fence point"));

} else {

Vector2l point = geofence\_limit.get\_fence\_point\_with\_index(packet.idx);

mavlink\_msg\_fence\_point\_send(chan, 0, 0, packet.idx, geofence\_limit.fence\_total(),

point.x\*1.0e-7, point.y\*1.0e-7);

}

break;

}

#endif // AP\_LIMITS ENABLED

} // end switch

} // end handle mavlink

uint16\_t

GCS\_MAVLINK::\_count\_parameters()

{

// if we haven't cached the parameter count yet...

if (0 == \_parameter\_count) {

AP\_Param \*vp;

AP\_Param::ParamToken token;

vp = AP\_Param::first(&token, NULL);

do {

\_parameter\_count++;

} while (NULL != (vp = AP\_Param::next\_scalar(&token, NULL)));

}

return \_parameter\_count;

}

/\*\*

\* @brief Send the next pending parameter, called from deferred message

\* handling code

\*/

void

GCS\_MAVLINK::queued\_param\_send()

{

// Check to see if we are sending parameters

if (NULL == \_queued\_parameter) return;

AP\_Param \*vp;

float value;

// copy the current parameter and prepare to move to the next

vp = \_queued\_parameter;

// if the parameter can be cast to float, report it here and break out of the loop

value = vp->cast\_to\_float(\_queued\_parameter\_type);

char param\_name[AP\_MAX\_NAME\_SIZE];

vp->copy\_name\_token(&\_queued\_parameter\_token, param\_name, sizeof(param\_name), true);

mavlink\_msg\_param\_value\_send(

chan,

param\_name,

value,

mav\_var\_type(\_queued\_parameter\_type),

\_queued\_parameter\_count,

\_queued\_parameter\_index);

\_queued\_parameter = AP\_Param::next\_scalar(&\_queued\_parameter\_token, &\_queued\_parameter\_type);

\_queued\_parameter\_index++;

}

/\*\*

\* @brief Send the next pending waypoint, called from deferred message

\* handling code

\*/

void

GCS\_MAVLINK::queued\_waypoint\_send()

{

if (waypoint\_receiving &&

waypoint\_request\_i < (unsigned)g.command\_total) {

mavlink\_msg\_mission\_request\_send(

chan,

waypoint\_dest\_sysid,

waypoint\_dest\_compid,

waypoint\_request\_i);

}

}

/\*

\* a delay() callback that processes MAVLink packets. We set this as the

\* callback in long running library initialisation routines to allow

\* MAVLink to process packets while waiting for the initialisation to

\* complete

\*/

static void mavlink\_delay(unsigned long t)

{

uint32\_t tstart;

if (in\_mavlink\_delay) {

// this should never happen, but let's not tempt fate by

// letting the stack grow too much

delay(t);

return;

}

in\_mavlink\_delay = true;

tstart = millis();

do {

if( !gcs\_check() ) {

// delay 0.1 millisecond if the gcs\_check didn't do anything

// the while below will extend this time

delayMicroseconds(100);

}

#if USB\_MUX\_PIN > 0

check\_usb\_mux();

#endif

} while (millis() - tstart < t);

in\_mavlink\_delay = false;

}

/\*

\* send a message on both GCS links

\*/

static void gcs\_send\_message(enum ap\_message id)

{

gcs0.send\_message(id);

if (gcs3.initialised) {

gcs3.send\_message(id);

}

}

/\*

\* send data streams in the given rate range on both links

\*/

static void gcs\_data\_stream\_send(void)

{

gcs0.data\_stream\_send();

if (gcs3.initialised) {

gcs3.data\_stream\_send();

}

}

/\*

\* look for incoming commands on the GCS links

\*/

static void gcs\_update(void)

{

gcs0.update();

if (gcs3.initialised) {

gcs3.update();

}

}

static void gcs\_send\_text(gcs\_severity severity, const char \*str)

{

gcs0.send\_text(severity, str);

if (gcs3.initialised) {

gcs3.send\_text(severity, str);

}

}

static void gcs\_send\_text\_P(gcs\_severity severity, const prog\_char\_t \*str)

{

gcs0.send\_text(severity, str);

if (gcs3.initialised) {

gcs3.send\_text(severity, str);

}

}

/\*

\* send a low priority formatted message to the GCS

\* only one fits in the queue, so if you send more than one before the

\* last one gets into the serial buffer then the old one will be lost

\*/

static void gcs\_send\_text\_fmt(const prog\_char\_t \*fmt, ...)

{

char fmtstr[40];

va\_list arg\_list;

uint8\_t i;

for (i=0; i<sizeof(fmtstr)-1; i++) {

fmtstr[i] = pgm\_read\_byte((const prog\_char \*)(fmt++));

if (fmtstr[i] == 0) break;

}

fmtstr[i] = 0;

pending\_status.severity = (uint8\_t)SEVERITY\_LOW;

va\_start(arg\_list, fmt);

vsnprintf((char \*)pending\_status.text, sizeof(pending\_status.text), fmtstr, arg\_list);

va\_end(arg\_list);

mavlink\_send\_message(MAVLINK\_COMM\_0, MSG\_STATUSTEXT, 0);

if (gcs3.initialised) {

mavlink\_send\_message(MAVLINK\_COMM\_1, MSG\_STATUSTEXT, 0);

}

}

## Log

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

#if LOGGING\_ENABLED == ENABLED

// Code to Write and Read packets from DataFlash log memory

// Code to interact with the user to dump or erase logs

#define HEAD\_BYTE1 0xA3 // Decimal 163

#define HEAD\_BYTE2 0x95 // Decimal 149

#define END\_BYTE 0xBA // Decimal 186

// These are function definitions so the Menu can be constructed before the functions

// are defined below. Order matters to the compiler.

static bool print\_log\_menu(void);

static int8\_t dump\_log(uint8\_t argc, const Menu::arg \*argv);

static int8\_t erase\_logs(uint8\_t argc, const Menu::arg \*argv);

static int8\_t select\_logs(uint8\_t argc, const Menu::arg \*argv);

// This is the help function

// PSTR is an AVR macro to read strings from flash memory

// printf\_P is a version of print\_f that reads from flash memory

//static int8\_t help\_log(uint8\_t argc, const Menu::arg \*argv)

/\*{

\* cliSerial->printf\_P(PSTR("\n"

\* "Commands:\n"

\* " dump <n>"

\* " erase (all logs)\n"

\* " enable <name> | all\n"

\* " disable <name> | all\n"

\* "\n"));

\* return 0;

\* }\*/

// Creates a constant array of structs representing menu options

// and stores them in Flash memory, not RAM.

// User enters the string in the console to call the functions on the right.

// See class Menu in AP\_Coommon for implementation details

const struct Menu::command log\_menu\_commands[] PROGMEM = {

{"dump", dump\_log},

{"erase", erase\_logs},

{"enable", select\_logs},

{"disable", select\_logs}

};

static int32\_t get\_int(float f)

{

float\_int.float\_value = f;

return float\_int.int\_value;

}

static float get\_float(int32\_t i)

{

float\_int.int\_value = i;

return float\_int.float\_value;

}

// A Macro to create the Menu

MENU2(log\_menu, "Log", log\_menu\_commands, print\_log\_menu);

static bool

print\_log\_menu(void)

{

int16\_t log\_start;

int16\_t log\_end;

int16\_t temp;

int16\_t last\_log\_num = DataFlash.find\_last\_log();

uint16\_t num\_logs = DataFlash.get\_num\_logs();

cliSerial->printf\_P(PSTR("logs enabled: "));

if (0 == g.log\_bitmask) {

cliSerial->printf\_P(PSTR("none"));

}else{

if (g.log\_bitmask & MASK\_LOG\_ATTITUDE\_FAST) cliSerial->printf\_P(PSTR(" ATTITUDE\_FAST"));

if (g.log\_bitmask & MASK\_LOG\_ATTITUDE\_MED) cliSerial->printf\_P(PSTR(" ATTITUDE\_MED"));

if (g.log\_bitmask & MASK\_LOG\_GPS) cliSerial->printf\_P(PSTR(" GPS"));

if (g.log\_bitmask & MASK\_LOG\_PM) cliSerial->printf\_P(PSTR(" PM"));

if (g.log\_bitmask & MASK\_LOG\_CTUN) cliSerial->printf\_P(PSTR(" CTUN"));

if (g.log\_bitmask & MASK\_LOG\_NTUN) cliSerial->printf\_P(PSTR(" NTUN"));

if (g.log\_bitmask & MASK\_LOG\_RAW) cliSerial->printf\_P(PSTR(" RAW"));

if (g.log\_bitmask & MASK\_LOG\_CMD) cliSerial->printf\_P(PSTR(" CMD"));

if (g.log\_bitmask & MASK\_LOG\_CUR) cliSerial->printf\_P(PSTR(" CURRENT"));

if (g.log\_bitmask & MASK\_LOG\_MOTORS) cliSerial->printf\_P(PSTR(" MOTORS"));

if (g.log\_bitmask & MASK\_LOG\_OPTFLOW) cliSerial->printf\_P(PSTR(" OPTFLOW"));

if (g.log\_bitmask & MASK\_LOG\_PID) cliSerial->printf\_P(PSTR(" PID"));

if (g.log\_bitmask & MASK\_LOG\_ITERM) cliSerial->printf\_P(PSTR(" ITERM"));

if (g.log\_bitmask & MASK\_LOG\_INAV) cliSerial->printf\_P(PSTR(" INAV"));

if (g.log\_bitmask & MASK\_LOG\_CAMERA) cliSerial->printf\_P(PSTR(" CAMERA"));

}

cliSerial->println();

if (num\_logs == 0) {

cliSerial->printf\_P(PSTR("\nNo logs\n\n"));

}else{

cliSerial->printf\_P(PSTR("\n%u logs\n"), (unsigned)num\_logs);

for(int16\_t i=num\_logs; i>=1; i--) {

int16\_t last\_log\_start = log\_start, last\_log\_end = log\_end;

temp = last\_log\_num-i+1;

DataFlash.get\_log\_boundaries(temp, log\_start, log\_end);

cliSerial->printf\_P(PSTR("Log %d, start %d, end %d\n"), (int)temp, (int)log\_start, (int)log\_end);

if (last\_log\_start == log\_start && last\_log\_end == log\_end) {

// we are printing bogus logs

break;

}

}

cliSerial->println();

}

return(true);

}

static int8\_t

dump\_log(uint8\_t argc, const Menu::arg \*argv)

{

int16\_t dump\_log;

int16\_t dump\_log\_start;

int16\_t dump\_log\_end;

int16\_t last\_log\_num;

// check that the requested log number can be read

dump\_log = argv[1].i;

last\_log\_num = DataFlash.find\_last\_log();

if (dump\_log == -2) {

for(uint16\_t count=1; count<=DataFlash.df\_NumPages; count++) {

DataFlash.StartRead(count);

cliSerial->printf\_P(PSTR("DF page, log file #, log page: %d,\t"), (int)count);

cliSerial->printf\_P(PSTR("%d,\t"), (int)DataFlash.GetFileNumber());

cliSerial->printf\_P(PSTR("%d\n"), (int)DataFlash.GetFilePage());

}

return(-1);

} else if (dump\_log <= 0) {

cliSerial->printf\_P(PSTR("dumping all\n"));

Log\_Read(1, DataFlash.df\_NumPages);

return(-1);

} else if ((argc != 2) || (dump\_log <= (last\_log\_num - DataFlash.get\_num\_logs())) || (dump\_log > last\_log\_num)) {

cliSerial->printf\_P(PSTR("bad log number\n"));

return(-1);

}

DataFlash.get\_log\_boundaries(dump\_log, dump\_log\_start, dump\_log\_end);

/\*cliSerial->printf\_P(PSTR("Dumping Log number %d, start %d, end %d\n"),

\* dump\_log,

\* dump\_log\_start,

\* dump\_log\_end);

\*/

Log\_Read(dump\_log\_start, dump\_log\_end);

//cliSerial->printf\_P(PSTR("Done\n"));

return (0);

}

static void do\_erase\_logs(void)

{

gcs\_send\_text\_P(SEVERITY\_LOW, PSTR("Erasing logs\n"));

DataFlash.EraseAll(mavlink\_delay);

gcs\_send\_text\_P(SEVERITY\_LOW, PSTR("Log erase complete\n"));

}

static int8\_t

erase\_logs(uint8\_t argc, const Menu::arg \*argv)

{

in\_mavlink\_delay = true;

do\_erase\_logs();

in\_mavlink\_delay = false;

return 0;

}

static int8\_t

select\_logs(uint8\_t argc, const Menu::arg \*argv)

{

uint16\_t bits;

if (argc != 2) {

cliSerial->printf\_P(PSTR("missing log type\n"));

return(-1);

}

bits = 0;

// Macro to make the following code a bit easier on the eye.

// Pass it the capitalised name of the log option, as defined

// in defines.h but without the LOG\_ prefix. It will check for

// that name as the argument to the command, and set the bit in

// bits accordingly.

//

if (!strcasecmp\_P(argv[1].str, PSTR("all"))) {

bits = ~0;

} else {

#define TARG(\_s) if (!strcasecmp\_P(argv[1].str, PSTR(# \_s))) bits |= MASK\_LOG\_ ## \_s

TARG(ATTITUDE\_FAST);

TARG(ATTITUDE\_MED);

TARG(GPS);

TARG(PM);

TARG(CTUN);

TARG(NTUN);

TARG(MODE);

TARG(RAW);

TARG(CMD);

TARG(CUR);

TARG(MOTORS);

TARG(OPTFLOW);

TARG(PID);

TARG(ITERM);

TARG(INAV);

TARG(CAMERA);

#undef TARG

}

if (!strcasecmp\_P(argv[0].str, PSTR("enable"))) {

g.log\_bitmask.set\_and\_save(g.log\_bitmask | bits);

}else{

g.log\_bitmask.set\_and\_save(g.log\_bitmask & ~bits);

}

return(0);

}

static int8\_t

process\_logs(uint8\_t argc, const Menu::arg \*argv)

{

log\_menu.run();

return 0;

}

// print\_latlon - prints an latitude or longitude value held in an int32\_t

// probably this should be moved to AP\_Common

void print\_latlon(BetterStream \*s, int32\_t lat\_or\_lon)

{

int32\_t dec\_portion, frac\_portion;

int32\_t abs\_lat\_or\_lon = labs(lat\_or\_lon);

// extract decimal portion (special handling of negative numbers to ensure we round towards zero)

dec\_portion = abs\_lat\_or\_lon / T7;

// extract fractional portion

frac\_portion = abs\_lat\_or\_lon - dec\_portion\*T7;

// print output including the minus sign

if( lat\_or\_lon < 0 ) {

s->printf\_P(PSTR("-"));

}

s->printf\_P(PSTR("%ld.%07ld"),(long)dec\_portion,(long)frac\_portion);

}

// Write an GPS packet. Total length : 31 bytes

static void Log\_Write\_GPS()

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_GPS\_MSG);

DataFlash.WriteLong(g\_gps->time); // 1

DataFlash.WriteByte(g\_gps->num\_sats); // 2

DataFlash.WriteLong(g\_gps->latitude); // 3

DataFlash.WriteLong(g\_gps->longitude); // 4

DataFlash.WriteLong(current\_loc.alt); // 5

DataFlash.WriteLong(g\_gps->altitude); // 6

DataFlash.WriteInt(g\_gps->ground\_speed); // 7

DataFlash.WriteLong(g\_gps->ground\_course); // 8

DataFlash.WriteByte(END\_BYTE);

}

// Read a GPS packet

static void Log\_Read\_GPS()

{

int32\_t temp1 = DataFlash.ReadLong(); // 1 time

int8\_t temp2 = DataFlash.ReadByte(); // 2 sats

int32\_t temp3 = DataFlash.ReadLong(); // 3 lat

int32\_t temp4 = DataFlash.ReadLong(); // 4 lon

float temp5 = DataFlash.ReadLong() / 100.0; // 5 sensor alt

float temp6 = DataFlash.ReadLong() / 100.0; // 6 gps alt

int16\_t temp7 = DataFlash.ReadInt(); // 7 ground speed

int32\_t temp8 = DataFlash.ReadLong(); // 8 ground course

// 1 2 3 4 5 6 7 8

cliSerial->printf\_P(PSTR("GPS, %ld, %d, "),

(long)temp1, // 1 time

(int)temp2); // 2 sats

print\_latlon(&Serial, temp3);

cliSerial->print\_P(PSTR(", "));

print\_latlon(&Serial, temp4);

cliSerial->printf\_P(PSTR(", %4.4f, %4.4f, %d, %ld\n"),

temp5, // 5 gps alt

temp6, // 6 sensor alt

(int)temp7, // 7 ground speed

(long)temp8); // 8 ground course

}

static void Log\_Write\_Raw()

{

Vector3f gyro = ins.get\_gyro();

Vector3f accel = ins.get\_accel();

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_RAW\_MSG);

DataFlash.WriteLong(get\_int(gyro.x));

DataFlash.WriteLong(get\_int(gyro.y));

DataFlash.WriteLong(get\_int(gyro.z));

DataFlash.WriteLong(get\_int(accel.x));

DataFlash.WriteLong(get\_int(accel.y));

DataFlash.WriteLong(get\_int(accel.z));

DataFlash.WriteByte(END\_BYTE);

/\*

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_RAW\_MSG);

DataFlash.WriteLong(get\_int(ahrs.\_omega\_I.x));

DataFlash.WriteLong(get\_int(ahrs.\_omega\_I.y));

DataFlash.WriteByte(END\_BYTE);

\*/

}

// Read a raw accel/gyro packet

static void Log\_Read\_Raw()

{

float logvar;

cliSerial->printf\_P(PSTR("RAW,"));

for (int16\_t y = 0; y < 6; y++) {

logvar = get\_float(DataFlash.ReadLong());

cliSerial->print(logvar);

cliSerial->print\_P(PSTR(", "));

}

cliSerial->println\_P(PSTR(" "));

/\*

float temp1 = get\_float(DataFlash.ReadLong());

float temp2 = get\_float(DataFlash.ReadLong());

cliSerial->printf\_P(PSTR("RAW, %4.4f, %4.4f\n"),

temp1,

temp2);

\*/

}

// Write an Current data packet. Total length : 16 bytes

static void Log\_Write\_Current()

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_CURRENT\_MSG);

DataFlash.WriteInt(g.rc\_3.control\_in); // 1

DataFlash.WriteLong(throttle\_integrator); // 2

DataFlash.WriteInt(battery\_voltage1 \* 100.0); // 3

DataFlash.WriteInt(current\_amps1 \* 100.0); // 4

DataFlash.WriteInt(current\_total1); // 5

DataFlash.WriteByte(END\_BYTE);

}

// Read a Current packet

static void Log\_Read\_Current()

{

int16\_t temp1 = DataFlash.ReadInt(); // 1

int32\_t temp2 = DataFlash.ReadLong(); // 2

float temp3 = DataFlash.ReadInt() / 100.f; // 3

float temp4 = DataFlash.ReadInt() / 100.f; // 4

int16\_t temp5 = DataFlash.ReadInt(); // 5

// 1 2 3 4 5

cliSerial->printf\_P(PSTR("CURR, %d, %ld, %4.4f, %4.4f, %d\n"),

(int)temp1,

(long)temp2,

temp3,

temp4,

(int)temp5);

}

// Write an Motors packet. Total length : 12 ~ 20 bytes

static void Log\_Write\_Motors()

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_MOTORS\_MSG);

#if FRAME\_CONFIG == TRI\_FRAME

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_1]); //1

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_2]); //2

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_4]); //3

DataFlash.WriteInt(g.rc\_4.radio\_out); //4

#elif FRAME\_CONFIG == HEXA\_FRAME

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_1]); //1

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_2]); //2

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_3]); //3

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_4]); //4

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_5]); //5

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_6]); //6

#elif FRAME\_CONFIG == Y6\_FRAME

//left

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_2]); //1

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_3]); //2

//right

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_5]); //3

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_1]); //4

//back

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_6]); //5

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_4]); //6

#elif FRAME\_CONFIG == OCTA\_FRAME || FRAME\_CONFIG == OCTA\_QUAD\_FRAME

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_1]); //1

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_2]); //2

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_3]); //3

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_4]); //4

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_5]); //5

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_6]); //6

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_7]); //7

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_8]); //8

#elif FRAME\_CONFIG == HELI\_FRAME

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_1]); //1

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_2]); //2

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_3]); //3

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_4]); //4

DataFlash.WriteInt(motors.ext\_gyro\_gain); //5

#else // quads

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_1]); //1

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_2]); //2

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_3]); //3

DataFlash.WriteInt(motors.motor\_out[AP\_MOTORS\_MOT\_4]); //4

#endif

DataFlash.WriteByte(END\_BYTE);

}

// Read a Motors packet.

static void Log\_Read\_Motors()

{

#if FRAME\_CONFIG == HEXA\_FRAME || FRAME\_CONFIG == Y6\_FRAME

int16\_t temp1 = DataFlash.ReadInt(); // 1

int16\_t temp2 = DataFlash.ReadInt(); // 2

int16\_t temp3 = DataFlash.ReadInt(); // 3

int16\_t temp4 = DataFlash.ReadInt(); // 4

int16\_t temp5 = DataFlash.ReadInt(); // 5

int16\_t temp6 = DataFlash.ReadInt(); // 6

// 1 2 3 4 5 6

cliSerial->printf\_P(PSTR("MOT, %d, %d, %d, %d, %d, %d\n"),

(int)temp1, //1

(int)temp2, //2

(int)temp3, //3

(int)temp4, //4

(int)temp5, //5

(int)temp6); //6

#elif FRAME\_CONFIG == OCTA\_FRAME || FRAME\_CONFIG == OCTA\_QUAD\_FRAME

int16\_t temp1 = DataFlash.ReadInt(); // 1

int16\_t temp2 = DataFlash.ReadInt(); // 2

int16\_t temp3 = DataFlash.ReadInt(); // 3

int16\_t temp4 = DataFlash.ReadInt(); // 4

int16\_t temp5 = DataFlash.ReadInt(); // 5

int16\_t temp6 = DataFlash.ReadInt(); // 6

int16\_t temp7 = DataFlash.ReadInt(); // 7

int16\_t temp8 = DataFlash.ReadInt(); // 8

// 1 2 3 4 5 6 7 8

cliSerial->printf\_P(PSTR("MOT, %d, %d, %d, %d, %d, %d, %d, %d\n"),

(int)temp1, //1

(int)temp2, //2

(int)temp3, //3

(int)temp4, //4

(int)temp5, //5

(int)temp6, //6

(int)temp7, //7

(int)temp8); //8

#elif FRAME\_CONFIG == HELI\_FRAME

int16\_t temp1 = DataFlash.ReadInt(); // 1

int16\_t temp2 = DataFlash.ReadInt(); // 2

int16\_t temp3 = DataFlash.ReadInt(); // 3

int16\_t temp4 = DataFlash.ReadInt(); // 4

int16\_t temp5 = DataFlash.ReadInt(); // 5

// 1 2 3 4 5

cliSerial->printf\_P(PSTR("MOT, %d, %d, %d, %d, %d\n"),

(int)temp1, //1

(int)temp2, //2

(int)temp3, //3

(int)temp4, //4

(int)temp5); //5

#else // quads, TRIs

int16\_t temp1 = DataFlash.ReadInt(); // 1

int16\_t temp2 = DataFlash.ReadInt(); // 2

int16\_t temp3 = DataFlash.ReadInt(); // 3

int16\_t temp4 = DataFlash.ReadInt(); // 4

// 1 2 3 4

cliSerial->printf\_P(PSTR("MOT, %d, %d, %d, %d\n"),

(int)temp1, //1

(int)temp2, //2

(int)temp3, //3

(int)temp4); //4;

#endif

}

// Write an optical flow packet. Total length : 30 bytes

static void Log\_Write\_Optflow()

{

#if OPTFLOW == ENABLED

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_OPTFLOW\_MSG);

DataFlash.WriteInt((int)optflow.dx);

DataFlash.WriteInt((int)optflow.dy);

DataFlash.WriteInt((int)optflow.surface\_quality);

DataFlash.WriteInt((int)optflow.x\_cm);

DataFlash.WriteInt((int)optflow.y\_cm);

DataFlash.WriteLong(optflow.vlat); //optflow\_offset.lat + optflow.lat);

DataFlash.WriteLong(optflow.vlon); //optflow\_offset.lng + optflow.lng);

DataFlash.WriteLong(of\_roll);

DataFlash.WriteLong(of\_pitch);

DataFlash.WriteByte(END\_BYTE);

#endif // OPTFLOW == ENABLED

}

// Read an optical flow packet.

static void Log\_Read\_Optflow()

{

int16\_t temp1 = DataFlash.ReadInt(); // 1

int16\_t temp2 = DataFlash.ReadInt(); // 2

int16\_t temp3 = DataFlash.ReadInt(); // 3

int16\_t temp4 = DataFlash.ReadInt(); // 4

int16\_t temp5 = DataFlash.ReadInt(); // 5

float temp6 = DataFlash.ReadLong(); // 6

float temp7 = DataFlash.ReadLong(); // 7

int32\_t temp8 = DataFlash.ReadLong(); // 8

int32\_t temp9 = DataFlash.ReadLong(); // 9

cliSerial->printf\_P(PSTR("OF, %d, %d, %d, %d, %d, %4.7f, %4.7f, %ld, %ld\n"),

(int)temp1,

(int)temp2,

(int)temp3,

(int)temp4,

(int)temp5,

temp6,

temp7,

(long)temp8,

(long)temp9);

}

// Write an Nav Tuning packet. Total length : 24 bytes

static void Log\_Write\_Nav\_Tuning()

{

//Matrix3f tempmat = dcm.get\_dcm\_matrix();

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_NAV\_TUNING\_MSG);

DataFlash.WriteInt(wp\_distance); // 1

DataFlash.WriteInt(wp\_bearing/100); // 2

DataFlash.WriteInt(long\_error); // 3

DataFlash.WriteInt(lat\_error); // 4

DataFlash.WriteInt(nav\_pitch); // 5

DataFlash.WriteInt(nav\_roll); // 6

DataFlash.WriteInt(lon\_speed); // 7

DataFlash.WriteInt(lat\_speed); // 8

DataFlash.WriteByte(END\_BYTE);

}

// Read a Nav Tuning packet.

static void Log\_Read\_Nav\_Tuning()

{

int16\_t temp;

cliSerial->printf\_P(PSTR("NTUN, "));

for(int8\_t i = 1; i < 8; i++ ) {

temp = DataFlash.ReadInt();

cliSerial->printf\_P(PSTR("%d, "), (int)temp);

}

// read 8

temp = DataFlash.ReadInt();

cliSerial->printf\_P(PSTR("%d\n"), (int)temp);

}

// Write a control tuning packet. Total length : 26 bytes

static void Log\_Write\_Control\_Tuning()

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_CONTROL\_TUNING\_MSG);

DataFlash.WriteInt(g.rc\_3.control\_in); // 1

DataFlash.WriteInt(sonar\_alt); // 2

DataFlash.WriteInt(baro\_alt); // 3

DataFlash.WriteInt(next\_WP.alt); // 4

DataFlash.WriteInt(nav\_throttle); // 5

DataFlash.WriteInt(angle\_boost); // 6

DataFlash.WriteInt(climb\_rate); // 7

DataFlash.WriteInt(g.rc\_3.servo\_out); // 8

DataFlash.WriteInt(desired\_climb\_rate); // 9

DataFlash.WriteByte(END\_BYTE);

}

// Read an control tuning packet

static void Log\_Read\_Control\_Tuning()

{

int16\_t temp;

cliSerial->printf\_P(PSTR("CTUN, "));

for(uint8\_t i = 1; i < 9; i++ ) {

temp = DataFlash.ReadInt();

cliSerial->printf\_P(PSTR("%d, "), (int)temp);

}

// read 9

temp = DataFlash.ReadInt();

cliSerial->printf\_P(PSTR("%d\n"), (int)temp);

}

static void Log\_Write\_Iterm()

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_ITERM\_MSG);

DataFlash.WriteInt((int16\_t)g.pi\_stabilize\_roll.get\_integrator()); // 1

DataFlash.WriteInt((int16\_t)g.pi\_stabilize\_pitch.get\_integrator()); // 2

DataFlash.WriteInt((int16\_t)g.pi\_stabilize\_yaw.get\_integrator()); // 3

DataFlash.WriteInt((int16\_t)g.pid\_rate\_roll.get\_integrator()); // 4

DataFlash.WriteInt((int16\_t)g.pid\_rate\_pitch.get\_integrator()); // 5

DataFlash.WriteInt((int16\_t)g.pid\_rate\_yaw.get\_integrator()); // 6

DataFlash.WriteInt((int16\_t)g.pid\_nav\_lat.get\_integrator()); // 7

DataFlash.WriteInt((int16\_t)g.pid\_nav\_lon.get\_integrator()); // 8

DataFlash.WriteInt((int16\_t)g.pid\_loiter\_rate\_lat.get\_integrator()); // 9

DataFlash.WriteInt((int16\_t)g.pid\_loiter\_rate\_lon.get\_integrator()); // 10

DataFlash.WriteInt((int16\_t)g.pid\_throttle.get\_integrator()); // 11

DataFlash.WriteInt(g.throttle\_cruise); // 12

DataFlash.WriteByte(END\_BYTE);

}

// Read an control tuning packet

static void Log\_Read\_Iterm()

{

int16\_t temp;

cliSerial->printf\_P(PSTR("ITERM, "));

for(uint8\_t i = 1; i < 12; i++ ) {

temp = DataFlash.ReadInt();

cliSerial->printf\_P(PSTR("%d, "), (int)temp);

}

// read 12

temp = DataFlash.ReadInt();

cliSerial->println((int)temp);

}

// Write a performance monitoring packet. Total length : 11 bytes

static void Log\_Write\_Performance()

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_PERFORMANCE\_MSG);

DataFlash.WriteByte(ahrs.renorm\_range\_count); //1

DataFlash.WriteByte(ahrs.renorm\_blowup\_count); //2

DataFlash.WriteByte(gps\_fix\_count); //3

DataFlash.WriteInt(perf\_info\_get\_num\_long\_running()); //4 - number of long running loops

DataFlash.WriteInt(perf\_info\_get\_num\_loops()); //5 - total number of loops

DataFlash.WriteLong(perf\_info\_get\_max\_time()); //6 - time of longest running loop

DataFlash.WriteByte(END\_BYTE);

}

// Read a performance packet

static void Log\_Read\_Performance()

{

int8\_t temp1 = DataFlash.ReadByte();

int8\_t temp2 = DataFlash.ReadByte();

int8\_t temp3 = DataFlash.ReadByte();

uint16\_t temp4 = DataFlash.ReadInt();

uint16\_t temp5 = DataFlash.ReadInt();

uint32\_t temp6 = DataFlash.ReadLong();

// 1 2 3 4 5 6

cliSerial->printf\_P(PSTR("PM, %d, %d, %d, %u, %u, %lu\n"),

(int)temp1,

(int)temp2,

(int)temp3,

(unsigned int)temp4,

(unsigned int)temp5,

(unsigned long)temp6);

}

// Write a command processing packet. Total length : 21 bytes

static void Log\_Write\_Cmd(byte num, struct Location \*wp)

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_CMD\_MSG);

DataFlash.WriteByte(g.command\_total); // 1

DataFlash.WriteByte(num); // 2

DataFlash.WriteByte(wp->id); // 3

DataFlash.WriteByte(wp->options); // 4

DataFlash.WriteByte(wp->p1); // 5

DataFlash.WriteLong(wp->alt); // 6

DataFlash.WriteLong(wp->lat); // 7

DataFlash.WriteLong(wp->lng); // 8

DataFlash.WriteByte(END\_BYTE);

}

//CMD, 3, 0, 16, 8, 1, 800, 340440192, -1180692736

// Read a command processing packet

static void Log\_Read\_Cmd()

{

int8\_t temp1 = DataFlash.ReadByte();

int8\_t temp2 = DataFlash.ReadByte();

int8\_t temp3 = DataFlash.ReadByte();

int8\_t temp4 = DataFlash.ReadByte();

int8\_t temp5 = DataFlash.ReadByte();

int32\_t temp6 = DataFlash.ReadLong();

int32\_t temp7 = DataFlash.ReadLong();

int32\_t temp8 = DataFlash.ReadLong();

// 1 2 3 4 5 6 7 8

cliSerial->printf\_P(PSTR( "CMD, %d, %d, %d, %d, %d, %ld, %ld, %ld\n"),

(int)temp1,

(int)temp2,

(int)temp3,

(int)temp4,

(int)temp5,

(long)temp6,

(long)temp7,

(long)temp8);

}

// Write an attitude packet. Total length : 16 bytes

static void Log\_Write\_Attitude()

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_ATTITUDE\_MSG);

DataFlash.WriteInt(control\_roll); // 1

DataFlash.WriteInt((int16\_t)ahrs.roll\_sensor); // 2

DataFlash.WriteInt(control\_pitch); // 3

DataFlash.WriteInt((int16\_t)ahrs.pitch\_sensor); // 4

DataFlash.WriteInt(g.rc\_4.control\_in); // 5

DataFlash.WriteInt((uint16\_t)ahrs.yaw\_sensor); // 6

DataFlash.WriteInt((uint16\_t)nav\_yaw); // 7 (this used to be compass.heading)

DataFlash.WriteByte(END\_BYTE);

}

// Read an attitude packet

static void Log\_Read\_Attitude()

{

int16\_t temp1 = DataFlash.ReadInt();

int16\_t temp2 = DataFlash.ReadInt();

int16\_t temp3 = DataFlash.ReadInt();

int16\_t temp4 = DataFlash.ReadInt();

int16\_t temp5 = DataFlash.ReadInt();

uint16\_t temp6 = DataFlash.ReadInt();

uint16\_t temp7 = DataFlash.ReadInt();

// 1 2 3 4 5 6 7 8 9

cliSerial->printf\_P(PSTR("ATT, %d, %d, %d, %d, %d, %u, %u\n"),

(int)temp1,

(int)temp2,

(int)temp3,

(int)temp4,

(int)temp5,

(unsigned)temp6,

(unsigned)temp7);

}

// Write an INAV packet. Total length : 52 Bytes

static void Log\_Write\_INAV()

{

#if INERTIAL\_NAV\_XY == ENABLED || INERTIAL\_NAV\_Z == ENABLED

Vector3f accel\_corr = inertial\_nav.accel\_correction.get();

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_INAV\_MSG);

DataFlash.WriteInt((int16\_t)baro\_alt); // 1 barometer altitude

DataFlash.WriteInt((int16\_t)inertial\_nav.get\_altitude()); // 2 accel + baro filtered altitude

DataFlash.WriteInt((int16\_t)baro\_rate); // 3 barometer based climb rate

DataFlash.WriteInt((int16\_t)inertial\_nav.get\_velocity\_z()); // 4 accel + baro based climb rate

DataFlash.WriteLong(get\_int(accel\_corr.x)); // 5 accel correction x-axis

DataFlash.WriteLong(get\_int(accel\_corr.y)); // 6 accel correction y-axis

DataFlash.WriteLong(get\_int(accel\_corr.z)); // 7 accel correction z-axis

DataFlash.WriteLong(get\_int(inertial\_nav.accel\_correction\_ef.z)); // 8 accel correction earth frame

DataFlash.WriteLong(g\_gps->latitude-home.lat); // 9 lat from home

DataFlash.WriteLong(g\_gps->longitude-home.lng); // 10 lon from home

DataFlash.WriteLong(get\_int(inertial\_nav.get\_latitude\_diff())); // 11 accel based lat from home

DataFlash.WriteLong(get\_int(inertial\_nav.get\_longitude\_diff())); // 12 accel based lon from home

DataFlash.WriteLong(get\_int(inertial\_nav.get\_latitude\_velocity())); // 13 accel based lat velocity

DataFlash.WriteLong(get\_int(inertial\_nav.get\_longitude\_velocity())); // 14 accel based lon velocity

DataFlash.WriteByte(END\_BYTE);

#endif

}

// Read an INAV packet

static void Log\_Read\_INAV()

{

int16\_t temp1 = DataFlash.ReadInt(); // 1 barometer altitude

int16\_t temp2 = DataFlash.ReadInt(); // 2 accel + baro filtered altitude

int16\_t temp3 = DataFlash.ReadInt(); // 3 barometer based climb rate

int16\_t temp4 = DataFlash.ReadInt(); // 4 accel + baro based climb rate

float temp5 = get\_float(DataFlash.ReadLong()); // 5 accel correction x-axis

float temp6 = get\_float(DataFlash.ReadLong()); // 6 accel correction y-axis

float temp7 = get\_float(DataFlash.ReadLong()); // 7 accel correction z-axis

float temp8 = get\_float(DataFlash.ReadLong()); // 8 accel correction earth frame

int32\_t temp9 = DataFlash.ReadLong(); // 9 lat from home

int32\_t temp10 = DataFlash.ReadLong(); // 10 lon from home

float temp11 = get\_float(DataFlash.ReadLong()); // 11 accel based lat from home

float temp12 = get\_float(DataFlash.ReadLong()); // 12 accel based lon from home

float temp13 = get\_float(DataFlash.ReadLong()); // 13 accel based lat velocity

float temp14 = get\_float(DataFlash.ReadLong()); // 14 accel based lon velocity

// 1 2 3 4 5 6 7 8 9 10 11 12 13 14

cliSerial->printf\_P(PSTR("INAV, %d, %d, %d, %d, %6.4f, %6.4f, %6.4f, %6.4f, %ld, %ld, %6.4f, %6.4f, %6.4f, %6.4f\n"),

(int)temp1,

(int)temp2,

(int)temp3,

(int)temp4,

temp5,

temp6,

temp7,

temp8,

temp9,

temp10,

temp11,

temp12,

temp13,

temp14);

}

// Write a mode packet. Total length : 7 bytes

static void Log\_Write\_Mode(byte mode)

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_MODE\_MSG);

DataFlash.WriteByte(mode);

DataFlash.WriteInt(g.throttle\_cruise);

DataFlash.WriteByte(END\_BYTE);

}

// Read a mode packet

static void Log\_Read\_Mode()

{

cliSerial->printf\_P(PSTR("MOD:"));

print\_flight\_mode(DataFlash.ReadByte());

cliSerial->printf\_P(PSTR(", %d\n"),(int)DataFlash.ReadInt());

}

// Write Startup packet. Total length : 4 bytes

static void Log\_Write\_Startup()

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_STARTUP\_MSG);

DataFlash.WriteByte(END\_BYTE);

}

// Read a startup packet

static void Log\_Read\_Startup()

{

cliSerial->printf\_P(PSTR("START UP\n"));

}

#define DATA\_INT32 0

#define DATA\_FLOAT 1

#define DATA\_INT16 2

#define DATA\_UINT16 3

#define DATA\_EVENT 4

static void Log\_Write\_Data(uint8\_t \_index, int32\_t \_data)

{

if (g.log\_bitmask == 0) return;

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_DATA\_MSG);

DataFlash.WriteByte(\_index);

DataFlash.WriteByte(DATA\_INT32);

DataFlash.WriteLong(\_data);

DataFlash.WriteByte(END\_BYTE);

}

static void Log\_Write\_Data(uint8\_t \_index, float \_data)

{

if (g.log\_bitmask == 0) return;

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_DATA\_MSG);

DataFlash.WriteByte(\_index);

DataFlash.WriteByte(DATA\_FLOAT);

DataFlash.WriteLong(get\_int(\_data));

DataFlash.WriteByte(END\_BYTE);

}

static void Log\_Write\_Data(uint8\_t \_index, int16\_t \_data)

{

if (g.log\_bitmask == 0) return;

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_DATA\_MSG);

DataFlash.WriteByte(\_index);

DataFlash.WriteByte(DATA\_INT16);

DataFlash.WriteInt(\_data);

DataFlash.WriteByte(END\_BYTE);

}

static void Log\_Write\_Data(uint8\_t \_index, uint16\_t \_data)

{

if (g.log\_bitmask == 0) return;

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_DATA\_MSG);

DataFlash.WriteByte(\_index);

DataFlash.WriteByte(DATA\_UINT16);

DataFlash.WriteInt(\_data);

DataFlash.WriteByte(END\_BYTE);

}

static void Log\_Write\_Event(uint8\_t \_index)

{

if (g.log\_bitmask == 0) return;

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_DATA\_MSG);

DataFlash.WriteByte(\_index);

DataFlash.WriteByte(DATA\_EVENT);

DataFlash.WriteByte(END\_BYTE);

}

// Read a mode packet

static void Log\_Read\_Data()

{

int8\_t \_index = DataFlash.ReadByte();

int8\_t \_type = DataFlash.ReadByte();

if(\_type == DATA\_EVENT) {

cliSerial->printf\_P(PSTR("EV: %u\n"), \_index);

}else if(\_type == DATA\_FLOAT) {

float \_value = get\_float(DataFlash.ReadLong());

cliSerial->printf\_P(PSTR("DATA: %u, %1.6f\n"), \_index, \_value);

}else if(\_type == DATA\_INT16) {

int16\_t \_value = DataFlash.ReadInt();

cliSerial->printf\_P(PSTR("DATA: %u, %d\n"), \_index, \_value);

}else if(\_type == DATA\_UINT16) {

uint16\_t \_value = DataFlash.ReadInt();

cliSerial->printf\_P(PSTR("DATA: %u, %u\n"), \_index, \_value);

}else if(\_type == DATA\_INT32) {

int32\_t \_value = DataFlash.ReadLong();

cliSerial->printf\_P(PSTR("DATA: %u, %ld\n"), \_index, \_value);

}

}

// Write an PID packet. Total length : 28 bytes

static void Log\_Write\_PID(int8\_t pid\_id, int32\_t error, int32\_t p, int32\_t i, int32\_t d, int32\_t output, float gain)

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_PID\_MSG);

DataFlash.WriteByte(pid\_id); // 1

DataFlash.WriteLong(error); // 2

DataFlash.WriteLong(p); // 3

DataFlash.WriteLong(i); // 4

DataFlash.WriteLong(d); // 5

DataFlash.WriteLong(output); // 6

DataFlash.WriteLong(gain \* 1000); // 7

DataFlash.WriteByte(END\_BYTE);

}

// Read a PID packet

static void Log\_Read\_PID()

{

int8\_t temp1 = DataFlash.ReadByte(); // pid id

int32\_t temp2 = DataFlash.ReadLong(); // error

int32\_t temp3 = DataFlash.ReadLong(); // p

int32\_t temp4 = DataFlash.ReadLong(); // i

int32\_t temp5 = DataFlash.ReadLong(); // d

int32\_t temp6 = DataFlash.ReadLong(); // output

float temp7 = DataFlash.ReadLong() / 1000.f; // gain

// 1 2 3 4 5 6 7

cliSerial->printf\_P(PSTR("PID-%d, %ld, %ld, %ld, %ld, %ld, %4.4f\n"),

(int)temp1, // pid id

(long)temp2, // error

(long)temp3, // p

(long)temp4, // i

(long)temp5, // d

(long)temp6, // output

temp7); // gain

}

// Write a DMP attitude packet. Total length : 16 bytes

static void Log\_Write\_DMP()

{

#if SECONDARY\_DMP\_ENABLED == ENABLED

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_DMP\_MSG);

DataFlash.WriteInt((int16\_t)ahrs.roll\_sensor); // 1

DataFlash.WriteInt((int16\_t)ahrs2.roll\_sensor); // 2

DataFlash.WriteInt((int16\_t)ahrs.pitch\_sensor); // 3

DataFlash.WriteInt((int16\_t)ahrs2.pitch\_sensor); // 4

DataFlash.WriteInt((uint16\_t)ahrs.yaw\_sensor); // 5

DataFlash.WriteInt((uint16\_t)ahrs2.yaw\_sensor); // 6

DataFlash.WriteByte(END\_BYTE);

#endif

}

// Read a DMP attitude packet

static void Log\_Read\_DMP()

{

int16\_t temp1 = DataFlash.ReadInt();

int16\_t temp2 = DataFlash.ReadInt();

int16\_t temp3 = DataFlash.ReadInt();

int16\_t temp4 = DataFlash.ReadInt();

uint16\_t temp5 = DataFlash.ReadInt();

uint16\_t temp6 = DataFlash.ReadInt();

// 1 2 3 4 5 6

cliSerial->printf\_P(PSTR("DMP, %d, %d, %d, %d, %u, %u\n"),

(int)temp1,

(int)temp2,

(int)temp3,

(int)temp4,

(unsigned)temp5,

(unsigned)temp6);

}

// Write a Camera packet. Total length : 26 bytes

static void Log\_Write\_Camera()

{

#if CAMERA == ENABLED

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_CAMERA\_MSG);

DataFlash.WriteLong(g\_gps->time); // 1

DataFlash.WriteLong(current\_loc.lat); // 2

DataFlash.WriteLong(current\_loc.lng); // 3

DataFlash.WriteLong(current\_loc.alt); // 4

DataFlash.WriteInt((int16\_t)ahrs.roll\_sensor); // 5

DataFlash.WriteInt((int16\_t)ahrs.pitch\_sensor); // 6

DataFlash.WriteInt((uint16\_t)ahrs.yaw\_sensor); // 7

DataFlash.WriteByte(END\_BYTE);

#endif

}

// Read a camera packet

static void Log\_Read\_Camera()

{

int32\_t temp1 = DataFlash.ReadLong(); // 1 time

int32\_t temp2 = DataFlash.ReadLong(); // 2 lat

int32\_t temp3 = DataFlash.ReadLong(); // 3 lon

float temp4 = DataFlash.ReadLong() / 100.0; // 4 altitude

int16\_t temp5 = DataFlash.ReadInt(); // 5 roll in centidegrees

int16\_t temp6 = DataFlash.ReadInt(); // 6 pitch in centidegrees

uint16\_t temp7 = DataFlash.ReadInt(); // 7 yaw in centidegrees

// 1

cliSerial->printf\_P(PSTR("CAMERA, %ld, "),(long)temp1); // 1 time

print\_latlon(cliSerial, temp2); // 2 lat

cliSerial->print\_P(PSTR(", "));

print\_latlon(cliSerial, temp3); // 3 lon

// 4 5 6 7

cliSerial->printf\_P(PSTR(", %4.4f, %d, %d, %u\n"),

temp4, // 4 altitude

(int)temp5, // 5 roll in centidegrees

(int)temp6, // 6 pitch in centidegrees

(unsigned int)temp7); // 7 yaw in centidegrees

}

// Write an error packet. Total length : 6 bytes

static void Log\_Write\_Error(uint8\_t sub\_system, uint8\_t error\_code)

{

DataFlash.WriteByte(HEAD\_BYTE1);

DataFlash.WriteByte(HEAD\_BYTE2);

DataFlash.WriteByte(LOG\_ERROR\_MSG);

DataFlash.WriteByte(sub\_system); // 1 sub system

DataFlash.WriteByte(error\_code); // 2 error code

DataFlash.WriteByte(END\_BYTE);

}

// Read an error packet

static void Log\_Read\_Error()

{

uint8\_t sub\_system = DataFlash.ReadByte(); // 1 sub system

uint8\_t error\_code = DataFlash.ReadByte(); // 2 error code

cliSerial->print\_P(PSTR("ERR, "));

// print subsystem

switch(sub\_system) {

case ERROR\_SUBSYSTEM\_MAIN:

cliSerial->print\_P(PSTR("MAIN"));

break;

case ERROR\_SUBSYSTEM\_RADIO:

cliSerial->print\_P(PSTR("RADIO"));

break;

case ERROR\_SUBSYSTEM\_COMPASS:

cliSerial->print\_P(PSTR("COM"));

break;

case ERROR\_SUBSYSTEM\_OPTFLOW:

cliSerial->print\_P(PSTR("OF"));

break;

case ERROR\_SUBSYSTEM\_FAILSAFE:

cliSerial->print\_P(PSTR("FS"));

break;

default:

cliSerial->printf\_P(PSTR("%d"),(int)sub\_system); // 1 sub system

break;

}

// print error code

cliSerial->printf\_P(PSTR(", %d\n"),(int)error\_code); // 2 error code

}

// Read the DataFlash log memory

static void Log\_Read(int16\_t start\_page, int16\_t end\_page)

{

int16\_t packet\_count = 0;

#ifdef AIRFRAME\_NAME

cliSerial->printf\_P(PSTR((AIRFRAME\_NAME)

#endif

cliSerial->printf\_P(PSTR("\n" THISFIRMWARE

"\nFree RAM: %u\n"),

(unsigned) memcheck\_available\_memory());

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

cliSerial->printf\_P(PSTR("APM 2\n"));

#elif CONFIG\_APM\_HARDWARE == APM2\_BETA\_HARDWARE

cliSerial->printf\_P(PSTR("APM 2Beta\n"));

#else

cliSerial->printf\_P(PSTR("APM 1\n"));

#endif

#if CLI\_ENABLED == ENABLED

setup\_show(0, NULL);

#endif

if(start\_page > end\_page) {

packet\_count = Log\_Read\_Process(start\_page, DataFlash.df\_NumPages);

packet\_count += Log\_Read\_Process(1, end\_page);

} else {

packet\_count = Log\_Read\_Process(start\_page, end\_page);

}

//cliSerial->printf\_P(PSTR("Number of packets read: %d\n"), (int)packet\_count);

}

// Read the DataFlash log memory : Packet Parser

static int16\_t Log\_Read\_Process(int16\_t start\_page, int16\_t end\_page)

{

byte data;

byte log\_step = 0;

int16\_t page = start\_page;

int16\_t packet\_count = 0;

DataFlash.StartRead(start\_page);

while(page < end\_page && page != -1){

data = DataFlash.ReadByte();

// This is a state machine to read the packets

switch(log\_step) {

case 0:

if(data == HEAD\_BYTE1) // Head byte 1

log\_step++;

break;

case 1:

if(data == HEAD\_BYTE2) // Head byte 2

log\_step++;

else{

log\_step = 0;

cliSerial->println\_P(PSTR("."));

}

break;

case 2:

log\_step = 0;

switch(data) {

case LOG\_ATTITUDE\_MSG:

Log\_Read\_Attitude();

break;

case LOG\_MODE\_MSG:

Log\_Read\_Mode();

break;

case LOG\_CONTROL\_TUNING\_MSG:

Log\_Read\_Control\_Tuning();

break;

case LOG\_NAV\_TUNING\_MSG:

Log\_Read\_Nav\_Tuning();

break;

case LOG\_PERFORMANCE\_MSG:

Log\_Read\_Performance();

break;

case LOG\_RAW\_MSG:

Log\_Read\_Raw();

break;

case LOG\_CMD\_MSG:

Log\_Read\_Cmd();

break;

case LOG\_CURRENT\_MSG:

Log\_Read\_Current();

break;

case LOG\_STARTUP\_MSG:

Log\_Read\_Startup();

break;

case LOG\_MOTORS\_MSG:

Log\_Read\_Motors();

break;

case LOG\_OPTFLOW\_MSG:

Log\_Read\_Optflow();

break;

case LOG\_GPS\_MSG:

Log\_Read\_GPS();

break;

case LOG\_DATA\_MSG:

Log\_Read\_Data();

break;

case LOG\_PID\_MSG:

Log\_Read\_PID();

break;

case LOG\_ITERM\_MSG:

Log\_Read\_Iterm();

break;

case LOG\_DMP\_MSG:

Log\_Read\_DMP();

break;

case LOG\_INAV\_MSG:

Log\_Read\_INAV();

break;

case LOG\_CAMERA\_MSG:

Log\_Read\_Camera();

break;

case LOG\_ERROR\_MSG:

Log\_Read\_Error();

break;

}

break;

case 3:

if(data == END\_BYTE){

packet\_count++;

}else{

cliSerial->printf\_P(PSTR("Error Reading END\_BYTE: %d\n"),data);

}

log\_step = 0; // Restart sequence: new packet...

break;

}

page = DataFlash.GetPage();

}

return packet\_count;

}

#else // LOGGING\_ENABLED

static void Log\_Write\_Startup() {

}

static void Log\_Read\_Startup() {

}

static void Log\_Read(int16\_t start\_page, int16\_t end\_page) {

}

static void Log\_Write\_Cmd(byte num, struct Location \*wp) {

}

static void Log\_Write\_Mode(byte mode) {

}

static void Log\_Write\_Raw() {

}

void print\_latlon(BetterStream \*s, int32\_t lat\_or\_lon) {

}

static void Log\_Write\_GPS() {

}

static void Log\_Write\_Current() {

}

static void Log\_Write\_Iterm() {

}

static void Log\_Write\_Attitude() {

}

static void Log\_Write\_INAV() {

}

static void Log\_Write\_Data(uint8\_t \_index, float \_data){

}

static void Log\_Write\_Data(uint8\_t \_index, int32\_t \_data){

}

static void Log\_Write\_Data(uint8\_t \_index, int16\_t \_data){

}

static void Log\_Write\_Data(uint8\_t \_index, uint16\_t \_data){

}

static void Log\_Write\_Event(uint8\_t \_index){

}

static void Log\_Write\_Optflow() {

}

static void Log\_Write\_Nav\_Tuning() {

}

static void Log\_Write\_Control\_Tuning() {

}

static void Log\_Write\_Motors() {

}

static void Log\_Write\_Performance() {

}

static void Log\_Write\_PID(int8\_t pid\_id, int32\_t error, int32\_t p, int32\_t i, int32\_t d, int32\_t output, float gain) {

}

static void Log\_Write\_DMP() {

}

static void Log\_Write\_Camera() {

}

static void Log\_Write\_Error(uint8\_t sub\_system, uint8\_t error\_code) {

}

static int8\_t process\_logs(uint8\_t argc, const Menu::arg \*argv) {

return 0;

}

#endif // LOGGING\_DISABLED

## Parameters.h

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

#ifndef PARAMETERS\_H

#define PARAMETERS\_H

#include <AP\_Common.h>

// Global parameter class.

//

class Parameters {

public:

// The version of the layout as described by the parameter enum.

//

// When changing the parameter enum in an incompatible fashion, this

// value should be incremented by one.

//

// The increment will prevent old parameters from being used incorrectly

// by newer code.

//

static const uint16\_t k\_format\_version = 120;

// The parameter software\_type is set up solely for ground station use

// and identifies the software type (eg ArduPilotMega versus

// ArduCopterMega)

// GCS will interpret values 0-9 as ArduPilotMega. Developers may use

// values within that range to identify different branches.

//

static const uint16\_t k\_software\_type = 10; // 0 for APM

// trunk

// Parameter identities.

//

// The enumeration defined here is used to ensure that every parameter

// or parameter group has a unique ID number. This number is used by

// AP\_Var to store and locate parameters in EEPROM.

//

// Note that entries without a number are assigned the next number after

// the entry preceding them. When adding new entries, ensure that they

// don't overlap.

//

// Try to group related variables together, and assign them a set

// range in the enumeration. Place these groups in numerical order

// at the end of the enumeration.

//

// WARNING: Care should be taken when editing this enumeration as the

// AP\_Var load/save code depends on the values here to identify

// variables saved in EEPROM.

//

//

enum {

// Layout version number, always key zero.

//

k\_param\_format\_version = 0,

k\_param\_software\_type,

k\_param\_ins\_old, // \*\*\* Deprecated, remove with next eeprom number change

k\_param\_ins, // libraries/AP\_InertialSensor variables

// simulation

k\_param\_sitl = 10,

// Misc

//

k\_param\_log\_bitmask = 20,

k\_param\_log\_last\_filenumber, // \*\*\* Deprecated - remove

// with next eeprom number

// change

k\_param\_toy\_yaw\_rate, // THOR The memory

// location for the

// Yaw Rate 1 = fast,

// 2 = med, 3 = slow

k\_param\_crosstrack\_min\_distance,

k\_param\_rssi\_pin,

k\_param\_throttle\_accel\_enabled,

k\_param\_yaw\_override\_behaviour,

k\_param\_acro\_trainer\_enabled,

k\_param\_pilot\_velocity\_z\_max, // 28

// 65: AP\_Limits Library

k\_param\_limits = 65,

k\_param\_gpslock\_limit,

k\_param\_geofence\_limit,

k\_param\_altitude\_limit,

//

// 80: Heli

//

k\_param\_heli\_servo\_1 = 80,

k\_param\_heli\_servo\_2,

k\_param\_heli\_servo\_3,

k\_param\_heli\_servo\_4,

k\_param\_heli\_pitch\_ff,

k\_param\_heli\_roll\_ff,

k\_param\_heli\_yaw\_ff,

//

// 90: Motors

//

k\_param\_motors = 90,

//

// 100: Inertial Nav

//

k\_param\_inertial\_nav = 100,

// 110: Telemetry control

//

k\_param\_gcs0 = 110,

k\_param\_gcs3,

k\_param\_sysid\_this\_mav,

k\_param\_sysid\_my\_gcs,

k\_param\_serial3\_baud,

k\_param\_telem\_delay,

//

// 140: Sensor parameters

//

k\_param\_imu = 140, // deprecated - can be deleted

k\_param\_battery\_monitoring = 141,

k\_param\_volt\_div\_ratio,

k\_param\_curr\_amp\_per\_volt,

k\_param\_input\_voltage,

k\_param\_pack\_capacity,

k\_param\_compass\_enabled,

k\_param\_compass,

k\_param\_sonar\_enabled,

k\_param\_frame\_orientation,

k\_param\_optflow\_enabled,

k\_param\_low\_voltage,

k\_param\_ch7\_option,

k\_param\_auto\_slew\_rate,

k\_param\_sonar\_type,

k\_param\_super\_simple = 155,

k\_param\_axis\_enabled = 157,

k\_param\_copter\_leds\_mode,

k\_param\_ahrs, // AHRS group

//

// 160: Navigation parameters

//

k\_param\_rtl\_altitude = 160,

k\_param\_crosstrack\_gain,

k\_param\_rtl\_loiter\_time,

k\_param\_rtl\_alt\_final,

k\_param\_tilt\_comp, //164

//

// Camera and mount parameters

//

k\_param\_camera = 165,

k\_param\_camera\_mount,

k\_param\_camera\_mount2,

//

// Batery monitoring parameters

//

k\_param\_battery\_volt\_pin = 168,

k\_param\_battery\_curr\_pin, // 169

//

// 170: Radio settings

//

k\_param\_rc\_1 = 170,

k\_param\_rc\_2,

k\_param\_rc\_3,

k\_param\_rc\_4,

k\_param\_rc\_5,

k\_param\_rc\_6,

k\_param\_rc\_7,

k\_param\_rc\_8,

k\_param\_rc\_10,

k\_param\_rc\_11,

k\_param\_throttle\_min,

k\_param\_throttle\_max,

k\_param\_failsafe\_throttle,

k\_param\_throttle\_fs\_action, // remove

k\_param\_failsafe\_throttle\_value,

k\_param\_throttle\_cruise,

k\_param\_esc\_calibrate,

k\_param\_radio\_tuning,

k\_param\_radio\_tuning\_high,

k\_param\_radio\_tuning\_low,

k\_param\_rc\_speed = 192,

k\_param\_failsafe\_battery\_enabled,

k\_param\_throttle\_mid, // 194

//

// 200: flight modes

//

k\_param\_flight\_mode1 = 200,

k\_param\_flight\_mode2,

k\_param\_flight\_mode3,

k\_param\_flight\_mode4,

k\_param\_flight\_mode5,

k\_param\_flight\_mode6,

k\_param\_simple\_modes,

//

// 210: Waypoint data

//

k\_param\_waypoint\_mode = 210, // remove

k\_param\_command\_total,

k\_param\_command\_index,

k\_param\_command\_nav\_index, // remove

k\_param\_waypoint\_radius,

k\_param\_circle\_radius,

k\_param\_waypoint\_speed\_max,

k\_param\_land\_speed,

k\_param\_auto\_velocity\_z\_min,

k\_param\_auto\_velocity\_z\_max, // 219

//

// 220: PI/D Controllers

//

k\_param\_acro\_p = 221,

k\_param\_axis\_lock\_p, // remove

k\_param\_pid\_rate\_roll,

k\_param\_pid\_rate\_pitch,

k\_param\_pid\_rate\_yaw,

k\_param\_pi\_stabilize\_roll,

k\_param\_pi\_stabilize\_pitch,

k\_param\_pi\_stabilize\_yaw,

k\_param\_pi\_loiter\_lat,

k\_param\_pi\_loiter\_lon,

k\_param\_pid\_loiter\_rate\_lat,

k\_param\_pid\_loiter\_rate\_lon,

k\_param\_pid\_nav\_lat,

k\_param\_pid\_nav\_lon,

k\_param\_pi\_alt\_hold,

k\_param\_pid\_throttle,

k\_param\_pid\_optflow\_roll,

k\_param\_pid\_optflow\_pitch,

k\_param\_acro\_balance\_roll, // scalar (not PID)

k\_param\_acro\_balance\_pitch, // scalar (not PID)

k\_param\_pid\_throttle\_accel, // 241

// 254,255: reserved

};

AP\_Int16 format\_version;

AP\_Int8 software\_type;

// Telemetry control

//

AP\_Int16 sysid\_this\_mav;

AP\_Int16 sysid\_my\_gcs;

AP\_Int8 serial3\_baud;

AP\_Int8 telem\_delay;

AP\_Int16 rtl\_altitude;

AP\_Int8 sonar\_enabled;

AP\_Int8 sonar\_type; // 0 = XL, 1 = LV,

// 2 = XLL (XL with 10m range)

// 3 = HRLV

AP\_Int8 battery\_monitoring; // 0=disabled, 3=voltage only,

// 4=voltage and current

AP\_Float volt\_div\_ratio;

AP\_Float curr\_amp\_per\_volt;

AP\_Float input\_voltage;

AP\_Int16 pack\_capacity; // Battery pack capacity less reserve

AP\_Int8 failsafe\_battery\_enabled; // battery failsafe enabled

AP\_Int8 compass\_enabled;

AP\_Int8 optflow\_enabled;

AP\_Float low\_voltage;

AP\_Int8 super\_simple;

AP\_Int16 rtl\_alt\_final;

AP\_Int8 tilt\_comp;

AP\_Int8 axis\_enabled;

AP\_Int8 copter\_leds\_mode; // Operating mode of LED

// lighting system

AP\_Int8 battery\_volt\_pin;

AP\_Int8 battery\_curr\_pin;

AP\_Int8 rssi\_pin;

AP\_Int8 throttle\_accel\_enabled; // enable/disable accel based throttle controller

AP\_Int8 yaw\_override\_behaviour; // controls when autopilot takes back normal control of yaw after pilot overrides

// Waypoints

//

AP\_Int8 command\_total;

AP\_Int8 command\_index;

AP\_Int16 waypoint\_radius;

AP\_Int16 circle\_radius;

AP\_Int16 waypoint\_speed\_max;

AP\_Float crosstrack\_gain;

AP\_Int16 crosstrack\_min\_distance;

AP\_Int32 rtl\_loiter\_time;

AP\_Int16 land\_speed;

AP\_Int16 auto\_velocity\_z\_min; // minimum vertical velocity (i.e. maximum descent) the autopilot may request

AP\_Int16 auto\_velocity\_z\_max; // maximum vertical velocity the autopilot may request

AP\_Int16 pilot\_velocity\_z\_max; // maximum vertical velocity the pilot may request

// Throttle

//

AP\_Int16 throttle\_min;

AP\_Int16 throttle\_max;

AP\_Int8 failsafe\_throttle;

AP\_Int16 failsafe\_throttle\_value;

AP\_Int16 throttle\_cruise;

AP\_Int16 throttle\_mid;

// Flight modes

//

AP\_Int8 flight\_mode1;

AP\_Int8 flight\_mode2;

AP\_Int8 flight\_mode3;

AP\_Int8 flight\_mode4;

AP\_Int8 flight\_mode5;

AP\_Int8 flight\_mode6;

AP\_Int8 simple\_modes;

// Misc

//

AP\_Int16 log\_bitmask;

AP\_Int8 toy\_yaw\_rate; // THOR The

// Yaw Rate 1

// = fast, 2 =

// med, 3 =

// slow

AP\_Int8 esc\_calibrate;

AP\_Int8 radio\_tuning;

AP\_Int16 radio\_tuning\_high;

AP\_Int16 radio\_tuning\_low;

AP\_Int8 frame\_orientation;

AP\_Int8 ch7\_option;

AP\_Int16 auto\_slew\_rate;

#if FRAME\_CONFIG == HELI\_FRAME

// Heli

RC\_Channel heli\_servo\_1, heli\_servo\_2, heli\_servo\_3, heli\_servo\_4; // servos for swash plate and tail

AP\_Float heli\_pitch\_ff; // pitch rate feed-forward

AP\_Float heli\_roll\_ff; // roll rate feed-forward

AP\_Float heli\_yaw\_ff; // yaw rate feed-forward

#endif

// RC channels

RC\_Channel rc\_1;

RC\_Channel rc\_2;

RC\_Channel rc\_3;

RC\_Channel rc\_4;

RC\_Channel\_aux rc\_5;

RC\_Channel\_aux rc\_6;

RC\_Channel\_aux rc\_7;

RC\_Channel\_aux rc\_8;

#if MOUNT == ENABLED

RC\_Channel\_aux rc\_10;

RC\_Channel\_aux rc\_11;

#endif

AP\_Int16 rc\_speed; // speed of fast RC Channels in Hz

// Acro parameters

AP\_Float acro\_p;

AP\_Int16 acro\_balance\_roll;

AP\_Int16 acro\_balance\_pitch;

AP\_Int8 acro\_trainer\_enabled;

// PI/D controllers

AC\_PID pid\_rate\_roll;

AC\_PID pid\_rate\_pitch;

AC\_PID pid\_rate\_yaw;

AC\_PID pid\_loiter\_rate\_lat;

AC\_PID pid\_loiter\_rate\_lon;

AC\_PID pid\_nav\_lat;

AC\_PID pid\_nav\_lon;

AC\_PID pid\_throttle;

AC\_PID pid\_throttle\_accel;

AC\_PID pid\_optflow\_roll;

AC\_PID pid\_optflow\_pitch;

APM\_PI pi\_loiter\_lat;

APM\_PI pi\_loiter\_lon;

APM\_PI pi\_stabilize\_roll;

APM\_PI pi\_stabilize\_pitch;

APM\_PI pi\_stabilize\_yaw;

APM\_PI pi\_alt\_hold;

// Note: keep initializers here in the same order as they are declared

// above.

Parameters() :

#if FRAME\_CONFIG == HELI\_FRAME

heli\_servo\_1 (CH\_1),

heli\_servo\_2 (CH\_2),

heli\_servo\_3 (CH\_3),

heli\_servo\_4 (CH\_4),

#endif

rc\_1 (CH\_1),

rc\_2 (CH\_2),

rc\_3 (CH\_3),

rc\_4 (CH\_4),

rc\_5 (CH\_5),

rc\_6 (CH\_6),

rc\_7 (CH\_7),

rc\_8 (CH\_8),

#if MOUNT == ENABLED

rc\_10 (CH\_10),

rc\_11 (CH\_11),

#endif

// PID controller initial P initial I initial D

// initial imax

//-----------------------------------------------------------------------------------------------------

pid\_rate\_roll (RATE\_ROLL\_P, RATE\_ROLL\_I, RATE\_ROLL\_D, RATE\_ROLL\_IMAX \* 100),

pid\_rate\_pitch (RATE\_PITCH\_P, RATE\_PITCH\_I, RATE\_PITCH\_D, RATE\_PITCH\_IMAX \* 100),

pid\_rate\_yaw (RATE\_YAW\_P, RATE\_YAW\_I, RATE\_YAW\_D, RATE\_YAW\_IMAX \* 100),

pid\_loiter\_rate\_lat (LOITER\_RATE\_P, LOITER\_RATE\_I, LOITER\_RATE\_D, LOITER\_RATE\_IMAX \* 100),

pid\_loiter\_rate\_lon (LOITER\_RATE\_P, LOITER\_RATE\_I, LOITER\_RATE\_D, LOITER\_RATE\_IMAX \* 100),

pid\_nav\_lat (NAV\_P, NAV\_I, NAV\_D, NAV\_IMAX \* 100),

pid\_nav\_lon (NAV\_P, NAV\_I, NAV\_D, NAV\_IMAX \* 100),

pid\_throttle (THROTTLE\_P, THROTTLE\_I, THROTTLE\_D, THROTTLE\_IMAX),

pid\_throttle\_accel (THROTTLE\_ACCEL\_P, THROTTLE\_ACCEL\_I, THROTTLE\_ACCEL\_D, THROTTLE\_ACCEL\_IMAX),

pid\_optflow\_roll (OPTFLOW\_ROLL\_P, OPTFLOW\_ROLL\_I, OPTFLOW\_ROLL\_D, OPTFLOW\_IMAX \* 100),

pid\_optflow\_pitch (OPTFLOW\_PITCH\_P, OPTFLOW\_PITCH\_I, OPTFLOW\_PITCH\_D, OPTFLOW\_IMAX \* 100),

// PI controller initial P initial I initial

// imax

//----------------------------------------------------------------------

pi\_loiter\_lat (LOITER\_P, LOITER\_I, LOITER\_IMAX \* 100),

pi\_loiter\_lon (LOITER\_P, LOITER\_I, LOITER\_IMAX \* 100),

pi\_stabilize\_roll (STABILIZE\_ROLL\_P, STABILIZE\_ROLL\_I, STABILIZE\_ROLL\_IMAX \* 100),

pi\_stabilize\_pitch (STABILIZE\_PITCH\_P, STABILIZE\_PITCH\_I, STABILIZE\_PITCH\_IMAX \* 100),

pi\_stabilize\_yaw (STABILIZE\_YAW\_P, STABILIZE\_YAW\_I, STABILIZE\_YAW\_IMAX \* 100),

pi\_alt\_hold (ALT\_HOLD\_P, ALT\_HOLD\_I, ALT\_HOLD\_IMAX)

{

}

};

extern const AP\_Param::Info var\_info[];

#endif // PARAMETERS\_H

## Parameters

/// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: t -\*-

/\*

\* ArduCopter parameter definitions

\*

\* This firmware is free software; you can redistribute it and/or

\* modify it under the terms of the GNU Lesser General Public

\* License as published by the Free Software Foundation; either

\* version 2.1 of the License, or (at your option) any later version.

\*/

#define GSCALAR(v, name, def) { g.v.vtype, name, Parameters::k\_param\_ ## v, &g.v, {def\_value : def} }

#define GGROUP(v, name, class) { AP\_PARAM\_GROUP, name, Parameters::k\_param\_ ## v, &g.v, {group\_info : class::var\_info} }

#define GOBJECT(v, name, class) { AP\_PARAM\_GROUP, name, Parameters::k\_param\_ ## v, &v, {group\_info : class::var\_info} }

const AP\_Param::Info var\_info[] PROGMEM = {

// @Param: SYSID\_SW\_MREV

// @DisplayName: Eeprom format version number

// @Description: This value is incremented when changes are made to the eeprom format

// @User: Advanced

GSCALAR(format\_version, "SYSID\_SW\_MREV", 0),

// @Param: SYSID\_SW\_TYPE

// @DisplayName: Software Type

// @Description: This is used by the ground station to recognise the software type (eg ArduPlane vs ArduCopter)

// @User: Advanced

GSCALAR(software\_type, "SYSID\_SW\_TYPE", Parameters::k\_software\_type),

// @Param: SYSID\_THISMAV

// @DisplayName: Mavlink version

// @Description: Allows reconising the mavlink version

// @User: Advanced

GSCALAR(sysid\_this\_mav, "SYSID\_THISMAV", MAV\_SYSTEM\_ID),

GSCALAR(sysid\_my\_gcs, "SYSID\_MYGCS", 255),

// @Param: SERIAL3\_BAUD

// @DisplayName: Telemetry Baud Rate

// @Description: The baud rate used on the telemetry port

// @Values: 1:1200,2:2400,4:4800,9:9600,19:19200,38:38400,57:57600,111:111100,115:115200

// @User: Standard

GSCALAR(serial3\_baud, "SERIAL3\_BAUD", SERIAL3\_BAUD/1000),

// @Param: TELEM\_DELAY

// @DisplayName: Telemetry startup delay

// @Description: The amount of time (in seconds) to delay radio telemetry to prevent an Xbee bricking on power up

// @User: Standard

// @Units: seconds

// @Range: 0 10

// @Increment: 1

GSCALAR(telem\_delay, "TELEM\_DELAY", 0),

// @Param: ALT\_RTL

// @DisplayName: RTL Altitude

// @Description: The minimum altitude the model will move to before Returning to Launch. Set to zero to return at current altitude.

// @Units: Centimeters

// @Range: 0 4000

// @Increment: 1

// @User: Standard

GSCALAR(rtl\_altitude, "RTL\_ALT", RTL\_ALT),

// @Param: SONAR\_ENABLE

// @DisplayName: Enable Sonar

// @Description: Setting this to Enabled(1) will enable the sonar. Setting this to Disabled(0) will disable the sonar

// @Values: 0:Disabled,1:Enabled

// @User: Standard

GSCALAR(sonar\_enabled, "SONAR\_ENABLE", DISABLED),

// @Param: SONAR\_TYPE

// @DisplayName: Sonar type

// @Description: Used to adjust scaling to match the sonar used (only Maxbotix sonars are supported at this time)

// @Values: 0:XL-EZ0,1:LV-EZ0,2:XLL-EZ0,3:HRLV

// @User: Standard

GSCALAR(sonar\_type, "SONAR\_TYPE", AP\_RANGEFINDER\_MAXSONARXL),

// @Param: BATT\_MONITOR

// @DisplayName: Battery monitoring

// @Description: Controls enabling monitoring of the battery's voltage and current

// @Values: 0:Disabled,3:Voltage Only,4:Voltage and Current

// @User: Standard

GSCALAR(battery\_monitoring, "BATT\_MONITOR", DISABLED),

// @Param: FS\_BATT\_ENABLE

// @DisplayName: Battery Failsafe Enable

// @Description: Controls whether failsafe will be invoked when battery voltage or current runs low

// @Values: 0:Disabled,1:Enabled

// @User: Standard

GSCALAR(failsafe\_battery\_enabled, "FS\_BATT\_ENABLE", FS\_BATTERY),

// @Param: VOLT\_DIVIDER

// @DisplayName: Voltage Divider

// @Description: Used to convert the voltage of the voltage sensing pin (BATT\_VOLT\_PIN) to the actual battery's voltage (pin voltage \* INPUT\_VOLTS/1024 \* VOLT\_DIVIDER)

// @User: Advanced

GSCALAR(volt\_div\_ratio, "VOLT\_DIVIDER", VOLT\_DIV\_RATIO),

// @Param: AMP\_PER\_VOLT

// @DisplayName: Current Amps per volt

// @Description: Used to convert the voltage on the current sensing pin (BATT\_CURR\_PIN) to the actual current being consumed in amps (curr pin voltage \* INPUT\_VOLTS/1024 \* AMP\_PER\_VOLT )

// @User: Advanced

GSCALAR(curr\_amp\_per\_volt, "AMP\_PER\_VOLT", CURR\_AMP\_PER\_VOLT),

// @Param: INPUT\_VOLTS

// @DisplayName: Max internal voltage of the battery voltage and current sensing pins

// @Description: Used to convert the voltage read in on the voltage and current pins for battery monitoring. Normally 5 meaning 5 volts.

// @User: Advanced

GSCALAR(input\_voltage, "INPUT\_VOLTS", INPUT\_VOLTAGE),

// @Param: BATT\_CAPACITY

// @DisplayName: Battery Capacity

// @Description: Battery capacity in milliamp-hours (mAh)

// @Units: mAh

GSCALAR(pack\_capacity, "BATT\_CAPACITY", HIGH\_DISCHARGE),

// @Param: MAG\_ENABLE

// @DisplayName: Enable Compass

// @Description: Setting this to Enabled(1) will enable the compass. Setting this to Disabled(0) will disable the compass

// @Values: 0:Disabled,1:Enabled

// @User: Standard

GSCALAR(compass\_enabled, "MAG\_ENABLE", MAGNETOMETER),

// @Param: FLOW\_ENABLE

// @DisplayName: Enable Optical Flow

// @Description: Setting this to Enabled(1) will enable optical flow. Setting this to Disabled(0) will disable optical flow

// @Values: 0:Disabled,1:Enabled

// @User: Standard

GSCALAR(optflow\_enabled, "FLOW\_ENABLE", DISABLED),

// @Param: LOW\_VOLT

// @DisplayName: Low Voltage

// @Description: Set this to the voltage you want to represent low voltage

// @Range: 0 20

// @Increment: .1

// @User: Standard

GSCALAR(low\_voltage, "LOW\_VOLT", LOW\_VOLTAGE),

// @Param: SUPER\_SIMPLE

// @DisplayName: Enable Super Simple Mode

// @Description: Setting this to Enabled(1) will enable Super Simple Mode. Setting this to Disabled(0) will disable Super Simple Mode

// @Values: 0:Disabled,1:Enabled

// @User: Standard

GSCALAR(super\_simple, "SUPER\_SIMPLE", SUPER\_SIMPLE),

// @Param: RTL\_ALT\_FINAL

// @DisplayName: RTL Final Altitude

// @Description: This is the altitude the vehicle will move to as the final stage of Returning to Launch or after completing a mission. Set to -1 to disable, zero to land.

// @Units: Centimeters

// @Range: -1 1000

// @Increment: 1

// @User: Standard

GSCALAR(rtl\_alt\_final, "RTL\_ALT\_FINAL", RTL\_ALT\_FINAL),

// @Param: TILT

// @DisplayName: Auto Tilt Compensation

// @Description: This is a feed-forward compensation which helps the aircraft achieve target waypoint speed.

// @Range: 0 100

// @Increment: 1

// @User: Advanced

GSCALAR(tilt\_comp, "TILT", TILT\_COMPENSATION),

// @Param: BATT\_VOLT\_PIN

// @DisplayName: Battery Voltage sensing pin

// @Description: Setting this to 0 ~ 13 will enable battery current sensing on pins A0 ~ A13.

// @Values: -1:Disabled, 0:A0, 1:A1, 13:A13

// @User: Standard

GSCALAR(battery\_volt\_pin, "BATT\_VOLT\_PIN", BATTERY\_VOLT\_PIN),

// @Param: BATT\_CURR\_PIN

// @DisplayName: Battery Current sensing pin

// @Description: Setting this to 0 ~ 13 will enable battery current sensing on pins A0 ~ A13.

// @Values: -1:Disabled, 1:A1, 2:A2, 13:A13

// @User: Standard

GSCALAR(battery\_curr\_pin, "BATT\_CURR\_PIN", BATTERY\_CURR\_PIN),

// @Param: RSSI\_PIN

// @DisplayName: Receiver RSSI sensing pin

// @Description: This selects an analog pin for the receiver RSSI voltage. It assumes the voltage is 5V for max rssi, 0V for minimum

// @Values: -1:Disabled, 0:A0, 1:A1, 2:A2, 13:A13

// @User: Standard

GSCALAR(rssi\_pin, "RSSI\_PIN", -1),

// @Param: THR\_ACC\_ENABLE

// @DisplayName: Enable Accel based throttle controller

// @Description: This allows enabling and disabling the accelerometer based throttle controller. If disabled a velocity based controller is used.

// @Values: 0:Disabled, 1:Enabled

// @User: Standard

GSCALAR(throttle\_accel\_enabled, "THR\_ACC\_ENABLE", 1),

// @Param: YAW\_OVR\_BEHAVE

// @DisplayName: Yaw override behaviour

// @Description: Controls when autopilot takes back normal control of yaw after pilot overrides

// @Values: 0:At Next WP, 1:On Mission Restart

// @User: Advanced

GSCALAR(yaw\_override\_behaviour, "YAW\_OVR\_BEHAVE", YAW\_OVERRIDE\_BEHAVIOUR\_AT\_NEXT\_WAYPOINT),

// @Param: WP\_TOTAL

// @DisplayName: Waypoint Total

// @Description: Total number of commands in the mission stored in the eeprom. Do not update this parameter directly!

// @User: Advanced

GSCALAR(command\_total, "WP\_TOTAL", 0),

// @Param: WP\_INDEX

// @DisplayName: Waypoint Index

// @Description: The index number of the command that is currently being executed. Do not update this parameter directly!

// @User: Advanced

GSCALAR(command\_index, "WP\_INDEX", 0),

// @Param: WP\_RADIUS

// @DisplayName: Waypoint Radius

// @Description: Defines the distance from a waypoint, that when crossed indicates the wp has been hit.

// @Units: Meters

// @Range: 1 127

// @Increment: 1

// @User: Standard

GSCALAR(waypoint\_radius, "WP\_RADIUS", WP\_RADIUS\_DEFAULT),

// @Param: CIRCLE\_RADIUS

// @DisplayName: Circle radius

// @Description: Defines the radius of the circle the vehicle will fly when in Circle flight mode

// @Units: Meters

// @Range: 1 127

// @Increment: 1

// @User: Standard

GSCALAR(circle\_radius, "CIRCLE\_RADIUS", CIRCLE\_RADIUS),

// @Param: WP\_SPEED\_MAX

// @DisplayName: Waypoint Max Speed Target

// @Description: Defines the speed which the aircraft will attempt to maintain during a WP mission.

// @Units: Centimeters/Second

// @Increment: 100

// @User: Standard

GSCALAR(waypoint\_speed\_max, "WP\_SPEED\_MAX", WAYPOINT\_SPEED\_MAX),

// @Param: XTRK\_GAIN\_SC

// @DisplayName: Cross-Track Gain

// @Description: This controls the rate that the Auto Controller will attempt to return original track

// @Units: Dimensionless

// @User: Standard

GSCALAR(crosstrack\_gain, "XTRK\_GAIN\_SC", CROSSTRACK\_GAIN),

// @Param: XTRK\_MIN\_DIST

// @DisplayName: Crosstrack mininum distance

// @Description: Minimum distance in meters between waypoints to do crosstrack correction.

// @Units: Meters

// @Range: 0 32767

// @Increment: 1

// @User: Standard

GSCALAR(crosstrack\_min\_distance, "XTRK\_MIN\_DIST", CROSSTRACK\_MIN\_DISTANCE),

// @Param: RTL\_LOITER\_TIME

// @DisplayName: RTL loiter time

// @Description: Time (in milliseconds) to loiter above home before begining final descent

// @Units: ms

// @Range: 0 60000

// @Increment: 1000

// @User: Standard

GSCALAR(rtl\_loiter\_time, "RTL\_LOIT\_TIME", RTL\_LOITER\_TIME),

// @Param: LAND\_SPEED

// @DisplayName: Land speed

// @Description: The descent speed for the final stage of landing in cm/s

// @Units: Centimeters/Second

// @Range: 10 200

// @Increment: 10

// @User: Standard

GSCALAR(land\_speed, "LAND\_SPEED", LAND\_SPEED),

// @Param: AUTO\_VELZ\_MIN

// @DisplayName: Autopilot's min vertical speed (max descent) in cm/s

// @Description: The minimum vertical velocity (i.e. descent speed) the autopilot may request in cm/s

// @Units: Centimeters/Second

// @Range: -500 -50

// @Increment: 10

// @User: Standard

GSCALAR(auto\_velocity\_z\_min, "AUTO\_VELZ\_MIN", AUTO\_VELZ\_MIN),

// @Param: AUTO\_VELZ\_MAX

// @DisplayName: Auto pilot's max vertical speed in cm/s

// @Description: The maximum vertical velocity the autopilot may request in cm/s

// @Units: Centimeters/Second

// @Range: 50 500

// @Increment: 10

// @User: Standard

GSCALAR(auto\_velocity\_z\_max, "AUTO\_VELZ\_MAX", AUTO\_VELZ\_MAX),

// @Param: PILOT\_VELZ\_MAX

// @DisplayName: Pilot maximum vertical speed

// @Description: The maximum vertical velocity the pilot may request in cm/s

// @Units: Centimeters/Second

// @Range: 10 500

// @Increment: 10

// @User: Standard

GSCALAR(pilot\_velocity\_z\_max, "PILOT\_VELZ\_MAX", PILOT\_VELZ\_MAX),

// @Param: THR\_MIN

// @DisplayName: Minimum Throttle

// @Description: The minimum throttle that will be sent to the motors to keep them spinning

// @Units: ms

// @Range: 0 1000

// @Increment: 1

// @User: Standard

GSCALAR(throttle\_min, "THR\_MIN", MINIMUM\_THROTTLE),

// @Param: THR\_MAX

// @DisplayName: Maximum Throttle

// @Description: The maximum throttle that will be sent to the motors

// @Units: ms

// @Range: 0 1000

// @Increment: 1

// @User: Standard

GSCALAR(throttle\_max, "THR\_MAX", MAXIMUM\_THROTTLE),

// @Param: FS\_THR\_ENABLE

// @DisplayName: Throttle Failsafe Enable

// @Description: The throttle failsafe allows you to configure a software failsafe activated by a setting on the throttle input channel

// @Values: 0:Disabled,1:Enabled always RTL,2:Enabled Continue with Mission in Auto Mode

// @User: Standard

GSCALAR(failsafe\_throttle, "FS\_THR\_ENABLE", FS\_THR\_DISABLED),

// @Param: FS\_THR\_VALUE

// @DisplayName: Throttle Failsafe Value

// @Description: The PWM level on channel 3 below which throttle sailsafe triggers

// @User: Standard

GSCALAR(failsafe\_throttle\_value, "FS\_THR\_VALUE", FS\_THR\_VALUE\_DEFAULT),

// @Param: TRIM\_THROTTLE

// @DisplayName: Throttle Trim

// @Description: The PWM level on channel 3 below which throttle sailsafe triggers

// @User: Standard

GSCALAR(throttle\_cruise, "TRIM\_THROTTLE", THROTTLE\_CRUISE),

// @Param: THR\_MID

// @DisplayName: Throttle Mid Position

// @Description: The throttle output (0 ~ 1000) when throttle stick is in mid position. Used to scale the manual throttle so that the mid throttle stick position is close to the throttle required to hover

// @User: Standard

// @Range: 300 700

// @Increment: 1

GSCALAR(throttle\_mid, "THR\_MID", THR\_MID),

// @Param: FLTMODE1

// @DisplayName: Flight Mode 1

// @Description: Flight mode when Channel 5 pwm is <= 1230

// @User: Standard

GSCALAR(flight\_mode1, "FLTMODE1", FLIGHT\_MODE\_1),

// @Param: FLTMODE2

// @DisplayName: Flight Mode 2

// @Description: Flight mode when Channel 5 pwm is >1230, <= 1360

// @User: Standard

GSCALAR(flight\_mode2, "FLTMODE2", FLIGHT\_MODE\_2),

// @Param: FLTMODE3

// @DisplayName: Flight Mode 3

// @Description: Flight mode when Channel 5 pwm is >1360, <= 1490

// @User: Standard

GSCALAR(flight\_mode3, "FLTMODE3", FLIGHT\_MODE\_3),

// @Param: FLTMODE4

// @DisplayName: Flight Mode 4

// @Description: Flight mode when Channel 5 pwm is >1490, <= 1620

// @User: Standard

GSCALAR(flight\_mode4, "FLTMODE4", FLIGHT\_MODE\_4),

// @Param: FLTMODE5

// @DisplayName: Flight Mode 5

// @Description: Flight mode when Channel 5 pwm is >1620, <= 1749

// @User: Standard

GSCALAR(flight\_mode5, "FLTMODE5", FLIGHT\_MODE\_5),

// @Param: FLTMODE6

// @DisplayName: Flight Mode 6

// @Description: Flight mode when Channel 5 pwm is >=1750

// @User: Standard

GSCALAR(flight\_mode6, "FLTMODE6", FLIGHT\_MODE\_6),

// @Param: SIMPLE

// @DisplayName: Simple mode bitmask

// @Description: Bitmask which holds which flight modes use simple heading mode (eg bit 0 = 1 means Flight Mode 0 uses simple mode)

// @User: Advanced

GSCALAR(simple\_modes, "SIMPLE", 0),

// @Param: LOG\_BITMASK

// @DisplayName: Log bitmask

// @Description: 2 byte bitmap of log types to enable

// @User: Advanced

GSCALAR(log\_bitmask, "LOG\_BITMASK", DEFAULT\_LOG\_BITMASK),

// @Param: TOY\_RATE

// @DisplayName: Toy Yaw Rate

// @Description: Controls yaw rate in Toy mode. Higher values will cause a slower yaw rate. Do not set to zero!

// @User: Advanced

// @Range: 1 10

GSCALAR(toy\_yaw\_rate, "TOY\_RATE", 1),

// @Param: ESC

// @DisplayName: ESC Calibration

// @Description: Controls whether ArduCopter will enter ESC calibration on the next restart. Do not adjust this parameter manually.

// @User: Advanced

// @Values: 0:Normal Start-up,1:Start-up in ESC Calibration mode

GSCALAR(esc\_calibrate, "ESC", 0),

// @Param: TUNE

// @DisplayName: Channel 6 Tuning

// @Description: Controls which parameters (normally PID gains) are being tuned with transmitter's channel 6 knob

// @User: Standard

// @Values: 0:CH6\_NONE,1:CH6\_STABILIZE\_KP,2:CH6\_STABILIZE\_KI,3:CH6\_YAW\_KP,4:CH6\_RATE\_KP,5:CH6\_RATE\_KI,6:CH6\_YAW\_RATE\_KP,7:CH6\_THROTTLE\_KP,8:CH6\_TOP\_BOTTOM\_RATIO,9:CH6\_RELAY,10:CH6\_TRAVERSE\_SPEED,11:CH6\_NAV\_KP,12:CH6\_LOITER\_KP,13:CH6\_HELI\_EXTERNAL\_GYRO,14:CH6\_THR\_HOLD\_KP,17:CH6\_OPTFLOW\_KP,18:CH6\_OPTFLOW\_KI,19:CH6\_OPTFLOW\_KD,20:CH6\_NAV\_KI,21:CH6\_RATE\_KD,22:CH6\_LOITER\_RATE\_KP,23:CH6\_LOITER\_RATE\_KD,24:CH6\_YAW\_KI,25:CH6\_ACRO\_KP,26:CH6\_YAW\_RATE\_KD,27:CH6\_LOITER\_KI,28:CH6\_LOITER\_RATE\_KI,29:CH6\_STABILIZE\_KD,30:CH6\_AHRS\_YAW\_KP,31:CH6\_AHRS\_KP,32:CH6\_INAV\_TC,33:CH6\_THROTTLE\_KI,34:CH6\_THR\_ACCEL\_KP,35:CH6\_THR\_ACCEL\_KI,36:CH6\_THR\_ACCEL\_KD

GSCALAR(radio\_tuning, "TUNE", 0),

// @Param: TUNE\_LOW

// @DisplayName: Tuning minimum

// @Description: The minimum value that will be applied to the parameter currently being tuned with the transmitter's channel 6 knob

// @User: Standard

// @Range: 0 32767

GSCALAR(radio\_tuning\_low, "TUNE\_LOW", 0),

// @Param: TUNE\_HIGH

// @DisplayName: Tuning maximum

// @Description: The maximum value that will be applied to the parameter currently being tuned with the transmitter's channel 6 knob

// @User: Standard

// @Range: 0 32767

GSCALAR(radio\_tuning\_high, "TUNE\_HIGH", 1000),

// @Param: FRAME

// @DisplayName: Frame Orientation

// @Description: Congrols motor mixing The maximum value that will be applied to the parameter currently being tuned with the transmitter's channel 6 knob

// @User: Standard

// @Range: 0 32767

GSCALAR(frame\_orientation, "FRAME", FRAME\_ORIENTATION),

// @Param: CH7\_OPT

// @DisplayName: Channel 7 option

// @Description: Select which function if performed when CH7 is above 1800 pwm

// @Values: 0:Do Nothing, 2:Flip, 3:Simple Mode, 4:RTL, 5:Save Trim, 7:Save WP, 9:Camera Trigger

// @User: Standard

GSCALAR(ch7\_option, "CH7\_OPT", CH7\_OPTION),

// @Param: AUTO\_SLEW

// @DisplayName: Auto Slew Rate

// @Description: This restricts the rate of change of the roll and pitch attitude commanded by the auto pilot

// @Units: Degrees/Second

// @Range: 1 45

// @Increment: 1

// @User: Advanced

GSCALAR(auto\_slew\_rate, "AUTO\_SLEW", AUTO\_SLEW\_RATE),

#if FRAME\_CONFIG == HELI\_FRAME

GGROUP(heli\_servo\_1, "HS1\_", RC\_Channel),

GGROUP(heli\_servo\_2, "HS2\_", RC\_Channel),

GGROUP(heli\_servo\_3, "HS3\_", RC\_Channel),

GGROUP(heli\_servo\_4, "HS4\_", RC\_Channel),

GSCALAR(heli\_pitch\_ff, "RATE\_PIT\_FF", HELI\_PITCH\_FF),

GSCALAR(heli\_roll\_ff, "RATE\_RLL\_FF", HELI\_ROLL\_FF),

GSCALAR(heli\_yaw\_ff, "RATE\_YAW\_FF", HELI\_YAW\_FF),

#endif

// RC channel

//-----------

// @Group: RC1\_

// @Path: ../libraries/RC\_Channel/RC\_Channel.cpp

GGROUP(rc\_1, "RC1\_", RC\_Channel),

// @Group: RC2\_

// @Path: ../libraries/RC\_Channel/RC\_Channel.cpp

GGROUP(rc\_2, "RC2\_", RC\_Channel),

// @Group: RC3\_

// @Path: ../libraries/RC\_Channel/RC\_Channel.cpp

GGROUP(rc\_3, "RC3\_", RC\_Channel),

// @Group: RC4\_

// @Path: ../libraries/RC\_Channel/RC\_Channel.cpp

GGROUP(rc\_4, "RC4\_", RC\_Channel),

// @Group: RC5\_

// @Path: ../libraries/RC\_Channel/RC\_Channel\_aux.cpp

GGROUP(rc\_5, "RC5\_", RC\_Channel\_aux),

// @Group: RC6\_

// @Path: ../libraries/RC\_Channel/RC\_Channel\_aux.cpp

GGROUP(rc\_6, "RC6\_", RC\_Channel\_aux),

// @Group: RC7\_

// @Path: ../libraries/RC\_Channel/RC\_Channel\_aux.cpp

GGROUP(rc\_7, "RC7\_", RC\_Channel\_aux),

// @Group: RC8\_

// @Path: ../libraries/RC\_Channel/RC\_Channel\_aux.cpp

GGROUP(rc\_8, "RC8\_", RC\_Channel\_aux),

#if MOUNT == ENABLED

// @Group: RC10\_

// @Path: ../libraries/RC\_Channel/RC\_Channel\_aux.cpp

GGROUP(rc\_10, "RC10\_", RC\_Channel\_aux),

// @Group: RC11\_

// @Path: ../libraries/RC\_Channel/RC\_Channel\_aux.cpp

GGROUP(rc\_11, "RC11\_", RC\_Channel\_aux),

#endif

// @Param: RC\_SPEED

// @DisplayName: ESC Update Speed

// @Description: This is the speed in Hertz that your ESCs will receive updates

// @Units: Hertz (Hz)

// @Values: 125,400,490

// @User: Advanced

GSCALAR(rc\_speed, "RC\_SPEED", RC\_FAST\_SPEED),

// @Param: ACRO\_P

// @DisplayName: Acro P gain

// @Description: Used to convert pilot roll, pitch and yaw input into a dssired rate of rotation in ACRO mode. Higher values mean faster rate of rotation.

// @Range: 1 10

// @User: Standard

GSCALAR(acro\_p, "ACRO\_P", ACRO\_P),

// @Param: AXIS\_ENABLE

// @DisplayName: Acro Axis

// @Description: Used to control whether acro mode actively maintains the current angle when control sticks are released (Enabled = maintains current angle)

// @Values: 0:Disabled, 1:Enabled

// @User: Standard

GSCALAR(axis\_enabled, "AXIS\_ENABLE", AXIS\_LOCK\_ENABLED),

// @Param: ACRO\_BAL\_ROLL

// @DisplayName: Acro Balance Roll

// @Description: rate at which roll angle returns to level in acro mode

// @Range: 0 300

// @Increment: 1

// @User: Advanced

GSCALAR(acro\_balance\_roll, "ACRO\_BAL\_ROLL", ACRO\_BALANCE\_ROLL),

// @Param: ACRO\_BAL\_PITCH

// @DisplayName: Acro Balance Pitch

// @Description: rate at which pitch angle returns to level in acro mode

// @Range: 0 300

// @Increment: 1

// @User: Advanced

GSCALAR(acro\_balance\_pitch, "ACRO\_BAL\_PITCH", ACRO\_BALANCE\_PITCH),

// @Param: ACRO\_TRAINER

// @DisplayName: Acro Trainer Enabled

// @Description: Set to 1 (Enabled) to make roll return to within 45 degrees of level automatically

// @Values: 0:Disabled,1:Enabled

// @User: Advanced

GSCALAR(acro\_trainer\_enabled, "ACRO\_TRAINER", ACRO\_TRAINER\_ENABLED),

// @Param: LED\_MODE

// @DisplayName: Copter LED Mode

// @Description: bitmap to control the copter led mode

// @Values: 0:Disabled,1:Enable,2:GPS On,4:Aux,8:Buzzer,16:Oscillate,32:Nav Blink,64:GPS Nav Blink

// @User: Standard

GSCALAR(copter\_leds\_mode, "LED\_MODE", 9),

// PID controller

//---------------

GGROUP(pid\_rate\_roll, "RATE\_RLL\_", AC\_PID),

GGROUP(pid\_rate\_pitch, "RATE\_PIT\_", AC\_PID),

GGROUP(pid\_rate\_yaw, "RATE\_YAW\_", AC\_PID),

GGROUP(pid\_loiter\_rate\_lat, "LOITER\_LAT\_", AC\_PID),

GGROUP(pid\_loiter\_rate\_lon, "LOITER\_LON\_", AC\_PID),

GGROUP(pid\_nav\_lat, "NAV\_LAT\_", AC\_PID),

GGROUP(pid\_nav\_lon, "NAV\_LON\_", AC\_PID),

GGROUP(pid\_throttle, "THR\_RATE\_", AC\_PID),

GGROUP(pid\_throttle\_accel,"THR\_ACCEL\_", AC\_PID),

GGROUP(pid\_optflow\_roll, "OF\_RLL\_", AC\_PID),

GGROUP(pid\_optflow\_pitch, "OF\_PIT\_", AC\_PID),

// PI controller

//--------------

GGROUP(pi\_stabilize\_roll, "STB\_RLL\_", APM\_PI),

GGROUP(pi\_stabilize\_pitch, "STB\_PIT\_", APM\_PI),

GGROUP(pi\_stabilize\_yaw, "STB\_YAW\_", APM\_PI),

GGROUP(pi\_alt\_hold, "THR\_ALT\_", APM\_PI),

GGROUP(pi\_loiter\_lat, "HLD\_LAT\_", APM\_PI),

GGROUP(pi\_loiter\_lon, "HLD\_LON\_", APM\_PI),

// variables not in the g class which contain EEPROM saved variables

// variables not in the g class which contain EEPROM saved variables

#if CAMERA == ENABLED

// @Group: CAM\_

// @Path: ../libraries/AP\_Camera/AP\_Camera.cpp

GOBJECT(camera, "CAM\_", AP\_Camera),

#endif

// @Group: COMPASS\_

// @Path: ../libraries/AP\_Compass/Compass.cpp

GOBJECT(compass, "COMPASS\_", Compass),

// @Group: INS\_

// @Path: ../libraries/AP\_InertialSensor/AP\_InertialSensor.cpp

#if HIL\_MODE == HIL\_MODE\_DISABLED

GOBJECT(ins, "INS\_", AP\_InertialSensor),

#endif

#if INERTIAL\_NAV\_XY == ENABLED || INERTIAL\_NAV\_Z == ENABLED

// @Group: INAV\_

// @Path: ../libraries/AP\_InertialNav/AP\_InertialNav.cpp

GOBJECT(inertial\_nav, "INAV\_", AP\_InertialNav),

#endif

GOBJECT(gcs0, "SR0\_", GCS\_MAVLINK),

GOBJECT(gcs3, "SR3\_", GCS\_MAVLINK),

// @Group: AHRS\_

// @Path: ../libraries/AP\_AHRS/AP\_AHRS.cpp

GOBJECT(ahrs, "AHRS\_", AP\_AHRS),

#if MOUNT == ENABLED

// @Group: MNT\_

// @Path: ../libraries/AP\_Mount/AP\_Mount.cpp

GOBJECT(camera\_mount, "MNT\_", AP\_Mount),

#endif

#if MOUNT2 == ENABLED

// @Group: MNT2\_

// @Path: ../libraries/AP\_Mount/AP\_Mount.cpp

GOBJECT(camera\_mount2, "MNT2\_", AP\_Mount),

#endif

#ifdef DESKTOP\_BUILD

GOBJECT(sitl, "SIM\_", SITL),

#endif

#if AP\_LIMITS == ENABLED

//@Group: LIM\_

//@Path: ../libraries/AP\_Limits/AP\_Limits.cpp,../libraries/AP\_Limits/AP\_Limit\_GPSLock.cpp, ../libraries/AP\_Limits/AP\_Limit\_Geofence.cpp, ../libraries/AP\_Limits/AP\_Limit\_Altitude.cpp, ../libraries/AP\_Limits/AP\_Limit\_Module.cpp

GOBJECT(limits, "LIM\_", AP\_Limits),

GOBJECT(gpslock\_limit, "LIM\_", AP\_Limit\_GPSLock),

GOBJECT(geofence\_limit, "LIM\_", AP\_Limit\_Geofence),

GOBJECT(altitude\_limit, "LIM\_", AP\_Limit\_Altitude),

#endif

#if FRAME\_CONFIG == HELI\_FRAME

// @Group: H\_

// @Path: ../libraries/AP\_Motors/AP\_MotorsHeli.cpp

GOBJECT(motors, "H\_", AP\_MotorsHeli),

#else

GOBJECT(motors, "MOT\_", AP\_Motors),

#endif

AP\_VAREND

};

static void load\_parameters(void)

{

// change the default for the AHRS\_GPS\_GAIN for ArduCopter

// if it hasn't been set by the user

if (!ahrs.gps\_gain.load()) {

ahrs.gps\_gain.set\_and\_save(1.0);

}

// setup different AHRS gains for ArduCopter than the default

// but allow users to override in their config

if (!ahrs.\_kp.load()) {

ahrs.\_kp.set\_and\_save(0.1);

}

if (!ahrs.\_kp\_yaw.load()) {

ahrs.\_kp\_yaw.set\_and\_save(0.1);

}

#if SECONDARY\_DMP\_ENABLED == ENABLED

if (!ahrs2.\_kp.load()) {

ahrs2.\_kp.set(0.1);

}

if (!ahrs2.\_kp\_yaw.load()) {

ahrs2.\_kp\_yaw.set(0.1);

}

#endif

if (!g.format\_version.load() ||

g.format\_version != Parameters::k\_format\_version) {

// erase all parameters

cliSerial->printf\_P(PSTR("Firmware change: erasing EEPROM...\n"));

AP\_Param::erase\_all();

// save the current format version

g.format\_version.set\_and\_save(Parameters::k\_format\_version);

default\_dead\_zones();

cliSerial->println\_P(PSTR("done."));

} else {

uint32\_t before = micros();

// Load all auto-loaded EEPROM variables

AP\_Param::load\_all();

cliSerial->printf\_P(PSTR("load\_all took %luus\n"), micros() - before);

}

}

## UserCode

// agmatthews USERHOOKS

void userhook\_init()

{

// put your initialisation code here

}

void userhook\_50Hz()

{

// put your 50Hz code here

}

## UserVariables.h

// agmatthews USERHOOKS

// user defined variables

// example variables used in Wii camera testing - replace with your own

// variables

#if WII\_CAMERA == 1

WiiCamera ircam;

int WiiRange=0;

int WiiRotation=0;

int WiiDisplacementX=0;

int WiiDisplacementY=0;

#endif

## commands

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

static void init\_commands()

{

g.command\_index = NO\_COMMAND;

command\_nav\_index = NO\_COMMAND;

command\_cond\_index = NO\_COMMAND;

prev\_nav\_index = NO\_COMMAND;

command\_cond\_queue.id = NO\_COMMAND;

command\_nav\_queue.id = NO\_COMMAND;

ap.fast\_corner = false;

reset\_desired\_speed();

}

// Getters

// -------

static struct Location get\_cmd\_with\_index(int i)

{

struct Location temp;

// Find out proper location in memory by using the start\_byte position + the index

// --------------------------------------------------------------------------------

if (i >= g.command\_total) {

// we do not have a valid command to load

// return a WP with a "Blank" id

temp.id = CMD\_BLANK;

// no reason to carry on

return temp;

}else{

// we can load a command, we don't process it yet

// read WP position

uintptr\_t mem = (WP\_START\_BYTE) + (i \* WP\_SIZE);

temp.id = eeprom\_read\_byte((uint8\_t\*)mem);

mem++;

temp.options = eeprom\_read\_byte((uint8\_t\*)mem);

mem++;

temp.p1 = eeprom\_read\_byte((uint8\_t\*)mem);

mem++;

temp.alt = eeprom\_read\_dword((uint32\_t\*)mem); // alt is stored in CM! Alt is stored relative!

mem += 4;

temp.lat = eeprom\_read\_dword((uint32\_t\*)mem); // lat is stored in decimal \* 10,000,000

mem += 4;

temp.lng = eeprom\_read\_dword((uint32\_t\*)mem); // lon is stored in decimal \* 10,000,000

}

// Add on home altitude if we are a nav command (or other command with altitude) and stored alt is relative

//if((temp.id < MAV\_CMD\_NAV\_LAST || temp.id == MAV\_CMD\_CONDITION\_CHANGE\_ALT) && temp.options & MASK\_OPTIONS\_RELATIVE\_ALT){

//temp.alt += home.alt;

//}

if(temp.options & WP\_OPTION\_RELATIVE) {

// If were relative, just offset from home

temp.lat += home.lat;

temp.lng += home.lng;

}

return temp;

}

// Setters

// -------

static void set\_cmd\_with\_index(struct Location temp, int i)

{

i = constrain(i, 0, g.command\_total.get());

//cliSerial->printf("set\_command: %d with id: %d\n", i, temp.id);

// store home as 0 altitude!!!

// Home is always a MAV\_CMD\_NAV\_WAYPOINT (16)

if (i == 0) {

temp.alt = 0;

temp.id = MAV\_CMD\_NAV\_WAYPOINT;

}

uintptr\_t mem = WP\_START\_BYTE + (i \* WP\_SIZE);

eeprom\_write\_byte((uint8\_t \*) mem, temp.id);

mem++;

eeprom\_write\_byte((uint8\_t \*) mem, temp.options);

mem++;

eeprom\_write\_byte((uint8\_t \*) mem, temp.p1);

mem++;

eeprom\_write\_dword((uint32\_t \*) mem, temp.alt); // Alt is stored in CM!

mem += 4;

eeprom\_write\_dword((uint32\_t \*) mem, temp.lat); // Lat is stored in decimal degrees \* 10^7

mem += 4;

eeprom\_write\_dword((uint32\_t \*) mem, temp.lng); // Long is stored in decimal degrees \* 10^7

// Make sure our WP\_total

if(g.command\_total < (i+1))

g.command\_total.set\_and\_save(i+1);

}

static int32\_t get\_RTL\_alt()

{

if(g.rtl\_altitude <= 0) {

return min(current\_loc.alt, RTL\_ALT\_MAX);

}else if (g.rtl\_altitude < current\_loc.alt) {

return min(current\_loc.alt, RTL\_ALT\_MAX);

}else{

return g.rtl\_altitude;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// This function sets the waypoint and modes for Return to Launch

// It's not currently used

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

\* This function sets the next waypoint command

\* It precalculates all the necessary stuff.

\*/

static void set\_next\_WP(struct Location \*wp)

{

// copy the current WP into the OldWP slot

// ---------------------------------------

if (next\_WP.lat == 0 || command\_nav\_index <= 1) {

prev\_WP = current\_loc;

}else{

if (get\_distance\_cm(&current\_loc, &next\_WP) < 500)

prev\_WP = next\_WP;

else

prev\_WP = current\_loc;

}

// Load the next\_WP slot

// ---------------------

next\_WP.options = wp->options;

next\_WP.lat = wp->lat;

next\_WP.lng = wp->lng;

// Set new target altitude so we can track it for climb\_rate

// To-Do: remove the restriction on negative altitude targets (after testing)

set\_new\_altitude(max(wp->alt,100));

// this is handy for the groundstation

// -----------------------------------

wp\_distance = get\_distance\_cm(&current\_loc, &next\_WP);

wp\_bearing = get\_bearing\_cd(&prev\_WP, &next\_WP);

// calc the location error:

calc\_location\_error(&next\_WP);

// to check if we have missed the WP

// ---------------------------------

original\_wp\_bearing = wp\_bearing;

}

// run this at setup on the ground

// -------------------------------

static void init\_home()

{

set\_home\_is\_set(true);

home.id = MAV\_CMD\_NAV\_WAYPOINT;

home.lng = g\_gps->longitude; // Lon \* 10\*\*7

home.lat = g\_gps->latitude; // Lat \* 10\*\*7

home.alt = 0; // Home is always 0

// Save Home to EEPROM

// -------------------

// no need to save this to EPROM

set\_cmd\_with\_index(home, 0);

#if INERTIAL\_NAV\_XY == ENABLED

// set inertial nav's home position

inertial\_nav.set\_current\_position(g\_gps->longitude, g\_gps->latitude);

#endif

if (g.log\_bitmask & MASK\_LOG\_CMD)

Log\_Write\_Cmd(0, &home);

// update navigation scalers. used to offset the shrinking longitude as we go towards the poles

float rads = (fabs((float)home.lat)/t7) \* 0.0174532925;

scaleLongDown = cos(rads);

scaleLongUp = 1.0f/cos(rads);

// Save prev loc this makes the calcs look better before commands are loaded

prev\_WP = home;

// Load home for a default guided\_WP

// -------------

guided\_WP = home;

guided\_WP.alt += g.rtl\_altitude;

}

## commands\_logic

/// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Command Event Handlers

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// process\_nav\_command - main switch statement to initiate the next nav command in the command\_nav\_queue

static void process\_nav\_command()

{

switch(command\_nav\_queue.id) {

case MAV\_CMD\_NAV\_TAKEOFF: // 22

do\_takeoff();

break;

case MAV\_CMD\_NAV\_WAYPOINT: // 16 Navigate to Waypoint

do\_nav\_wp();

break;

case MAV\_CMD\_NAV\_LAND: // 21 LAND to Waypoint

do\_land();

break;

case MAV\_CMD\_NAV\_LOITER\_UNLIM: // 17 Loiter indefinitely

do\_loiter\_unlimited();

break;

case MAV\_CMD\_NAV\_LOITER\_TURNS: //18 Loiter N Times

do\_loiter\_turns();

break;

case MAV\_CMD\_NAV\_LOITER\_TIME: // 19

do\_loiter\_time();

break;

case MAV\_CMD\_NAV\_RETURN\_TO\_LAUNCH: //20

do\_RTL();

break;

// point the copter and camera at a region of interest (ROI)

case MAV\_CMD\_NAV\_ROI: // 80

do\_nav\_roi();

break;

default:

break;

}

}

static void process\_cond\_command()

{

switch(command\_cond\_queue.id) {

case MAV\_CMD\_CONDITION\_DELAY: // 112

do\_wait\_delay();

break;

case MAV\_CMD\_CONDITION\_DISTANCE: // 114

do\_within\_distance();

break;

case MAV\_CMD\_CONDITION\_CHANGE\_ALT: // 113

do\_change\_alt();

break;

case MAV\_CMD\_CONDITION\_YAW: // 115

do\_yaw();

break;

default:

break;

}

}

static void process\_now\_command()

{

switch(command\_cond\_queue.id) {

case MAV\_CMD\_DO\_JUMP: // 177

do\_jump();

break;

case MAV\_CMD\_DO\_CHANGE\_SPEED: // 178

do\_change\_speed();

break;

case MAV\_CMD\_DO\_SET\_HOME: // 179

do\_set\_home();

break;

case MAV\_CMD\_DO\_SET\_SERVO: // 183

do\_set\_servo();

break;

case MAV\_CMD\_DO\_SET\_RELAY: // 181

do\_set\_relay();

break;

case MAV\_CMD\_DO\_REPEAT\_SERVO: // 184

do\_repeat\_servo();

break;

case MAV\_CMD\_DO\_REPEAT\_RELAY: // 182

do\_repeat\_relay();

break;

#if CAMERA == ENABLED

case MAV\_CMD\_DO\_CONTROL\_VIDEO: // Control on-board camera capturing. |Camera ID (-1 for all)| Transmission: 0: disabled, 1: enabled compressed, 2: enabled raw| Transmission mode: 0: video stream, >0: single images every n seconds (decimal)| Recording: 0: disabled, 1: enabled compressed, 2: enabled raw| Empty| Empty| Empty|

break;

case MAV\_CMD\_DO\_DIGICAM\_CONFIGURE: // Mission command to configure an on-board camera controller system. |Modes: P, TV, AV, M, Etc| Shutter speed: Divisor number for one second| Aperture: F stop number| ISO number e.g. 80, 100, 200, Etc| Exposure type enumerator| Command Identity| Main engine cut-off time before camera trigger in seconds/10 (0 means no cut-off)|

break;

case MAV\_CMD\_DO\_DIGICAM\_CONTROL: // Mission command to control an on-board camera controller system. |Session control e.g. show/hide lens| Zoom's absolute position| Zooming step value to offset zoom from the current position| Focus Locking, Unlocking or Re-locking| Shooting Command| Command Identity| Empty|

do\_take\_picture();

break;

#endif

#if MOUNT == ENABLED

case MAV\_CMD\_DO\_MOUNT\_CONFIGURE: // Mission command to configure a camera mount |Mount operation mode (see MAV\_CONFIGURE\_MOUNT\_MODE enum)| stabilize roll? (1 = yes, 0 = no)| stabilize pitch? (1 = yes, 0 = no)| stabilize yaw? (1 = yes, 0 = no)| Empty| Empty| Empty|

camera\_mount.configure\_cmd();

break;

case MAV\_CMD\_DO\_MOUNT\_CONTROL: // Mission command to control a camera mount |pitch(deg\*100) or lat, depending on mount mode.| roll(deg\*100) or lon depending on mount mode| yaw(deg\*100) or alt (in cm) depending on mount mode| Empty| Empty| Empty| Empty|

camera\_mount.control\_cmd();

break;

#endif

default:

// do nothing with unrecognized MAVLink messages

break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Verify command Handlers

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// verify\_must - switch statement to ensure the active navigation command is progressing

// returns true once the active navigation command completes successfully

static bool verify\_must()

{

switch(command\_nav\_queue.id) {

case MAV\_CMD\_NAV\_TAKEOFF:

return verify\_takeoff();

break;

case MAV\_CMD\_NAV\_WAYPOINT:

return verify\_nav\_wp();

break;

case MAV\_CMD\_NAV\_LAND:

return verify\_land();

break;

case MAV\_CMD\_NAV\_LOITER\_UNLIM:

return verify\_loiter\_unlimited();

break;

case MAV\_CMD\_NAV\_LOITER\_TURNS:

return verify\_loiter\_turns();

break;

case MAV\_CMD\_NAV\_LOITER\_TIME:

return verify\_loiter\_time();

break;

case MAV\_CMD\_NAV\_RETURN\_TO\_LAUNCH:

return verify\_RTL();

break;

case MAV\_CMD\_NAV\_ROI: // 80

return verify\_nav\_roi();

break;

default:

//gcs\_send\_text\_P(SEVERITY\_HIGH,PSTR("<verify\_must: default> No current Must commands"));

return false;

break;

}

}

// verify\_may - switch statement to ensure the active conditional command is progressing

// returns true once the active conditional command completes successfully

static bool verify\_may()

{

switch(command\_cond\_queue.id) {

case MAV\_CMD\_CONDITION\_DELAY:

return verify\_wait\_delay();

break;

case MAV\_CMD\_CONDITION\_DISTANCE:

return verify\_within\_distance();

break;

case MAV\_CMD\_CONDITION\_CHANGE\_ALT:

return verify\_change\_alt();

break;

case MAV\_CMD\_CONDITION\_YAW:

return verify\_yaw();

break;

default:

//gcs\_send\_text\_P(SEVERITY\_HIGH,PSTR("<verify\_must: default> No current May commands"));

return false;

break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// do\_RTL - start Return-to-Launch

static void do\_RTL(void)

{

// set rtl state

rtl\_state = RTL\_STATE\_INITIAL\_CLIMB;

// set roll, pitch and yaw modes

set\_roll\_pitch\_mode(RTL\_RP);

set\_yaw\_mode(YAW\_HOLD); // first stage of RTL is the initial climb so just hold current yaw

set\_throttle\_mode(RTL\_THR);

// set navigation mode

wp\_control = LOITER\_MODE;

// initial climb starts at current location

set\_next\_WP(&current\_loc);

// override altitude to RTL altitude

set\_new\_altitude(get\_RTL\_alt());

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Nav (Must) commands

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// do\_takeoff - initiate takeoff navigation command

static void do\_takeoff()

{

wp\_control = LOITER\_MODE;

// Start with current location

Location temp = current\_loc;

// alt is always relative

temp.alt = command\_nav\_queue.alt;

// Set our waypoint

set\_next\_WP(&temp);

// set our yaw mode

set\_yaw\_mode(YAW\_HOLD);

// prevent flips

reset\_I\_all();

}

// do\_nav\_wp - initiate move to next waypoint

static void do\_nav\_wp()

{

wp\_control = WP\_MODE;

set\_next\_WP(&command\_nav\_queue);

// this is our bitmask to verify we have met all conditions to move on

wp\_verify\_byte = 0;

// if no alt requirement in the waypoint, set the altitude achieved bit of wp\_verify\_byte

if((next\_WP.options & WP\_OPTION\_ALT\_REQUIRED) == false) {

wp\_verify\_byte |= NAV\_ALTITUDE;

}

// this will be used to remember the time in millis after we reach or pass the WP.

loiter\_time = 0;

// this is the delay, stored in seconds and expanded to millis

loiter\_time\_max = command\_nav\_queue.p1 \* 1000;

// reset control of yaw to default

if( g.yaw\_override\_behaviour == YAW\_OVERRIDE\_BEHAVIOUR\_AT\_NEXT\_WAYPOINT ) {

set\_yaw\_mode(AUTO\_YAW);

}

}

// do\_land - initiate landing procedure

static void do\_land()

{

// hold at our current location

set\_next\_WP(&current\_loc);

wp\_control = LOITER\_MODE;

// hold current heading

set\_yaw\_mode(YAW\_HOLD);

set\_throttle\_mode(THROTTLE\_LAND);

}

static void do\_loiter\_unlimited()

{

wp\_control = LOITER\_MODE;

//cliSerial->println("dloi ");

if(command\_nav\_queue.lat == 0) {

set\_next\_WP(&current\_loc);

wp\_control = LOITER\_MODE;

}else{

set\_next\_WP(&command\_nav\_queue);

wp\_control = WP\_MODE;

}

}

// do\_loiter\_turns - initiate moving in a circle

static void do\_loiter\_turns()

{

wp\_control = CIRCLE\_MODE;

if(command\_nav\_queue.lat == 0) {

// allow user to specify just the altitude

if(command\_nav\_queue.alt > 0) {

current\_loc.alt = command\_nav\_queue.alt;

}

set\_next\_WP(&current\_loc);

}else{

set\_next\_WP(&command\_nav\_queue);

}

circle\_WP = next\_WP;

loiter\_total = command\_nav\_queue.p1 \* 360;

loiter\_sum = 0;

old\_wp\_bearing = wp\_bearing;

circle\_angle = wp\_bearing + 18000;

circle\_angle = wrap\_360(circle\_angle);

circle\_angle \*= RADX100;

}

// do\_loiter\_time - initiate loitering at a point for a given time period

static void do\_loiter\_time()

{

if(command\_nav\_queue.lat == 0) {

wp\_control = LOITER\_MODE;

loiter\_time = millis();

set\_next\_WP(&current\_loc);

}else{

wp\_control = WP\_MODE;

set\_next\_WP(&command\_nav\_queue);

}

loiter\_time\_max = command\_nav\_queue.p1 \* 1000; // units are (seconds)

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Verify Nav (Must) commands

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// verify\_takeoff - check if we have completed the takeoff

static bool verify\_takeoff()

{

// wait until we are ready!

if(g.rc\_3.control\_in == 0) {

return false;

}

// are we above our target altitude?

return (alt\_change\_flag == REACHED\_ALT);

}

// verify\_land - returns true if landing has been completed

static bool verify\_land()

{

// rely on THROTTLE\_LAND mode to correctly update landing status

return ap.land\_complete;

}

// verify\_nav\_wp - check if we have reached the next way point

static bool verify\_nav\_wp()

{

// Altitude checking

if(next\_WP.options & WP\_OPTION\_ALT\_REQUIRED) {

// we desire a certain minimum altitude

if(alt\_change\_flag == REACHED\_ALT) {

// we have reached that altitude

wp\_verify\_byte |= NAV\_ALTITUDE;

}

}

// Did we pass the WP? // Distance checking

if((wp\_distance <= (g.waypoint\_radius \* 100)) || check\_missed\_wp()) {

// if we have a distance calc error, wp\_distance may be less than 0

if(wp\_distance > 0) {

wp\_verify\_byte |= NAV\_LOCATION;

if(loiter\_time == 0) {

loiter\_time = millis();

}

}

}

// Hold at Waypoint checking, we cant move on until this is OK

if(wp\_verify\_byte & NAV\_LOCATION) {

// we have reached our goal

// loiter at the WP

wp\_control = LOITER\_MODE;

if ((millis() - loiter\_time) > loiter\_time\_max) {

wp\_verify\_byte |= NAV\_DELAY;

//gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("verify\_must: LOITER time complete"));

//cliSerial->println("vlt done");

}

}

if( wp\_verify\_byte == (NAV\_LOCATION | NAV\_ALTITUDE | NAV\_DELAY) ) {

gcs\_send\_text\_fmt(PSTR("Reached Command #%i"),command\_nav\_index);

wp\_verify\_byte = 0;

copter\_leds\_nav\_blink = 15; // Cause the CopterLEDs to blink three times to indicate waypoint reached

return true;

}else{

return false;

}

}

static bool verify\_loiter\_unlimited()

{

if(wp\_control == WP\_MODE && wp\_distance <= (g.waypoint\_radius \* 100)) {

// switch to position hold

wp\_control = LOITER\_MODE;

}

return false;

}

// verify\_loiter\_time - check if we have loitered long enough

static bool verify\_loiter\_time()

{

if(wp\_control == LOITER\_MODE) {

if ((millis() - loiter\_time) > loiter\_time\_max) {

return true;

}

}

if(wp\_control == WP\_MODE && wp\_distance <= (g.waypoint\_radius \* 100)) {

// reset our loiter time

loiter\_time = millis();

// switch to position hold

wp\_control = LOITER\_MODE;

}

return false;

}

// verify\_loiter\_turns - check if we have circled the point enough

static bool verify\_loiter\_turns()

{

//cliSerial->printf("loiter\_sum: %d \n", loiter\_sum);

// have we rotated around the center enough times?

// -----------------------------------------------

if(abs(loiter\_sum) > loiter\_total) {

loiter\_total = 0;

loiter\_sum = 0;

//gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("verify\_must: LOITER orbits complete"));

// clear the command queue;

return true;

}

return false;

}

// verify\_RTL - handles any state changes required to implement RTL

// do\_RTL should have been called once first to initialise all variables

// returns true with RTL has completed successfully

static bool verify\_RTL()

{

bool retval = false;

switch( rtl\_state ) {

case RTL\_STATE\_INITIAL\_CLIMB:

// rely on verify\_altitude function to update alt\_change\_flag when we've reached the target

if(alt\_change\_flag == REACHED\_ALT || alt\_change\_flag == DESCENDING) {

// Set navigation target to home

set\_next\_WP(&home);

// override target altitude to RTL altitude

set\_new\_altitude(get\_RTL\_alt());

// set navigation mode

wp\_control = WP\_MODE;

// set yaw mode

set\_yaw\_mode(RTL\_YAW);

// advance to next rtl state

rtl\_state = RTL\_STATE\_RETURNING\_HOME;

}

break;

case RTL\_STATE\_RETURNING\_HOME:

// if we've reached home initiate loiter

if (wp\_distance <= g.waypoint\_radius \* 100 || check\_missed\_wp()) {

rtl\_state = RTL\_STATE\_LOITERING\_AT\_HOME;

wp\_control = LOITER\_MODE;

// set loiter timer

rtl\_loiter\_start\_time = millis();

// give pilot back control of yaw

set\_yaw\_mode(YAW\_HOLD);

}

break;

case RTL\_STATE\_LOITERING\_AT\_HOME:

// check if we've loitered long enough

if( millis() - rtl\_loiter\_start\_time > (uint32\_t)g.rtl\_loiter\_time.get() ) {

// initiate landing or descent

if(g.rtl\_alt\_final == 0 || ap.failsafe) {

// land

do\_land();

// override landing location (do\_land defaults to current location)

next\_WP.lat = home.lat;

next\_WP.lng = home.lng;

// update RTL state

rtl\_state = RTL\_STATE\_LAND;

}else{

// descend

if(current\_loc.alt > g.rtl\_alt\_final) {

set\_new\_altitude(g.rtl\_alt\_final);

}

// update RTL state

rtl\_state = RTL\_STATE\_FINAL\_DESCENT;

}

}

break;

case RTL\_STATE\_FINAL\_DESCENT:

// rely on altitude check to confirm we have reached final altitude

if(current\_loc.alt <= g.rtl\_alt\_final || alt\_change\_flag == REACHED\_ALT) {

// switch to regular loiter mode

set\_mode(LOITER);

// override location and altitude

set\_next\_WP(&home);

// override altitude to RTL altitude

set\_new\_altitude(g.rtl\_alt\_final);

retval = true;

}

break;

case RTL\_STATE\_LAND:

// rely on verify\_land to return correct status

retval = verify\_land();

break;

default:

// this should never happen

// TO-DO: log an error

retval = true;

break;

}

// true is returned if we've successfully completed RTL

return retval;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Condition (May) commands

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void do\_wait\_delay()

{

//cliSerial->print("dwd ");

condition\_start = millis();

condition\_value = command\_cond\_queue.lat \* 1000; // convert to milliseconds

//cliSerial->println(condition\_value,DEC);

}

static void do\_change\_alt()

{

Location temp = next\_WP;

condition\_start = current\_loc.alt;

//condition\_value = command\_cond\_queue.alt;

temp.alt = command\_cond\_queue.alt;

set\_next\_WP(&temp);

}

static void do\_within\_distance()

{

condition\_value = command\_cond\_queue.lat \* 100;

}

static void do\_yaw()

{

// get final angle, 1 = Relative, 0 = Absolute

if( command\_cond\_queue.lng == 0 ) {

// absolute angle

yaw\_look\_at\_heading = wrap\_360(command\_cond\_queue.alt \* 100);

}else{

// relative angle

yaw\_look\_at\_heading = wrap\_360(nav\_yaw + command\_cond\_queue.alt \* 100);

}

// get turn speed

if( command\_cond\_queue.lat == 0 ) {

// default to regular auto slew rate

yaw\_look\_at\_heading\_slew = AUTO\_YAW\_SLEW\_RATE;

}else{

int32\_t turn\_rate = (wrap\_180(yaw\_look\_at\_heading - nav\_yaw) / 100) / command\_cond\_queue.lat;

yaw\_look\_at\_heading\_slew = constrain(turn\_rate, 1, 360); // deg / sec

}

// set yaw mode

set\_yaw\_mode(YAW\_LOOK\_AT\_HEADING);

// TO-DO: restore support for clockwise / counter clockwise rotation held in command\_cond\_queue.p1

// command\_cond\_queue.p1; // 0 = undefined, 1 = clockwise, -1 = counterclockwise

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Verify Condition (May) commands

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static bool verify\_wait\_delay()

{

//cliSerial->print("vwd");

if ((unsigned)(millis() - condition\_start) > (unsigned)condition\_value) {

//cliSerial->println("y");

condition\_value = 0;

return true;

}

//cliSerial->println("n");

return false;

}

static bool verify\_change\_alt()

{

//cliSerial->printf("change\_alt, ca:%d, na:%d\n", (int)current\_loc.alt, (int)next\_WP.alt);

if ((int32\_t)condition\_start < next\_WP.alt) {

// we are going higer

if(current\_loc.alt > next\_WP.alt) {

return true;

}

}else{

// we are going lower

if(current\_loc.alt < next\_WP.alt) {

return true;

}

}

return false;

}

static bool verify\_within\_distance()

{

//cliSerial->printf("cond dist :%d\n", (int)condition\_value);

if (wp\_distance < condition\_value) {

condition\_value = 0;

return true;

}

return false;

}

// verify\_yaw - return true if we have reached the desired heading

static bool verify\_yaw()

{

if( labs(wrap\_180(ahrs.yaw\_sensor-yaw\_look\_at\_heading)) <= 200 ) {

return true;

}else{

return false;

}

}

// verify\_nav\_roi - verifies that actions required by MAV\_CMD\_NAV\_ROI have completed

// we assume the camera command has been successfully implemented by the do\_nav\_roi command

// so all we need to check is whether we needed to yaw the copter (due to the mount type) and

// whether that yaw has completed

// TO-DO: add support for other features of MAV\_NAV\_ROI including pointing at a given waypoint

static bool verify\_nav\_roi()

{

#if MOUNT == ENABLED

// check if mount type requires us to rotate the quad

if( camera\_mount.get\_mount\_type() != AP\_Mount::k\_pan\_tilt && camera\_mount.get\_mount\_type() != AP\_Mount::k\_pan\_tilt\_roll ) {

// ensure yaw has gotten to within 2 degrees of the target

if( labs(wrap\_180(ahrs.yaw\_sensor-yaw\_look\_at\_WP\_bearing)) <= 200 ) {

return true;

}else{

return false;

}

}else{

// if no rotation required, assume the camera instruction was implemented immediately

return true;

}

#else

// if we have no camera mount simply check we've reached the desired yaw

// ensure yaw has gotten to within 2 degrees of the target

if( labs(wrap\_180(ahrs.yaw\_sensor-yaw\_look\_at\_WP\_bearing)) <= 200 ) {

return true;

}else{

return false;

}

#endif

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Do (Now) commands

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void do\_change\_speed()

{

g.waypoint\_speed\_max = command\_cond\_queue.p1 \* 100;

}

static void do\_jump()

{

// Used to track the state of the jump command in Mission scripting

// -10 is a value that means the register is unused

// when in use, it contains the current remaining jumps

static int8\_t jump = -10; // used to track loops in jump command

//cliSerial->printf("do Jump: %d\n", jump);

if(jump == -10) {

//cliSerial->printf("Fresh Jump\n");

// we use a locally stored index for jump

jump = command\_cond\_queue.lat;

}

//cliSerial->printf("Jumps left: %d\n",jump);

if(jump > 0) {

//cliSerial->printf("Do Jump to %d\n",command\_cond\_queue.p1);

jump--;

change\_command(command\_cond\_queue.p1);

} else if (jump == 0) {

//cliSerial->printf("Did last jump\n");

// we're done, move along

jump = -11;

} else if (jump == -1) {

//cliSerial->printf("jumpForever\n");

// repeat forever

change\_command(command\_cond\_queue.p1);

}

}

static void do\_set\_home()

{

if(command\_cond\_queue.p1 == 1) {

init\_home();

} else {

home.id = MAV\_CMD\_NAV\_WAYPOINT;

home.lng = command\_cond\_queue.lng; // Lon \* 10\*\*7

home.lat = command\_cond\_queue.lat; // Lat \* 10\*\*7

home.alt = 0;

//home\_is\_set = true;

set\_home\_is\_set(true);

}

}

static void do\_set\_servo()

{

uint8\_t channel\_num = 0xff;

switch( command\_cond\_queue.p1 ) {

case 1:

channel\_num = CH\_1;

break;

case 2:

channel\_num = CH\_2;

break;

case 3:

channel\_num = CH\_3;

break;

case 4:

channel\_num = CH\_4;

break;

case 5:

channel\_num = CH\_5;

break;

case 6:

channel\_num = CH\_6;

break;

case 7:

channel\_num = CH\_7;

break;

case 8:

channel\_num = CH\_8;

break;

case 9:

// not used

break;

case 10:

channel\_num = CH\_10;

break;

case 11:

channel\_num = CH\_11;

break;

}

// send output to channel

if (channel\_num != 0xff) {

APM\_RC.enable\_out(channel\_num);

APM\_RC.OutputCh(channel\_num, command\_cond\_queue.alt);

}

}

static void do\_set\_relay()

{

if (command\_cond\_queue.p1 == 1) {

relay.on();

} else if (command\_cond\_queue.p1 == 0) {

relay.off();

}else{

relay.toggle();

}

}

static void do\_repeat\_servo()

{

event\_id = command\_cond\_queue.p1 - 1;

if(command\_cond\_queue.p1 >= CH\_5 + 1 && command\_cond\_queue.p1 <= CH\_8 + 1) {

event\_timer = 0;

event\_value = command\_cond\_queue.alt;

event\_repeat = command\_cond\_queue.lat \* 2;

event\_delay = command\_cond\_queue.lng \* 500.0; // /2 (half cycle time) \* 1000 (convert to milliseconds)

switch(command\_cond\_queue.p1) {

case CH\_5:

event\_undo\_value = g.rc\_5.radio\_trim;

break;

case CH\_6:

event\_undo\_value = g.rc\_6.radio\_trim;

break;

case CH\_7:

event\_undo\_value = g.rc\_7.radio\_trim;

break;

case CH\_8:

event\_undo\_value = g.rc\_8.radio\_trim;

break;

}

update\_events();

}

}

static void do\_repeat\_relay()

{

event\_id = RELAY\_TOGGLE;

event\_timer = 0;

event\_delay = command\_cond\_queue.lat \* 500.0; // /2 (half cycle time) \* 1000 (convert to milliseconds)

event\_repeat = command\_cond\_queue.alt \* 2;

update\_events();

}

// do\_nav\_roi - starts actions required by MAV\_CMD\_NAV\_ROI

// this involves either moving the camera to point at the ROI (region of interest)

// and possibly rotating the copter to point at the ROI if our mount type does not support a yaw feature

// Note: the ROI should already be in the command\_nav\_queue global variable

// TO-DO: add support for other features of MAV\_NAV\_ROI including pointing at a given waypoint

static void do\_nav\_roi()

{

#if MOUNT == ENABLED

// check if mount type requires us to rotate the quad

if( camera\_mount.get\_mount\_type() != AP\_Mount::k\_pan\_tilt && camera\_mount.get\_mount\_type() != AP\_Mount::k\_pan\_tilt\_roll ) {

yaw\_look\_at\_WP = command\_nav\_queue;

set\_yaw\_mode(YAW\_LOOK\_AT\_LOCATION);

}

// send the command to the camera mount

camera\_mount.set\_roi\_cmd(&command\_nav\_queue);

// TO-DO: expand handling of the do\_nav\_roi to support all modes of the MAVLink. Currently we only handle mode 4 (see below)

// 0: do nothing

// 1: point at next waypoint

// 2: point at a waypoint taken from WP# parameter (2nd parameter?)

// 3: point at a location given by alt, lon, lat parameters

// 4: point at a target given a target id (can't be implmented)

#else

// if we have no camera mount aim the quad at the location

yaw\_look\_at\_WP = command\_nav\_queue;

set\_yaw\_mode(YAW\_LOOK\_AT\_LOCATION);

#endif

}

// do\_take\_picture - take a picture with the camera library

static void do\_take\_picture()

{

#if CAMERA == ENABLED

camera.trigger\_pic();

if (g.log\_bitmask & MASK\_LOG\_CAMERA) {

Log\_Write\_Camera();

}

#endif

}

## commands\_process

/// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

// For changing active command mid-mission

//----------------------------------------

static void change\_command(uint8\_t cmd\_index)

{

//cliSerial->printf("change\_command: %d\n",cmd\_index );

// limit range

cmd\_index = min(g.command\_total - 1, cmd\_index);

// load command

struct Location temp = get\_cmd\_with\_index(cmd\_index);

//cliSerial->printf("loading cmd: %d with id:%d\n", cmd\_index, temp.id);

// verify it's a nav command

if(temp.id > MAV\_CMD\_NAV\_LAST) {

//gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("error: non-Nav cmd"));

}else{

// clear out command queue

init\_commands();

// copy command to the queue

command\_nav\_queue = temp;

command\_nav\_index = cmd\_index;

execute\_nav\_command();

}

}

// update\_commands - initiates new navigation commands if we have completed the previous command

// called by 10 Hz loop

static void update\_commands()

{

//cliSerial->printf("update\_commands: %d\n",increment );

// A: if we do not have any commands there is nothing to do

// B: We have completed the mission, don't redo the mission

// XXX debug

//uint8\_t tmp = g.command\_index.get();

//cliSerial->printf("command\_index %u \n", tmp);

if(g.command\_total <= 1 || g.command\_index >= 255)

return;

if(command\_nav\_queue.id == NO\_COMMAND) {

// Our queue is empty

// fill command queue with a new command if available, or exit mission

// -------------------------------------------------------------------

// find next nav command

int16\_t tmp\_index;

if(command\_nav\_index < g.command\_total) {

// what is the next index for a nav command?

tmp\_index = find\_next\_nav\_index(command\_nav\_index + 1);

if(tmp\_index == -1) {

exit\_mission();

return;

}else{

command\_nav\_index = tmp\_index;

command\_nav\_queue = get\_cmd\_with\_index(command\_nav\_index);

execute\_nav\_command();

}

// try to load the next nav for better speed control

// find\_next\_nav\_index takes the next guess to start the search

tmp\_index = find\_next\_nav\_index(command\_nav\_index + 1);

// Fast corner management

// ----------------------

if(tmp\_index == -1) {

// there are no more commands left

}else{

// we have at least one more cmd left

Location tmp\_loc = get\_cmd\_with\_index(tmp\_index);

if(tmp\_loc.lat == 0) {

ap.fast\_corner = false;

}else{

int32\_t temp = get\_bearing\_cd(&next\_WP, &tmp\_loc) - original\_wp\_bearing;

temp = wrap\_180(temp);

ap.fast\_corner = labs(temp) < 6000;

}

// If we try and stop at a corner, lets reset our desired speed to prevent

// too much jerkyness.

if(false == ap.fast\_corner){

reset\_desired\_speed();

}

}

}else{

// we are out of commands

exit\_mission();

return;

}

}

if(command\_cond\_queue.id == NO\_COMMAND) {

// Our queue is empty

// fill command queue with a new command if available, or do nothing

// -------------------------------------------------------------------

// no nav commands completed yet

if(prev\_nav\_index == NO\_COMMAND)

return;

if(command\_cond\_index >= command\_nav\_index) {

// don't process the fututre

return;

}else if(command\_cond\_index == NO\_COMMAND) {

// start from scratch

// look at command after the most recent completed nav

command\_cond\_index = prev\_nav\_index + 1;

}else{

// we've completed 1 cond, look at next command for another

command\_cond\_index++;

}

if(command\_cond\_index < (g.command\_total -2)) {

// we're OK to load a new command (last command must be a nav command)

command\_cond\_queue = get\_cmd\_with\_index(command\_cond\_index);

if(command\_cond\_queue.id > MAV\_CMD\_CONDITION\_LAST) {

// this is a do now command

process\_now\_command();

// clear command queue

command\_cond\_queue.id = NO\_COMMAND;

}else if(command\_cond\_queue.id > MAV\_CMD\_NAV\_LAST) {

// this is a conditional command

process\_cond\_command();

}else{

// this is a nav command, don't process

// clear the command conditional queue and index

prev\_nav\_index = NO\_COMMAND;

command\_cond\_index = NO\_COMMAND;

command\_cond\_queue.id = NO\_COMMAND;

}

}

}

}

// execute\_nav\_command - performs minor initialisation and logging before next navigation command in the queue is executed

static void execute\_nav\_command(void)

{

// This is what we report to MAVLINK

g.command\_index = command\_nav\_index;

// Save CMD to Log

if(g.log\_bitmask & MASK\_LOG\_CMD)

Log\_Write\_Cmd(g.command\_index, &command\_nav\_queue);

// clear navigation prameters

reset\_nav\_params();

// Act on the new command

process\_nav\_command();

// clear May indexes to force loading of more commands

// existing May commands are tossed.

command\_cond\_index = NO\_COMMAND;

}

// verify\_commands - high level function to check if navigation and conditional commands have completed

// called after GPS navigation update - not constantly

static void verify\_commands(void)

{

if(verify\_must()) {

//cliSerial->printf("verified must cmd %d\n" , command\_nav\_index);

command\_nav\_queue.id = NO\_COMMAND;

// store our most recent executed nav command

prev\_nav\_index = command\_nav\_index;

// Wipe existing conditionals

command\_cond\_index = NO\_COMMAND;

command\_cond\_queue.id = NO\_COMMAND;

}else{

//cliSerial->printf("verified must false %d\n" , command\_nav\_index);

}

if(verify\_may()) {

//cliSerial->printf("verified may cmd %d\n" , command\_cond\_index);

command\_cond\_queue.id = NO\_COMMAND;

}

}

// Finds the next navgation command in EEPROM

static int16\_t find\_next\_nav\_index(int16\_t search\_index)

{

Location tmp;

while(search\_index < g.command\_total) {

tmp = get\_cmd\_with\_index(search\_index);

if(tmp.id <= MAV\_CMD\_NAV\_LAST) {

return search\_index;

}else{

search\_index++;

}

}

return -1;

}

static void exit\_mission()

{

// we are out of commands

g.command\_index = 255;

// if we are on the ground, enter stabilize, else Land

if(ap.land\_complete) {

// we will disarm the motors after landing.

}else{

// If the approach altitude is valid (above 1m), do approach, else land

if(g.rtl\_alt\_final == 0) {

set\_mode(LAND);

}else{

set\_mode(LOITER);

set\_new\_altitude(g.rtl\_alt\_final);

}

}

}

## config.h

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

//

#ifndef \_\_ARDUCOPTER\_CONFIG\_H\_\_

#define \_\_ARDUCOPTER\_CONFIG\_H\_\_

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//

// WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING

//

// DO NOT EDIT this file to adjust your configuration. Create your own

// APM\_Config.h and use APM\_Config.h.example as a reference.

//

// WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING

///

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//

// Default and automatic configuration details.

//

// Notes for maintainers:

//

// - Try to keep this file organised in the same order as APM\_Config.h.example

//

#include "defines.h"

///

/// DO NOT EDIT THIS INCLUDE - if you want to make a local change, make that

/// change in your local copy of APM\_Config.h.

///

#ifdef USE\_CMAKE\_APM\_CONFIG

#include "APM\_Config\_cmake.h" // <== Prefer cmake config if it exists

#else

#include "APM\_Config.h" // <== THIS INCLUDE, DO NOT EDIT IT. EVER.

#endif

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

// HARDWARE CONFIGURATION AND CONNECTIONS

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

// APM HARDWARE

//

#ifndef CONFIG\_APM\_HARDWARE

# define CONFIG\_APM\_HARDWARE APM\_HARDWARE\_APM1

#endif

//////////////////////////////////////////////////////////////////////////////

// APM2 HARDWARE DEFAULTS

//

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

# define CONFIG\_IMU\_TYPE CONFIG\_IMU\_MPU6000

# define CONFIG\_PUSHBUTTON DISABLED

# define CONFIG\_RELAY DISABLED

# define MAG\_ORIENTATION AP\_COMPASS\_APM2\_SHIELD

# define CONFIG\_SONAR\_SOURCE SONAR\_SOURCE\_ANALOG\_PIN

# define MAGNETOMETER ENABLED

# ifdef APM2\_BETA\_HARDWARE

# define CONFIG\_BARO AP\_BARO\_BMP085

# else // APM2 Production Hardware (default)

# define CONFIG\_BARO AP\_BARO\_MS5611

# endif

#endif

//////////////////////////////////////////////////////////////////////////////

// FRAME\_CONFIG

//

#ifndef FRAME\_CONFIG

# define FRAME\_CONFIG QUAD\_FRAME

#endif

#ifndef FRAME\_ORIENTATION

# define FRAME\_ORIENTATION X\_FRAME

#endif

#ifndef TOY\_EDF

# define TOY\_EDF DISABLED

#endif

#ifndef TOY\_MIXER

# define TOY\_MIXER TOY\_LINEAR\_MIXER

#endif

/////////////////////////////////////////////////////////////////////////////////

// Bulk defines for TradHeli

#if FRAME\_CONFIG == HELI\_FRAME

# define RC\_FAST\_SPEED 125

# define RTL\_YAW YAW\_LOOK\_AT\_HOME

# define TILT\_COMPENSATION 5

# define RATE\_INTEGRATOR\_LEAK\_RATE 0.02

# define RATE\_ROLL\_D 0

# define RATE\_PITCH\_D 0

# define HELI\_PITCH\_FF 0

# define HELI\_ROLL\_FF 0

# define HELI\_YAW\_FF 0

# define RC\_FAST\_SPEED 125

# define STABILIZE\_THROTTLE THROTTLE\_MANUAL

# define MPU6K\_FILTER 10

#endif

// optical flow doesn't work in SITL yet

#ifdef DESKTOP\_BUILD

# define OPTFLOW DISABLED

#endif

//////////////////////////////////////////////////////////////////////////////

// IMU Selection

//

#ifndef CONFIG\_IMU\_TYPE

# define CONFIG\_IMU\_TYPE CONFIG\_IMU\_OILPAN

#endif

#ifndef MPU6K\_FILTER

# define MPU6K\_FILTER MPU6K\_DEFAULT\_FILTER

#endif

//////////////////////////////////////////////////////////////////////////////

// ADC Enable - used to eliminate for systems which don't have ADC.

//

#ifndef CONFIG\_ADC

# if CONFIG\_IMU\_TYPE == CONFIG\_IMU\_OILPAN

# define CONFIG\_ADC ENABLED

# else

# define CONFIG\_ADC DISABLED

# endif

#endif

//////////////////////////////////////////////////////////////////////////////

// PWM control

// default RC speed in Hz

#ifndef RC\_FAST\_SPEED

# define RC\_FAST\_SPEED 490

#endif

////////////////////////////////////////////////////////

// LED and IO Pins

//

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM1

# define A\_LED\_PIN 37

# define B\_LED\_PIN 36

# define C\_LED\_PIN 35

# define LED\_ON HIGH

# define LED\_OFF LOW

# define SLIDE\_SWITCH\_PIN 40

# define PUSHBUTTON\_PIN 41

# define USB\_MUX\_PIN -1

# define CLI\_SLIDER\_ENABLED DISABLED

# define OPTFLOW\_CS\_PIN 34

# define BATTERY\_VOLT\_PIN 0 // Battery voltage on A0

# define BATTERY\_CURR\_PIN 1 // Battery current on A1

#elif CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

# define A\_LED\_PIN 27

# define B\_LED\_PIN 26

# define C\_LED\_PIN 25

# define LED\_ON LOW

# define LED\_OFF HIGH

# define SLIDE\_SWITCH\_PIN (-1)

# define PUSHBUTTON\_PIN (-1)

# define CLI\_SLIDER\_ENABLED DISABLED

# define USB\_MUX\_PIN 23

# define OPTFLOW\_CS\_PIN A3

# define BATTERY\_VOLT\_PIN 1 // Battery voltage on A1

# define BATTERY\_CURR\_PIN 2 // Battery current on A2

#endif

////////////////////////////////////////////////////////////////////////////////

// CopterLEDs

//

#ifndef COPTER\_LEDS

#define COPTER\_LEDS ENABLED

#endif

#define COPTER\_LED\_ON HIGH

#define COPTER\_LED\_OFF LOW

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

#define COPTER\_LED\_1 AN4 // Motor or Aux LED

#define COPTER\_LED\_2 AN5 // Motor LED or Beeper

#define COPTER\_LED\_3 AN6 // Motor or GPS LED

#define COPTER\_LED\_4 AN7 // Motor LED

#define COPTER\_LED\_5 AN8 // Motor LED

#define COPTER\_LED\_6 AN9 // Motor LED

#define COPTER\_LED\_7 AN10 // Motor LED

#define COPTER\_LED\_8 AN11 // Motor LED

#elif CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM1

#define COPTER\_LED\_1 AN8 // Motor or Aux LED

#define COPTER\_LED\_2 AN9 // Motor LED

#define COPTER\_LED\_3 AN10 // Motor or GPS LED

#define COPTER\_LED\_4 AN11 // Motor LED

#define COPTER\_LED\_5 AN12 // Motor LED

#define COPTER\_LED\_6 AN13 // Motor LED

#define COPTER\_LED\_7 AN14 // Motor LED

#define COPTER\_LED\_8 AN15 // Motor LED

#endif

//////////////////////////////////////////////////////////////////////////////

// Pushbutton & Relay

//

#ifndef CONFIG\_PUSHBUTTON

# define CONFIG\_PUSHBUTTON ENABLED

#endif

#ifndef CONFIG\_RELAY

# define CONFIG\_RELAY ENABLED

#endif

//////////////////////////////////////////////////////////////////////////////

// Barometer

//

#ifndef CONFIG\_BARO

# define CONFIG\_BARO AP\_BARO\_BMP085

#endif

//////////////////////////////////////////////////////////////////////////////

// Sonar

//

#ifndef CONFIG\_SONAR\_SOURCE

# define CONFIG\_SONAR\_SOURCE SONAR\_SOURCE\_ADC

#endif

#if CONFIG\_SONAR\_SOURCE == SONAR\_SOURCE\_ADC && CONFIG\_ADC == DISABLED

# warning Cannot use ADC for CONFIG\_SONAR\_SOURCE, becaude CONFIG\_ADC is DISABLED

# warning Defaulting CONFIG\_SONAR\_SOURCE to ANALOG\_PIN

# undef CONFIG\_SONAR\_SOURCE

# define CONFIG\_SONAR\_SOURCE SONAR\_SOURCE\_ANALOG\_PIN

#endif

#if CONFIG\_SONAR\_SOURCE == SONAR\_SOURCE\_ADC

# ifndef CONFIG\_SONAR\_SOURCE\_ADC\_CHANNEL

# define CONFIG\_SONAR\_SOURCE\_ADC\_CHANNEL 7

# endif

#elif CONFIG\_SONAR\_SOURCE == SONAR\_SOURCE\_ANALOG\_PIN

# ifndef CONFIG\_SONAR\_SOURCE\_ANALOG\_PIN

# define CONFIG\_SONAR\_SOURCE\_ANALOG\_PIN A0

# endif

#else

# warning Invalid value for CONFIG\_SONAR\_SOURCE, disabling sonar

# undef SONAR\_ENABLED

# define SONAR\_ENABLED DISABLED

#endif

#ifndef CONFIG\_SONAR

# define CONFIG\_SONAR ENABLED

#endif

#ifndef SONAR\_ALT\_HEALTH\_MAX

# define SONAR\_ALT\_HEALTH\_MAX 3 // number of good reads that indicates a healthy sonar

#endif

#ifndef THR\_SURFACE\_TRACKING\_P

# define THR\_SURFACE\_TRACKING\_P 0.2 // gain for controlling how quickly sonar range adjusts target altitude (lower means slower reaction)

#endif

#ifndef THR\_SURFACE\_TRACKING\_VELZ\_MAX

# define THR\_SURFACE\_TRACKING\_VELZ\_MAX 30 // max speed number of good reads that indicates a healthy sonar

#endif

//////////////////////////////////////////////////////////////////////////////

// Channel 7 default option

//

#ifndef CH7\_OPTION

# define CH7\_OPTION CH7\_SAVE\_WP

#endif

//////////////////////////////////////////////////////////////////////////////

// HIL\_MODE OPTIONAL

#ifndef HIL\_MODE

#define HIL\_MODE HIL\_MODE\_DISABLED

#endif

#if HIL\_MODE != HIL\_MODE\_DISABLED // we are in HIL mode

# undef GPS\_PROTOCOL

# define GPS\_PROTOCOL GPS\_PROTOCOL\_NONE

#undef CONFIG\_SONAR

#define CONFIG\_SONAR DISABLED

#endif

//////////////////////////////////////////////////////////////////////////////

// GPS\_PROTOCOL

//

#ifndef GPS\_PROTOCOL

# define GPS\_PROTOCOL GPS\_PROTOCOL\_AUTO

#endif

#ifndef MAV\_SYSTEM\_ID

# define MAV\_SYSTEM\_ID 1

#endif

//////////////////////////////////////////////////////////////////////////////

// Serial port speeds.

//

#ifndef SERIAL0\_BAUD

# define SERIAL0\_BAUD 115200

#endif

#ifndef SERIAL3\_BAUD

# define SERIAL3\_BAUD 57600

#endif

//////////////////////////////////////////////////////////////////////////////

// Battery monitoring

//

#ifndef LOW\_VOLTAGE

# define LOW\_VOLTAGE 9.6

#endif

#ifndef VOLT\_DIV\_RATIO

# define VOLT\_DIV\_RATIO 3.56

#endif

#ifndef CURR\_AMP\_PER\_VOLT

# define CURR\_AMP\_PER\_VOLT 27.32

#endif

#ifndef CURR\_AMPS\_OFFSET

# define CURR\_AMPS\_OFFSET 0.0

#endif

#ifndef HIGH\_DISCHARGE

# define HIGH\_DISCHARGE 1760

#endif

// Battery failsafe

#ifndef FS\_BATTERY

# define FS\_BATTERY DISABLED

#endif

//////////////////////////////////////////////////////////////////////////////

// INPUT\_VOLTAGE

//

#ifndef INPUT\_VOLTAGE

# define INPUT\_VOLTAGE 5.0

#endif

//////////////////////////////////////////////////////////////////////////////

// MAGNETOMETER

#ifndef MAGNETOMETER

# define MAGNETOMETER ENABLED

#endif

#ifndef MAG\_ORIENTATION

# define MAG\_ORIENTATION AP\_COMPASS\_COMPONENTS\_DOWN\_PINS\_FORWARD

#endif

//////////////////////////////////////////////////////////////////////////////

// OPTICAL\_FLOW

#if defined( \_\_AVR\_ATmega2560\_\_ ) // determines if optical flow code is included

#define OPTFLOW ENABLED

#endif

#ifndef OPTFLOW // sets global enabled/disabled flag for optflow (as seen in CLI)

# define OPTFLOW DISABLED

#endif

#ifndef OPTFLOW\_ORIENTATION

# define OPTFLOW\_ORIENTATION AP\_OPTICALFLOW\_ADNS3080\_PINS\_FORWARD

#endif

#ifndef OPTFLOW\_RESOLUTION

# define OPTFLOW\_RESOLUTION ADNS3080\_RESOLUTION\_1600

#endif

#ifndef OPTFLOW\_FOV

# define OPTFLOW\_FOV AP\_OPTICALFLOW\_ADNS3080\_08\_FOV

#endif

// optical flow based loiter PI values

#ifndef OPTFLOW\_ROLL\_P

#define OPTFLOW\_ROLL\_P 2.5

#endif

#ifndef OPTFLOW\_ROLL\_I

#define OPTFLOW\_ROLL\_I 0.5

#endif

#ifndef OPTFLOW\_ROLL\_D

#define OPTFLOW\_ROLL\_D 0.12

#endif

#ifndef OPTFLOW\_PITCH\_P

#define OPTFLOW\_PITCH\_P 2.5

#endif

#ifndef OPTFLOW\_PITCH\_I

#define OPTFLOW\_PITCH\_I 0.5

#endif

#ifndef OPTFLOW\_PITCH\_D

#define OPTFLOW\_PITCH\_D 0.12

#endif

#ifndef OPTFLOW\_IMAX

#define OPTFLOW\_IMAX 1

#endif

//////////////////////////////////////////////////////////////////////////////

// RADIO CONFIGURATION

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

// FLIGHT\_MODE

//

#if !defined(FLIGHT\_MODE\_1)

# define FLIGHT\_MODE\_1 STABILIZE

#endif

#if !defined(FLIGHT\_MODE\_2)

# define FLIGHT\_MODE\_2 STABILIZE

#endif

#if !defined(FLIGHT\_MODE\_3)

# define FLIGHT\_MODE\_3 STABILIZE

#endif

#if !defined(FLIGHT\_MODE\_4)

# define FLIGHT\_MODE\_4 STABILIZE

#endif

#if !defined(FLIGHT\_MODE\_5)

# define FLIGHT\_MODE\_5 STABILIZE

#endif

#if !defined(FLIGHT\_MODE\_6)

# define FLIGHT\_MODE\_6 STABILIZE

#endif

//////////////////////////////////////////////////////////////////////////////

// Throttle Failsafe

//

// possible values for FS\_THR parameter

#define FS\_THR\_DISABLED 0

#define FS\_THR\_ENABLED\_ALWAYS\_RTL 1

#define FS\_THR\_ENABLED\_CONTINUE\_MISSION 2

#ifndef FS\_THR\_VALUE\_DEFAULT

# define FS\_THR\_VALUE\_DEFAULT 975

#endif

#ifndef MINIMUM\_THROTTLE

# define MINIMUM\_THROTTLE 130

#endif

#ifndef MAXIMUM\_THROTTLE

# define MAXIMUM\_THROTTLE 1000

#endif

#ifndef LAND\_SPEED

# define LAND\_SPEED 50 // the descent speed for the final stage of landing in cm/s

#endif

#ifndef LAND\_START\_ALT

# define LAND\_START\_ALT 1000 // altitude in cm where land controller switches to slow rate of descent

#endif

#ifndef LAND\_DETECTOR\_TRIGGER

# define LAND\_DETECTOR\_TRIGGER 50 // number of 50hz iterations with near zero climb rate and low throttle that triggers landing complete.

#endif

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

// STARTUP BEHAVIOUR

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

// GROUND\_START\_DELAY

//

#ifndef GROUND\_START\_DELAY

# define GROUND\_START\_DELAY 3

#endif

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

// FLIGHT AND NAVIGATION CONTROL

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

// Y6 Support

#ifndef TOP\_BOTTOM\_RATIO

# define TOP\_BOTTOM\_RATIO 1.00

#endif

//////////////////////////////////////////////////////////////////////////////

// CAMERA TRIGGER AND CONTROL

//

#ifndef CAMERA

# if defined( \_\_AVR\_ATmega1280\_\_ )

# define CAMERA DISABLED

# else

# define CAMERA ENABLED

# endif

#endif

//////////////////////////////////////////////////////////////////////////////

// MOUNT (ANTENNA OR CAMERA)

//

#ifndef MOUNT

# if defined( \_\_AVR\_ATmega1280\_\_ )

# define MOUNT DISABLED

# else

# define MOUNT ENABLED

# endif

#endif

#ifndef MOUNT2

# define MOUNT2 DISABLED

#endif

#if defined( \_\_AVR\_ATmega1280\_\_ ) && (MOUNT == ENABLED || MOUNT2 == ENABLED)

# warning "You choose to enable MOUNT on a small ATmega1280, CLI, CAMERA and AP\_LIMITS will be disabled to free some space for it"

// The small ATmega1280 chip does not have enough memory for mount support

// so disable CLI, this will allow mount support and other improvements to fit.

// This should almost have no side effects, because the APM planner can now do a complete board setup.

# define CLI\_ENABLED DISABLED

// The small ATmega1280 chip does not have enough memory for mount support

// so disable AUTO GPS support, this will allow mount support and other improvements to fit.

// This should almost have no side effects, because the most users use MTK anyways.

// If the user defined a GPS protocol, than we will NOT overwrite it

# if GPS\_PROTOCOL == GPS\_PROTOCOL\_AUTO

# undef GPS\_PROTOCOL

# define GPS\_PROTOCOL GPS\_PROTOCOL\_MTK

# endif

// To save some more space

# undef CAMERA

# define CAMERA DISABLED

# define AP\_LIMITS DISABLED

#endif

//////////////////////////////////////////////////////////////////////////////

// Attitude Control

//

// definitions for earth frame and body frame

// used to specify frame to rate controllers

#define EARTH\_FRAME 0

#define BODY\_FRAME 1

// Stabilize Mode

#ifndef STABILIZE\_THROTTLE

# define STABILIZE\_THROTTLE THROTTLE\_MANUAL\_TILT\_COMPENSATED

#endif

// Alt Hold Mode

#ifndef ALT\_HOLD\_YAW

# define ALT\_HOLD\_YAW YAW\_HOLD

#endif

#ifndef ALT\_HOLD\_RP

# define ALT\_HOLD\_RP ROLL\_PITCH\_STABLE

#endif

#ifndef ALT\_HOLD\_THR

# define ALT\_HOLD\_THR THROTTLE\_HOLD

#endif

// AUTO Mode

#ifndef AUTO\_YAW

# define AUTO\_YAW YAW\_LOOK\_AT\_NEXT\_WP

#endif

#ifndef AUTO\_RP

# define AUTO\_RP ROLL\_PITCH\_AUTO

#endif

#ifndef AUTO\_THR

# define AUTO\_THR THROTTLE\_AUTO

#endif

// Guided Mode

#ifndef GUIDED\_YAW

# define GUIDED\_YAW YAW\_LOOK\_AT\_NEXT\_WP

#endif

#ifndef GUIDED\_RP

# define GUIDED\_RP ROLL\_PITCH\_AUTO

#endif

#ifndef GUIDED\_THR

# define GUIDED\_THR THROTTLE\_AUTO

#endif

// CIRCLE Mode

#ifndef CIRCLE\_YAW

# define CIRCLE\_YAW YAW\_LOOK\_AT\_NEXT\_WP

#endif

#ifndef CIRCLE\_RP

# define CIRCLE\_RP ROLL\_PITCH\_AUTO

#endif

#ifndef CIRCLE\_THR

# define CIRCLE\_THR THROTTLE\_HOLD

#endif

// LOITER Mode

#ifndef LOITER\_YAW

# define LOITER\_YAW YAW\_HOLD

#endif

#ifndef LOITER\_RP

# define LOITER\_RP ROLL\_PITCH\_AUTO

#endif

#ifndef LOITER\_THR

# define LOITER\_THR THROTTLE\_HOLD

#endif

// RTL Mode

#ifndef RTL\_YAW

# define RTL\_YAW YAW\_LOOK\_AT\_NEXT\_WP

#endif

#ifndef RTL\_RP

# define RTL\_RP ROLL\_PITCH\_AUTO

#endif

#ifndef RTL\_THR

# define RTL\_THR THROTTLE\_AUTO

#endif

#ifndef SUPER\_SIMPLE

# define SUPER\_SIMPLE DISABLED

#endif

#ifndef SUPER\_SIMPLE\_RADIUS

# define SUPER\_SIMPLE\_RADIUS 1000

#endif

// RTL Mode

#ifndef RTL\_ALT\_FINAL

# define RTL\_ALT\_FINAL 200 // the altitude the vehicle will move to as the final stage of Returning to Launch. Set to zero to land.

#endif

#ifndef RTL\_ALT

# define RTL\_ALT 1500 // default alt to return to home in cm, 0 = Maintain current altitude

#endif

#ifndef RTL\_ALT\_MAX

# define RTL\_ALT\_MAX 8000 // Max height to return to home in cm (i.e 80m)

#endif

#ifndef RTL\_LOITER\_TIME

# define RTL\_LOITER\_TIME 5000 // Time (in milliseconds) to loiter above home before begining final descent

#endif

// Optical Flow LOITER Mode

#ifndef OF\_LOITER\_YAW

# define OF\_LOITER\_YAW YAW\_HOLD

#endif

#ifndef OF\_LOITER\_RP

# define OF\_LOITER\_RP ROLL\_PITCH\_STABLE\_OF

#endif

#ifndef OF\_LOITER\_THR

# define OF\_LOITER\_THR THROTTLE\_HOLD

#endif

//////////////////////////////////////////////////////////////////////////////

// Attitude Control

//

// Extra motor values that are changed from time to time by jani @ jDrones as software

// and charachteristics changes.

#ifdef MOTORS\_JD880

# define STABILIZE\_ROLL\_P 3.7

# define STABILIZE\_ROLL\_I 0.0

# define STABILIZE\_ROLL\_IMAX 8.0 // degrees

# define STABILIZE\_PITCH\_P 3.7

# define STABILIZE\_PITCH\_I 0.0

# define STABILIZE\_PITCH\_IMAX 8.0 // degrees

#endif

#ifdef MOTORS\_JD850

# define STABILIZE\_ROLL\_P 4.2

# define STABILIZE\_ROLL\_I 0.0

# define STABILIZE\_ROLL\_IMAX 8.0 // degrees

# define STABILIZE\_PITCH\_P 4.2

# define STABILIZE\_PITCH\_I 0.0

# define STABILIZE\_PITCH\_IMAX 8.0 // degrees

#endif

#ifndef ACRO\_P

# define ACRO\_P 4.5

#endif

#ifndef AXIS\_LOCK\_ENABLED

# define AXIS\_LOCK\_ENABLED ENABLED

#endif

// Good for smaller payload motors.

#ifndef STABILIZE\_ROLL\_P

# define STABILIZE\_ROLL\_P 4.5

#endif

#ifndef STABILIZE\_ROLL\_I

# define STABILIZE\_ROLL\_I 0.0

#endif

#ifndef STABILIZE\_ROLL\_IMAX

# define STABILIZE\_ROLL\_IMAX 8.0 // degrees

#endif

#ifndef STABILIZE\_PITCH\_P

# define STABILIZE\_PITCH\_P 4.5

#endif

#ifndef STABILIZE\_PITCH\_I

# define STABILIZE\_PITCH\_I 0.0

#endif

#ifndef STABILIZE\_PITCH\_IMAX

# define STABILIZE\_PITCH\_IMAX 8.0 // degrees

#endif

#ifndef STABILIZE\_YAW\_P

# define STABILIZE\_YAW\_P 4.5 // increase for more aggressive Yaw Hold, decrease if it's bouncy

#endif

#ifndef STABILIZE\_YAW\_I

# define STABILIZE\_YAW\_I 0.0

#endif

#ifndef STABILIZE\_YAW\_IMAX

# define STABILIZE\_YAW\_IMAX 8.0 // degrees \* 100

#endif

#ifndef YAW\_LOOK\_AHEAD\_MIN\_SPEED

# define YAW\_LOOK\_AHEAD\_MIN\_SPEED 1000 // minimum ground speed in cm/s required before copter is aimed at ground course

#endif

//////////////////////////////////////////////////////////////////////////////

// Stabilize Rate Control

//

#ifndef MAX\_INPUT\_ROLL\_ANGLE

# define MAX\_INPUT\_ROLL\_ANGLE 4500

#endif

#ifndef MAX\_INPUT\_PITCH\_ANGLE

# define MAX\_INPUT\_PITCH\_ANGLE 4500

#endif

#ifndef RATE\_ROLL\_P

# define RATE\_ROLL\_P 0.150

#endif

#ifndef RATE\_ROLL\_I

# define RATE\_ROLL\_I 0.100

#endif

#ifndef RATE\_ROLL\_D

# define RATE\_ROLL\_D 0.004

#endif

#ifndef RATE\_ROLL\_IMAX

# define RATE\_ROLL\_IMAX 5.0 // degrees

#endif

#ifndef RATE\_PITCH\_P

# define RATE\_PITCH\_P 0.150

#endif

#ifndef RATE\_PITCH\_I

# define RATE\_PITCH\_I 0.100

#endif

#ifndef RATE\_PITCH\_D

# define RATE\_PITCH\_D 0.004

#endif

#ifndef RATE\_PITCH\_IMAX

# define RATE\_PITCH\_IMAX 5.0 // degrees

#endif

#ifndef RATE\_YAW\_P

# define RATE\_YAW\_P 0.25

#endif

#ifndef RATE\_YAW\_I

# define RATE\_YAW\_I 0.015

#endif

#ifndef RATE\_YAW\_D

# define RATE\_YAW\_D 0.000

#endif

#ifndef RATE\_YAW\_IMAX

# define RATE\_YAW\_IMAX 8.0 // degrees

#endif

//////////////////////////////////////////////////////////////////////////////

// Rate controlled stabilized variables

//

#ifndef MAX\_ROLL\_OVERSHOOT

#define MAX\_ROLL\_OVERSHOOT 3000

#endif

#ifndef MAX\_PITCH\_OVERSHOOT

#define MAX\_PITCH\_OVERSHOOT 3000

#endif

#ifndef MAX\_YAW\_OVERSHOOT

#define MAX\_YAW\_OVERSHOOT 1000

#endif

#ifndef ACRO\_BALANCE\_ROLL

#define ACRO\_BALANCE\_ROLL 200

#endif

#ifndef ACRO\_BALANCE\_PITCH

#define ACRO\_BALANCE\_PITCH 200

#endif

#ifndef ACRO\_TRAINER\_ENABLED

#define ACRO\_TRAINER\_ENABLED ENABLED

#endif

//////////////////////////////////////////////////////////////////////////////

// Loiter control gains

//

#ifndef LOITER\_P

# define LOITER\_P .20

#endif

#ifndef LOITER\_I

# define LOITER\_I 0.0

#endif

#ifndef LOITER\_IMAX

# define LOITER\_IMAX 30 // degrees

#endif

// Loiter repositioning configuration (experimental)

#ifndef LOITER\_REPOSITIONING

# define LOITER\_REPOSITIONING DISABLED

#endif

#ifndef LOITER\_REPOSITION\_RATE

# define LOITER\_REPOSITION\_RATE 500.0 // cm/s

#endif

//////////////////////////////////////////////////////////////////////////////

// Loiter Navigation control gains

//

#ifndef LOITER\_RATE\_P

# define LOITER\_RATE\_P 5.0 //

#endif

#ifndef LOITER\_RATE\_I

# define LOITER\_RATE\_I 0.04 // Wind control

#endif

#ifndef LOITER\_RATE\_D

# define LOITER\_RATE\_D 0.40 // try 2 or 3 for LOITER\_RATE 1

#endif

#ifndef LOITER\_RATE\_IMAX

# define LOITER\_RATE\_IMAX 30 // degrees

#endif

//////////////////////////////////////////////////////////////////////////////

// WP Navigation control gains

//

#ifndef NAV\_P

# define NAV\_P 2.4 //

#endif

#ifndef NAV\_I

# define NAV\_I 0.17 // Wind control

#endif

#ifndef NAV\_D

# define NAV\_D 0.00 // .95

#endif

#ifndef NAV\_IMAX

# define NAV\_IMAX 18 // degrees

#endif

#ifndef AUTO\_SLEW\_RATE

# define AUTO\_SLEW\_RATE 30 // degrees/sec

#endif

#ifndef AUTO\_YAW\_SLEW\_RATE

# define AUTO\_YAW\_SLEW\_RATE 60 // degrees/sec

#endif

#ifndef WAYPOINT\_SPEED\_MAX

# define WAYPOINT\_SPEED\_MAX 500 // 6m/s error = 13mph

#endif

#ifndef WAYPOINT\_SPEED\_MIN

# define WAYPOINT\_SPEED\_MIN 150 // 1m/s

#endif

#ifndef TILT\_COMPENSATION

# define TILT\_COMPENSATION 54

#endif

//////////////////////////////////////////////////////////////////////////////

// Throttle control gains

//

#ifndef THROTTLE\_CRUISE

# define THROTTLE\_CRUISE 450 //

#endif

#ifndef THR\_MID

# define THR\_MID 500 // Throttle output (0 ~ 1000) when throttle stick is in mid position

#endif

#ifndef ALT\_HOLD\_P

# define ALT\_HOLD\_P 2.0

#endif

#ifndef ALT\_HOLD\_I

# define ALT\_HOLD\_I 0.0

#endif

#ifndef ALT\_HOLD\_IMAX

# define ALT\_HOLD\_IMAX 300

#endif

// RATE control

#ifndef THROTTLE\_P

# define THROTTLE\_P 6.0

#endif

#ifndef THROTTLE\_I

# define THROTTLE\_I 0.0

#endif

#ifndef THROTTLE\_D

# define THROTTLE\_D 0.2

#endif

#ifndef THROTTLE\_IMAX

# define THROTTLE\_IMAX 300

#endif

// default minimum and maximum vertical velocity the autopilot may request

#ifndef AUTO\_VELZ\_MIN

# define AUTO\_VELZ\_MIN -125

#endif

#ifndef AUTO\_VELZ\_MAX

# define AUTO\_VELZ\_MAX 125

#endif

// default maximum vertical velocity the pilot may request

#ifndef PILOT\_VELZ\_MAX

# define PILOT\_VELZ\_MAX 250 // maximum vertical velocity in cm/s

#endif

#define ACCELERATION\_MAX\_Z 750 // maximum veritcal acceleration in cm/s/s

// Throttle Accel control

#ifndef THROTTLE\_ACCEL\_P

# define THROTTLE\_ACCEL\_P 0.75

#endif

#ifndef THROTTLE\_ACCEL\_I

# define THROTTLE\_ACCEL\_I 1.50

#endif

#ifndef THROTTLE\_ACCEL\_D

# define THROTTLE\_ACCEL\_D 0.0

#endif

#ifndef THROTTLE\_ACCEL\_IMAX

# define THROTTLE\_ACCEL\_IMAX 500

#endif

//////////////////////////////////////////////////////////////////////////////

// Crosstrack compensation

//

#ifndef CROSSTRACK\_GAIN

# define CROSSTRACK\_GAIN .2

#endif

#ifndef CROSSTRACK\_MIN\_DISTANCE

# define CROSSTRACK\_MIN\_DISTANCE 15

#endif

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

// DEBUGGING

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

// DEBUG\_LEVEL

//

#ifndef DEBUG\_LEVEL

# define DEBUG\_LEVEL SEVERITY\_LOW

#endif

//////////////////////////////////////////////////////////////////////////////

// Dataflash logging control

//

// Logging must be disabled for 1280 build.

#if defined( \_\_AVR\_ATmega1280\_\_ )

# if LOGGING\_ENABLED == ENABLED

// If logging was enabled in APM\_Config or command line, warn the user.

# warning "Logging is not supported on ATmega1280"

# undef LOGGING\_ENABLED

# endif

# ifndef LOGGING\_ENABLED

# define LOGGING\_ENABLED DISABLED

# endif

#elif !defined(LOGGING\_ENABLED)

// Logging is enabled by default for all other builds.

# define LOGGING\_ENABLED ENABLED

#endif

#ifndef LOG\_ATTITUDE\_FAST

# define LOG\_ATTITUDE\_FAST DISABLED

#endif

#ifndef LOG\_ATTITUDE\_MED

# define LOG\_ATTITUDE\_MED ENABLED

#endif

#ifndef LOG\_GPS

# define LOG\_GPS ENABLED

#endif

#ifndef LOG\_PM

# define LOG\_PM ENABLED

#endif

#ifndef LOG\_CTUN

# define LOG\_CTUN ENABLED

#endif

#ifndef LOG\_NTUN

# define LOG\_NTUN ENABLED

#endif

#ifndef LOG\_MODE

# define LOG\_MODE ENABLED

#endif

#ifndef LOG\_RAW

# define LOG\_RAW DISABLED

#endif

#ifndef LOG\_CMD

# define LOG\_CMD ENABLED

#endif

// current

#ifndef LOG\_CUR

# define LOG\_CUR DISABLED

#endif

// quad motor PWMs

#ifndef LOG\_MOTORS

# define LOG\_MOTORS DISABLED

#endif

// optical flow

#ifndef LOG\_OPTFLOW

# define LOG\_OPTFLOW DISABLED

#endif

#ifndef LOG\_PID

# define LOG\_PID DISABLED

#endif

#ifndef LOG\_ITERM

# define LOG\_ITERM DISABLED

#endif

#ifndef LOG\_INAV

# define LOG\_INAV DISABLED

#endif

#ifndef LOG\_CAMERA

# define LOG\_CAMERA DISABLED

#endif

// calculate the default log\_bitmask

#define LOGBIT(\_s) (LOG\_ ## \_s ? MASK\_LOG\_ ## \_s : 0)

#define DEFAULT\_LOG\_BITMASK \

LOGBIT(ATTITUDE\_FAST) | \

LOGBIT(ATTITUDE\_MED) | \

LOGBIT(GPS) | \

LOGBIT(PM) | \

LOGBIT(CTUN) | \

LOGBIT(NTUN) | \

LOGBIT(MODE) | \

LOGBIT(RAW) | \

LOGBIT(CMD) | \

LOGBIT(CUR) | \

LOGBIT(MOTORS) | \

LOGBIT(OPTFLOW) | \

LOGBIT(PID) | \

LOGBIT(ITERM) | \

LOGBIT(INAV)

// if we are using fast, Disable Medium

//#if LOG\_ATTITUDE\_FAST == ENABLED

// #undef LOG\_ATTITUDE\_MED

// #define LOG\_ATTITUDE\_MED DISABLED

//#endif

//////////////////////////////////////////////////////////////////////////////

// Navigation defaults

//

#ifndef WP\_RADIUS\_DEFAULT

# define WP\_RADIUS\_DEFAULT 2

#endif

#ifndef CIRCLE\_RADIUS

# define CIRCLE\_RADIUS 10 // meters for circle mode

#endif

#ifndef USE\_CURRENT\_ALT

# define USE\_CURRENT\_ALT FALSE

#endif

//////////////////////////////////////////////////////////////////////////////

// AP\_Limits Defaults

//

// Enable/disable AP\_Limits

#ifndef AP\_LIMITS

#define AP\_LIMITS ENABLED

#endif

// Use PIN for displaying LIMITS status. 0 is disabled.

#ifndef LIMITS\_TRIGGERED\_PIN

#define LIMITS\_TRIGGERED\_PIN 0

#endif

// PWM of "on" state for LIM\_CHANNEL

#ifndef LIMITS\_ENABLE\_PWM

#define LIMITS\_ENABLE\_PWM 1800

#endif

#ifndef LIM\_ENABLED

#define LIM\_ENABLED 0

#endif

#ifndef LIM\_ALT\_ON

#define LIM\_ALT\_ON 0

#endif

#ifndef LIM\_FNC\_ON

#define LIM\_FNC\_ON 0

#endif

#ifndef LIM\_GPSLCK\_ON

#define LIM\_GPSLCK\_ON 0

#endif

//////////////////////////////////////////////////////////////////////////////

// Developer Items

//

// use this to completely disable the CLI

#ifndef CLI\_ENABLED

// Sorry the chip is just too small to let this fit

# if defined( \_\_AVR\_ATmega1280\_\_ )

# define CLI\_ENABLED DISABLED

# else

# define CLI\_ENABLED ENABLED

# endif

#endif

// use this to disable the CLI slider switch

#ifndef CLI\_SLIDER\_ENABLED

# define CLI\_SLIDER\_ENABLED DISABLED

#endif

// experimental mpu6000 DMP code

#ifndef DMP\_ENABLED

# define DMP\_ENABLED DISABLED

#endif

// experimental mpu6000 DMP code

#ifndef SECONDARY\_DMP\_ENABLED

# define SECONDARY\_DMP\_ENABLED DISABLED

#endif

// Inertia based contollers.

#ifndef INERTIAL\_NAV\_XY

# define INERTIAL\_NAV\_XY DISABLED

#endif

#ifndef INERTIAL\_NAV\_Z

# define INERTIAL\_NAV\_Z ENABLED

#endif

#endif // \_\_ARDUCOPTER\_CONFIG\_H\_\_

## config\_channels.h

#ifndef \_\_ARDUCOPTER\_CONFIG\_MOTORS\_H\_\_

#define \_\_ARDUCOPTER\_CONFIG\_MOTORS\_H\_\_

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//

// WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING

//

// DO NOT EDIT this file to adjust your configuration. Create your own

// APM\_Config.h and use APM\_Config.h.example as a reference.

//

// WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING

///

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//

#include "config.h" // Parent Config File

#include "APM\_Config.h" // User Overrides

#endif // \_\_ARDUCOPTER\_CONFIG\_MOTORS\_H\_\_

## control\_modes.pde

/// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

#define CONTROL\_SWITCH\_COUNTER 10 // 10 iterations at 100hz (i.e. 1/10th of a second) at a new switch position will cause flight mode change

static void read\_control\_switch()

{

static uint8\_t switch\_counter = 0;

byte switchPosition = readSwitch();

if (oldSwitchPosition != switchPosition) {

switch\_counter++;

if(switch\_counter >= CONTROL\_SWITCH\_COUNTER) {

oldSwitchPosition = switchPosition;

switch\_counter = 0;

// ignore flight mode changes if in failsafe

if( !ap.failsafe ) {

set\_mode(flight\_modes[switchPosition]);

if(g.ch7\_option != CH7\_SIMPLE\_MODE) {

// set Simple mode using stored paramters from Mission planner

// rather than by the control switch

set\_simple\_mode(g.simple\_modes & (1 << switchPosition));

}

}

}

}else{

// reset switch\_counter if there's been no change

// we don't want 10 intermittant blips causing a flight mode change

switch\_counter = 0;

}

}

static byte readSwitch(void){

int16\_t pulsewidth = g.rc\_5.radio\_in; // default for Arducopter

if (pulsewidth > 1230 && pulsewidth <= 1360) return 1;

if (pulsewidth > 1360 && pulsewidth <= 1490) return 2;

if (pulsewidth > 1490 && pulsewidth <= 1620) return 3;

if (pulsewidth > 1620 && pulsewidth <= 1749) return 4;

if (pulsewidth >= 1750) return 5;

return 0;

}

static void reset\_control\_switch()

{

oldSwitchPosition = -1;

read\_control\_switch();

}

// read at 10 hz

// set this to your trainer switch

static void read\_trim\_switch()

{

// return immediately if the CH7 switch has not changed position

if (ap\_system.CH7\_flag == (g.rc\_7.radio\_in >= CH7\_PWM\_TRIGGER)) {

return;

}

// set the ch7 flag

ap\_system.CH7\_flag = (g.rc\_7.radio\_in >= CH7\_PWM\_TRIGGER);

// multi-mode

int8\_t option;

if(g.ch7\_option == CH7\_MULTI\_MODE) {

if (g.rc\_6.radio\_in < CH6\_PWM\_TRIGGER\_LOW) {

option = CH7\_FLIP;

}else if (g.rc\_6.radio\_in > CH6\_PWM\_TRIGGER\_HIGH) {

option = CH7\_SAVE\_WP;

}else{

option = CH7\_RTL;

}

}else{

option = g.ch7\_option;

}

switch(option) {

case CH7\_FLIP:

// flip if switch is on, positive throttle and we're actually flying

if(ap\_system.CH7\_flag && g.rc\_3.control\_in >= 0 && ap.takeoff\_complete) {

init\_flip();

}

break;

case CH7\_SIMPLE\_MODE:

set\_simple\_mode(ap\_system.CH7\_flag);

break;

case CH7\_RTL:

if (ap\_system.CH7\_flag) {

// engage RTL

set\_mode(RTL);

}else{

// disengage RTL to previous flight mode if we are currently in RTL or loiter

if (control\_mode == RTL || control\_mode == LOITER) {

reset\_control\_switch();

}

}

break;

case CH7\_SAVE\_TRIM:

if(ap\_system.CH7\_flag && control\_mode <= ACRO && g.rc\_3.control\_in == 0) {

save\_trim();

}

break;

case CH7\_SAVE\_WP:

// save when CH7 switch is switched off

if (ap\_system.CH7\_flag == false) {

// if in auto mode, reset the mission

if(control\_mode == AUTO) {

CH7\_wp\_index = 0;

g.command\_total.set\_and\_save(1);

set\_mode(RTL);

return;

}

if(CH7\_wp\_index == 0) {

// this is our first WP, let's save WP 1 as a takeoff

// increment index to WP index of 1 (home is stored at 0)

CH7\_wp\_index = 1;

Location temp = home;

// set our location ID to 16, MAV\_CMD\_NAV\_WAYPOINT

temp.id = MAV\_CMD\_NAV\_TAKEOFF;

temp.alt = current\_loc.alt;

// save command:

// we use the current altitude to be the target for takeoff.

// only altitude will matter to the AP mission script for takeoff.

// If we are above the altitude, we will skip the command.

set\_cmd\_with\_index(temp, CH7\_wp\_index);

}

// increment index

CH7\_wp\_index++;

// set the next\_WP (home is stored at 0)

// max out at 100 since I think we need to stay under the EEPROM limit

CH7\_wp\_index = constrain(CH7\_wp\_index, 1, 100);

if(g.rc\_3.control\_in > 0) {

// set our location ID to 16, MAV\_CMD\_NAV\_WAYPOINT

current\_loc.id = MAV\_CMD\_NAV\_WAYPOINT;

}else{

// set our location ID to 21, MAV\_CMD\_NAV\_LAND

current\_loc.id = MAV\_CMD\_NAV\_LAND;

}

// save command

set\_cmd\_with\_index(current\_loc, CH7\_wp\_index);

// Cause the CopterLEDs to blink twice to indicate saved waypoint

copter\_leds\_nav\_blink = 10;

}

break;

#if CAMERA == ENABLED

case CH7\_CAMERA\_TRIGGER:

if(ap\_system.CH7\_flag) {

do\_take\_picture();

}

break;

#endif

case CH7\_SONAR:

// enable or disable the sonar

g.sonar\_enabled = ap\_system.CH7\_flag;

break;

}

}

// save\_trim - adds roll and pitch trims from the radio to ahrs

static void save\_trim()

{

// save roll and pitch trim

float roll\_trim = ToRad((float)g.rc\_1.control\_in/100.0);

float pitch\_trim = ToRad((float)g.rc\_2.control\_in/100.0);

ahrs.add\_trim(roll\_trim, pitch\_trim);

}

// auto\_trim - slightly adjusts the ahrs.roll\_trim and ahrs.pitch\_trim towards the current stick positions

// meant to be called continuously while the pilot attempts to keep the copter level

static void auto\_trim()

{

if(auto\_trim\_counter > 0) {

auto\_trim\_counter--;

// flash the leds

led\_mode = SAVE\_TRIM\_LEDS;

// calculate roll trim adjustment

float roll\_trim\_adjustment = ToRad((float)g.rc\_1.control\_in / 4000.0);

// calculate pitch trim adjustment

float pitch\_trim\_adjustment = ToRad((float)g.rc\_2.control\_in / 4000.0);

// make sure accelerometer values impact attitude quickly

ahrs.set\_fast\_gains(true);

// add trim to ahrs object

// save to eeprom on last iteration

ahrs.add\_trim(roll\_trim\_adjustment, pitch\_trim\_adjustment, (auto\_trim\_counter == 0));

// on last iteration restore leds and accel gains to normal

if(auto\_trim\_counter == 0) {

ahrs.set\_fast\_gains(false);

led\_mode = NORMAL\_LEDS;

}

}

}

## defines.h

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

#ifndef \_DEFINES\_H

#define \_DEFINES\_H

// Just so that it's completely clear...

#define ENABLED 1

#define DISABLED 0

// this avoids a very common config error

#define ENABLE ENABLED

#define DISABLE DISABLED

// Flight modes

// ------------

#define YAW\_HOLD 0 // heading hold at heading in nav\_yaw but allow input from pilot

#define YAW\_ACRO 1 // pilot controlled yaw using rate controller

#define YAW\_LOOK\_AT\_NEXT\_WP 2 // point towards next waypoint (no pilot input accepted)

#define YAW\_LOOK\_AT\_LOCATION 3 // point towards a location held in yaw\_look\_at\_WP (no pilot input accepted)

#define YAW\_LOOK\_AT\_HOME 4 // point towards home (no pilot input accepted)

#define YAW\_LOOK\_AT\_HEADING 5 // point towards a particular angle (not pilot input accepted)

#define YAW\_LOOK\_AHEAD 6 // WARNING! CODE IN DEVELOPMENT NOT PROVEN

#define YAW\_TOY 7 // THOR This is the Yaw mode

#define ROLL\_PITCH\_STABLE 0

#define ROLL\_PITCH\_ACRO 1

#define ROLL\_PITCH\_AUTO 2

#define ROLL\_PITCH\_STABLE\_OF 3

#define ROLL\_PITCH\_TOY 4 // THOR This is the Roll and Pitch mode

#define ROLL\_PITCH\_LOITER\_PR 5

#define THROTTLE\_MANUAL 0 // manual throttle mode - pilot input goes directly to motors

#define THROTTLE\_MANUAL\_TILT\_COMPENSATED 1 // mostly manual throttle but with some tilt compensation

#define THROTTLE\_ACCELERATION 2 // pilot inputs the desired acceleration

#define THROTTLE\_RATE 3 // pilot inputs the desired climb rate. Note: this uses the unstabilized rate controller

#define THROTTLE\_STABILIZED\_RATE 4 // pilot inputs the desired climb rate. Uses stabilized rate controller

#define THROTTLE\_DIRECT\_ALT 5 // pilot inputs a desired altitude from 0 ~ 10 meters

#define THROTTLE\_HOLD 6 // alt hold plus pilot input of climb rate

#define THROTTLE\_AUTO 7 // auto pilot altitude controller with target altitude held in next\_WP.alt

#define THROTTLE\_LAND 8 // landing throttle controller

// active altitude sensor

// ----------------------

#define SONAR 0

#define BARO 1

#define SONAR\_SOURCE\_ADC 1

#define SONAR\_SOURCE\_ANALOG\_PIN 2

// CH 7 control

#define CH7\_PWM\_TRIGGER 1800 // pwm value above which the channel 7 option will be invoked

#define CH6\_PWM\_TRIGGER\_HIGH 1800

#define CH6\_PWM\_TRIGGER\_LOW 1200

#define CH7\_DO\_NOTHING 0

#define CH7\_SET\_HOVER 1 // deprecated

#define CH7\_FLIP 2

#define CH7\_SIMPLE\_MODE 3

#define CH7\_RTL 4

#define CH7\_SAVE\_TRIM 5

#define CH7\_ADC\_FILTER 6 // deprecated

#define CH7\_SAVE\_WP 7

#define CH7\_MULTI\_MODE 8

#define CH7\_CAMERA\_TRIGGER 9

#define CH7\_SONAR 10 // allow enabling or disabling sonar in flight which helps avoid surface tracking when you are far above the ground

// Frame types

#define QUAD\_FRAME 0

#define TRI\_FRAME 1

#define HEXA\_FRAME 2

#define Y6\_FRAME 3

#define OCTA\_FRAME 4

#define HELI\_FRAME 5

#define OCTA\_QUAD\_FRAME 6

#define PLUS\_FRAME 0

#define X\_FRAME 1

#define V\_FRAME 2

// LED output

#define NORMAL\_LEDS 0

#define SAVE\_TRIM\_LEDS 1

// Internal defines, don't edit and expect things to work

// -------------------------------------------------------

#define TRUE 1

#define FALSE 0

#define ToRad(x) (x\*0.01745329252) // \*pi/180

#define ToDeg(x) (x\*57.2957795131) // \*180/pi

#define DEBUG 0

#define LOITER\_RANGE 60 // for calculating power outside of loiter radius

#define T6 1000000

#define T7 10000000

// GPS type codes - use the names, not the numbers

#define GPS\_PROTOCOL\_NONE -1

#define GPS\_PROTOCOL\_NMEA 0

#define GPS\_PROTOCOL\_SIRF 1

#define GPS\_PROTOCOL\_UBLOX 2

#define GPS\_PROTOCOL\_IMU 3

#define GPS\_PROTOCOL\_MTK 4

#define GPS\_PROTOCOL\_HIL 5

#define GPS\_PROTOCOL\_MTK19 6

#define GPS\_PROTOCOL\_AUTO 7

// HIL enumerations

#define HIL\_MODE\_DISABLED 0

#define HIL\_MODE\_ATTITUDE 1

#define HIL\_MODE\_SENSORS 2

// Altitude status definitions

#define REACHED\_ALT 0

#define DESCENDING 1

#define ASCENDING 2

// Auto Pilot modes

// ----------------

#define STABILIZE 0 // hold level position

#define ACRO 1 // rate control

#define ALT\_HOLD 2 // AUTO control

#define AUTO 3 // AUTO control

#define GUIDED 4 // AUTO control

#define LOITER 5 // Hold a single location

#define RTL 6 // AUTO control

#define CIRCLE 7 // AUTO control

#define POSITION 8 // AUTO control

#define LAND 9 // AUTO control

#define OF\_LOITER 10 // Hold a single location using optical flow

// sensor

#define TOY\_A 11 // THOR Enum for Toy mode

#define TOY\_M 12 // THOR Enum for Toy mode

#define NUM\_MODES 13

// CH\_6 Tuning

// -----------

#define CH6\_NONE 0 // no tuning performed

#define CH6\_STABILIZE\_KP 1 // stabilize roll/pitch angle controller's P term

#define CH6\_STABILIZE\_KI 2 // stabilize roll/pitch angle controller's I term

#define CH6\_STABILIZE\_KD 29 // stabilize roll/pitch angle controller's D term

#define CH6\_YAW\_KP 3 // stabilize yaw heading controller's P term

#define CH6\_YAW\_KI 24 // stabilize yaw heading controller's P term

#define CH6\_ACRO\_KP 25 // acro controller's P term. converts pilot input to a desired roll, pitch or yaw rate

#define CH6\_RATE\_KP 4 // body frame roll/pitch rate controller's P term

#define CH6\_RATE\_KI 5 // body frame roll/pitch rate controller's I term

#define CH6\_RATE\_KD 21 // body frame roll/pitch rate controller's D term

#define CH6\_YAW\_RATE\_KP 6 // body frame yaw rate controller's P term

#define CH6\_YAW\_RATE\_KD 26 // body frame yaw rate controller's D term

#define CH6\_THR\_HOLD\_KP 14 // altitude hold controller's P term (alt error to desired rate)

#define CH6\_THROTTLE\_KP 7 // throttle rate controller's P term (desired rate to acceleration or motor output)

#define CH6\_THROTTLE\_KI 33 // throttle rate controller's I term (desired rate to acceleration or motor output)

#define CH6\_THROTTLE\_KD 37 // throttle rate controller's D term (desired rate to acceleration or motor output)

#define CH6\_THR\_ACCEL\_KP 34 // accel based throttle controller's P term

#define CH6\_THR\_ACCEL\_KI 35 // accel based throttle controller's I term

#define CH6\_THR\_ACCEL\_KD 36 // accel based throttle controller's D term

#define CH6\_TOP\_BOTTOM\_RATIO 8 // upper/lower motor ratio (not used)

#define CH6\_RELAY 9 // switch relay on if ch6 high, off if low

#define CH6\_TRAVERSE\_SPEED 10 // maximum speed to next way point (0 to 10m/s)

#define CH6\_NAV\_KP 11 // navigation rate controller's P term (speed error to tilt angle)

#define CH6\_NAV\_KI 20 // navigation rate controller's I term (speed error to tilt angle)

#define CH6\_LOITER\_KP 12 // loiter distance controller's P term (position error to speed)

#define CH6\_LOITER\_KI 27 // loiter distance controller's I term (position error to speed)

#define CH6\_HELI\_EXTERNAL\_GYRO 13 // TradHeli specific external tail gyro gain

#define CH6\_OPTFLOW\_KP 17 // optical flow loiter controller's P term (position error to tilt angle)

#define CH6\_OPTFLOW\_KI 18 // optical flow loiter controller's I term (position error to tilt angle)

#define CH6\_OPTFLOW\_KD 19 // optical flow loiter controller's D term (position error to tilt angle)

#define CH6\_LOITER\_RATE\_KP 22 // loiter rate controller's P term (speed error to tilt angle)

#define CH6\_LOITER\_RATE\_KI 28 // loiter rate controller's I term (speed error to tilt angle)

#define CH6\_LOITER\_RATE\_KD 23 // loiter rate controller's D term (speed error to tilt angle)

#define CH6\_AHRS\_YAW\_KP 30 // ahrs's compass effect on yaw angle (0 = very low, 1 = very high)

#define CH6\_AHRS\_KP 31 // accelerometer effect on roll/pitch angle (0=low)

#define CH6\_INAV\_TC 32 // inertial navigation baro/accel and gps/accel time constant (1.5 = strong baro/gps correction on accel estimatehas very strong does not correct accel estimate, 7 = very weak correction)

// nav byte mask used with wp\_verify\_byte variable

// -----------------------------------------------

#define NAV\_LOCATION 1

#define NAV\_ALTITUDE 2

#define NAV\_DELAY 4

// Commands - Note that APM now uses a subset of the MAVLink protocol

// commands. See enum MAV\_CMD in the GCS\_Mavlink library

#define CMD\_BLANK 0 // there is no command stored in the mem location

// requested

#define NO\_COMMAND 0

// Navigation modes held in wp\_control variable

#define LOITER\_MODE 1

#define WP\_MODE 2

#define CIRCLE\_MODE 3

#define NO\_NAV\_MODE 4

// Yaw override behaviours - used for setting yaw\_override\_behaviour

#define YAW\_OVERRIDE\_BEHAVIOUR\_AT\_NEXT\_WAYPOINT 0 // auto pilot takes back yaw control at next waypoint

#define YAW\_OVERRIDE\_BEHAVIOUR\_AT\_MISSION\_RESTART 1 // auto pilot tkaes back control only when mission is restarted

// TOY mixing options

#define TOY\_LOOKUP\_TABLE 0

#define TOY\_LINEAR\_MIXER 1

#define TOY\_EXTERNAL\_MIXER 2

// Waypoint options

#define MASK\_OPTIONS\_RELATIVE\_ALT 1

#define WP\_OPTION\_ALT\_CHANGE 2

#define WP\_OPTION\_YAW 4

#define WP\_OPTION\_ALT\_REQUIRED 8

#define WP\_OPTION\_RELATIVE 16

//#define WP\_OPTION\_ 32

//#define WP\_OPTION\_ 64

#define WP\_OPTION\_NEXT\_CMD 128

// RTL state

#define RTL\_STATE\_INITIAL\_CLIMB 0

#define RTL\_STATE\_RETURNING\_HOME 1

#define RTL\_STATE\_LOITERING\_AT\_HOME 2

#define RTL\_STATE\_FINAL\_DESCENT 3

#define RTL\_STATE\_LAND 4

//repeating events

#define RELAY\_TOGGLE 5

// GCS Message ID's

/// NOTE: to ensure we never block on sending MAVLink messages

/// please keep each MSG\_ to a single MAVLink message. If need be

/// create new MSG\_ IDs for additional messages on the same

/// stream

enum ap\_message {

MSG\_HEARTBEAT,

MSG\_ATTITUDE,

MSG\_LOCATION,

MSG\_EXTENDED\_STATUS1,

MSG\_EXTENDED\_STATUS2,

MSG\_NAV\_CONTROLLER\_OUTPUT,

MSG\_CURRENT\_WAYPOINT,

MSG\_VFR\_HUD,

MSG\_RADIO\_OUT,

MSG\_RADIO\_IN,

MSG\_RAW\_IMU1,

MSG\_RAW\_IMU2,

MSG\_RAW\_IMU3,

MSG\_GPS\_RAW,

MSG\_SERVO\_OUT,

MSG\_NEXT\_WAYPOINT,

MSG\_NEXT\_PARAM,

MSG\_STATUSTEXT,

MSG\_LIMITS\_STATUS,

MSG\_AHRS,

MSG\_SIMSTATE,

MSG\_HWSTATUS,

MSG\_RETRY\_DEFERRED // this must be last

};

enum gcs\_severity {

SEVERITY\_LOW=1,

SEVERITY\_MEDIUM,

SEVERITY\_HIGH,

SEVERITY\_CRITICAL

};

// Logging parameters

#define TYPE\_AIRSTART\_MSG 0x00

#define TYPE\_GROUNDSTART\_MSG 0x01

#define LOG\_ATTITUDE\_MSG 0x01

#define LOG\_GPS\_MSG 0x02

#define LOG\_MODE\_MSG 0x03

#define LOG\_CONTROL\_TUNING\_MSG 0x04

#define LOG\_NAV\_TUNING\_MSG 0x05

#define LOG\_PERFORMANCE\_MSG 0x06

#define LOG\_RAW\_MSG 0x07

#define LOG\_CMD\_MSG 0x08

#define LOG\_CURRENT\_MSG 0x09

#define LOG\_STARTUP\_MSG 0x0A

#define LOG\_MOTORS\_MSG 0x0B

#define LOG\_OPTFLOW\_MSG 0x0C

#define LOG\_DATA\_MSG 0x0D

#define LOG\_PID\_MSG 0x0E

#define LOG\_ITERM\_MSG 0x0F

#define LOG\_DMP\_MSG 0x10

#define LOG\_INAV\_MSG 0x11

#define LOG\_CAMERA\_MSG 0x12

#define LOG\_ERROR\_MSG 0x13

#define LOG\_INDEX\_MSG 0xF0

#define MAX\_NUM\_LOGS 50

#define MASK\_LOG\_ATTITUDE\_FAST (1<<0)

#define MASK\_LOG\_ATTITUDE\_MED (1<<1)

#define MASK\_LOG\_GPS (1<<2)

#define MASK\_LOG\_PM (1<<3)

#define MASK\_LOG\_CTUN (1<<4)

#define MASK\_LOG\_NTUN (1<<5)

#define MASK\_LOG\_MODE (1<<6)

#define MASK\_LOG\_RAW (1<<7)

#define MASK\_LOG\_CMD (1<<8)

#define MASK\_LOG\_CUR (1<<9)

#define MASK\_LOG\_MOTORS (1<<10)

#define MASK\_LOG\_OPTFLOW (1<<11)

#define MASK\_LOG\_PID (1<<12)

#define MASK\_LOG\_ITERM (1<<13)

#define MASK\_LOG\_INAV (1<<14)

#define MASK\_LOG\_CAMERA (1<<15)

// DATA - event logging

#define DATA\_MAVLINK\_FLOAT 1

#define DATA\_MAVLINK\_INT32 2

#define DATA\_MAVLINK\_INT16 3

#define DATA\_MAVLINK\_INT8 4

#define DATA\_FAST\_LOOP 5

#define DATA\_MED\_LOOP 6

#define DATA\_AP\_STATE 7

#define DATA\_SIMPLE\_BEARING 8

#define DATA\_INIT\_SIMPLE\_BEARING 9

#define DATA\_ARMED 10

#define DATA\_DISARMED 11

#define DATA\_AUTO\_ARMED 15

#define DATA\_TAKEOFF 16

#define DATA\_DID\_REACH\_ALT 17

#define DATA\_LAND\_COMPLETE 18

#define DATA\_LOST\_GPS 19

#define DATA\_LOST\_COMPASS 20

#define DATA\_BEGIN\_FLIP 21

#define DATA\_END\_FLIP 22

#define DATA\_EXIT\_FLIP 23

#define DATA\_FLIP\_ABORTED 24

#define DATA\_SET\_HOME 25

#define DATA\_SET\_SIMPLE\_ON 26

#define DATA\_SET\_SIMPLE\_OFF 27

#define DATA\_REACHED\_ALT 28

#define DATA\_ASCENDING 29

#define DATA\_DESCENDING 30

#define DATA\_RTL\_REACHED\_ALT 31

// battery monitoring macros

#define BATTERY\_VOLTAGE(x) (x\*(g.input\_voltage/1024.0))\*g.volt\_div\_ratio

#define CURRENT\_AMPS(x) ((x\*(g.input\_voltage/1024.0))-CURR\_AMPS\_OFFSET)\*g.curr\_amp\_per\_volt

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* Expansion PIN's that people can use for various things. \*/

// AN0 - 7 are located at edge of IMU PCB "above" pressure sensor and

// Expansion port

// AN0 - 5 are also located next to voltage dividers and sliding SW2 switch

// AN0 - 3 has 10kOhm resistor in serial, include 3.9kOhm to make it as

// voltage divider

// AN4 - 5 are direct GPIO pins from atmega1280 and they are the latest pins

// next to SW2 switch

// Look more ArduCopter Wiki for voltage dividers and other ports

#define AN0 54 // resistor, vdiv use, divider 1 closest to relay

#define AN1 55 // resistor, vdiv use, divider 2

#define AN2 56 // resistor, vdiv use, divider 3

#define AN3 57 // resistor, vdiv use, divider 4 closest to SW2

#define AN4 58 // direct GPIO pin, default as analog input, next to SW2

// switch

#define AN5 59 // direct GPIO pin, default as analog input, next to SW2

// switch

#define AN6 60 // direct GPIO pin, default as analog input, close to

// Pressure sensor, Expansion Ports

#define AN7 61 // direct GPIO pin, default as analog input, close to

// Pressure sensor, Expansion Ports

// AN8 - 15 are located at edge of IMU PCB "above" pressure sensor and

// Expansion port

// AN8 - 15 PINs are not connected anywhere, they are located as last 8 pins

// on edge of the board above Expansion Ports

// even pins (8,10,12,14) are at edge of board, Odd pins (9,11,13,15) are on

// inner row

#define AN8 62 // NC

#define AN9 63 // NC

#define AN10 64 // NC

#define AN11 65 // NC

#define AN12 66 // NC

#define AN13 67 // NC

#define AN14 68 // NC

#define AN15 69 // NC

#define RELAY\_APM1\_PIN 47

#define RELAY\_APM2\_PIN 13

#define PIEZO\_PIN AN5 //Last pin on the back ADC connector

// RADIANS

#define RADX100 0.000174532925

#define DEGX100 5729.57795

// EEPROM addresses

#define EEPROM\_MAX\_ADDR 4096

// parameters get the first 1536 bytes of EEPROM, remainder is for waypoints

#define WP\_START\_BYTE 0x600 // where in memory home WP is stored + all other

// WP

#define WP\_SIZE 15

// fence points are stored at the end of the EEPROM

#define MAX\_FENCEPOINTS 6

#define FENCE\_WP\_SIZE sizeof(Vector2l)

#define FENCE\_START\_BYTE (EEPROM\_MAX\_ADDR-(MAX\_FENCEPOINTS\*FENCE\_WP\_SIZE))

#define MAX\_WAYPOINTS ((FENCE\_START\_BYTE - WP\_START\_BYTE) / WP\_SIZE) - 1 // -

// 1

// to

// be

// safe

// mark a function as not to be inlined

#define NOINLINE \_\_attribute\_\_((noinline))

// IMU selection

#define CONFIG\_IMU\_OILPAN 1

#define CONFIG\_IMU\_MPU6000 2

// MPU6K Filter Rates

# define MPU6K\_DEFAULT\_FILTER 0

# define MPU6K\_5HZ\_FILTER 5

# define MPU6K\_10HZ\_FILTER 10

# define MPU6K\_20HZ\_FILTER 20

# define MPU6K\_42HZ\_FILTER 42

# define MPU6K\_98HZ\_FILTER 98

// APM Hardware selection

#define APM\_HARDWARE\_APM1 1

#define APM\_HARDWARE\_APM2 2

#define AP\_BARO\_BMP085 1

#define AP\_BARO\_MS5611 2

// Error message sub systems and error codes

#define ERROR\_SUBSYSTEM\_MAIN 1

#define ERROR\_SUBSYSTEM\_RADIO 2

#define ERROR\_SUBSYSTEM\_COMPASS 3

#define ERROR\_SUBSYSTEM\_OPTFLOW 4

#define ERROR\_SUBSYSTEM\_FAILSAFE 5

// general error codes

#define ERROR\_CODE\_ERROR\_RESOLVED 0

#define ERROR\_CODE\_FAILED\_TO\_INITIALISE 1

// subsystem specific error codes -- radio

#define ERROR\_CODE\_RADIO\_LATE\_FRAME 2

// subsystem specific error codes -- failsafe

#define ERROR\_CODE\_FAILSAFE\_THROTTLE 2

#define ERROR\_CODE\_FAILSAFE\_BATTERY 3

#define ERROR\_CODE\_FAILSAFE\_WATCHDOG 4

#endif // \_DEFINES\_H

## events.pde

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

/\*

\* This event will be called when the failsafe changes

\* boolean failsafe reflects the current state

\*/

static void failsafe\_on\_event()

{

// if motors are not armed there is nothing to do

if( !motors.armed() ) {

return;

}

// log the error to the dataflash

Log\_Write\_Error(ERROR\_SUBSYSTEM\_FAILSAFE, ERROR\_CODE\_FAILSAFE\_THROTTLE);

// This is how to handle a failsafe.

switch(control\_mode) {

case STABILIZE:

case ACRO:

// if throttle is zero disarm motors

if (g.rc\_3.control\_in == 0) {

init\_disarm\_motors();

}else if(ap.home\_is\_set == true && home\_distance > g.waypoint\_radius) {

set\_mode(RTL);

}else{

// We have no GPS or are very close to home so we will land

set\_mode(LAND);

}

break;

case AUTO:

// failsafe\_throttle is 1 do RTL, 2 means continue with the mission

if (g.failsafe\_throttle == FS\_THR\_ENABLED\_ALWAYS\_RTL) {

if(home\_distance > g.waypoint\_radius) {

set\_mode(RTL);

}else{

// We are very close to home so we will land

set\_mode(LAND);

}

}

// if failsafe\_throttle is 2 (i.e. FS\_THR\_ENABLED\_CONTINUE\_MISSION) no need to do anything

break;

default:

if(ap.home\_is\_set == true && home\_distance > g.waypoint\_radius) {

set\_mode(RTL);

}else{

// We have no GPS or are very close to home so we will land

set\_mode(LAND);

}

break;

}

}

// failsafe\_off\_event - respond to radio contact being regained

// we must be in AUTO, LAND or RTL modes

// or Stabilize or ACRO mode but with motors disarmed

static void failsafe\_off\_event()

{

// no need to do anything except log the error as resolved

// user can now override roll, pitch, yaw and throttle and even use flight mode switch to restore previous flight mode

Log\_Write\_Error(ERROR\_SUBSYSTEM\_FAILSAFE, ERROR\_CODE\_ERROR\_RESOLVED);

}

static void low\_battery\_event(void)

{

// failsafe check

if (g.failsafe\_battery\_enabled && !ap.low\_battery && motors.armed()) {

switch(control\_mode) {

case STABILIZE:

case ACRO:

// if throttle is zero disarm motors

if (g.rc\_3.control\_in == 0) {

init\_disarm\_motors();

}else{

set\_mode(LAND);

}

break;

case AUTO:

if(ap.home\_is\_set == true && home\_distance > g.waypoint\_radius) {

set\_mode(RTL);

}else{

// We have no GPS or are very close to home so we will land

set\_mode(LAND);

}

break;

default:

set\_mode(LAND);

break;

}

}

// set the low battery flag

set\_low\_battery(true);

// warn the ground station and log to dataflash

gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Low Battery!"));

Log\_Write\_Error(ERROR\_SUBSYSTEM\_FAILSAFE, ERROR\_CODE\_FAILSAFE\_BATTERY);

#if COPTER\_LEDS == ENABLED

if ( bitRead(g.copter\_leds\_mode, 3) ) { // Only Activate if a battery is connected to avoid alarm on USB only

piezo\_on();

}

#endif // COPTER\_LEDS

}

static void update\_events() // Used for MAV\_CMD\_DO\_REPEAT\_SERVO and MAV\_CMD\_DO\_REPEAT\_RELAY

{

if(event\_repeat == 0 || (millis() - event\_timer) < event\_delay)

return;

if(event\_repeat != 0) { // event\_repeat = -1 means repeat forever

event\_timer = millis();

if (event\_id >= CH\_5 && event\_id <= CH\_8) {

if(event\_repeat%2) {

APM\_RC.OutputCh(event\_id, event\_value); // send to Servos

} else {

APM\_RC.OutputCh(event\_id, event\_undo\_value);

}

}

if (event\_id == RELAY\_TOGGLE) {

relay.toggle();

}

if (event\_repeat > 0) {

event\_repeat--;

}

}

}

## failsafe.pde

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

//

// failsafe support

// Andrew Tridgell, December 2011

//

// our failsafe strategy is to detect main loop lockup and disarm the motors

//

static bool failsafe\_enabled = true;

static uint16\_t failsafe\_last\_mainLoop\_count;

static uint32\_t failsafe\_last\_timestamp;

static bool in\_failsafe;

//

// failsafe\_enable - enable failsafe

//

void failsafe\_enable()

{

failsafe\_enabled = true;

failsafe\_last\_timestamp = micros();

}

//

// failsafe\_disable - used when we know we are going to delay the mainloop significantly

//

void failsafe\_disable()

{

failsafe\_enabled = false;

}

//

// failsafe\_check - this function is called from the core timer interrupt at 1kHz.

//

void failsafe\_check(uint32\_t tnow)

{

if (mainLoop\_count != failsafe\_last\_mainLoop\_count) {

// the main loop is running, all is OK

failsafe\_last\_mainLoop\_count = mainLoop\_count;

failsafe\_last\_timestamp = tnow;

in\_failsafe = false;

return;

}

if (failsafe\_enabled && tnow - failsafe\_last\_timestamp > 2000000) {

// motors are running but we have gone 2 second since the

// main loop ran. That means we're in trouble and should

// disarm the motors.

in\_failsafe = true;

}

if (failsafe\_enabled && in\_failsafe && tnow - failsafe\_last\_timestamp > 1000000) {

// disarm motors every second

failsafe\_last\_timestamp = tnow;

if(motors.armed()) {

motors.armed(false);

set\_armed(true);

motors.output();

Log\_Write\_Error(ERROR\_SUBSYSTEM\_FAILSAFE, ERROR\_CODE\_FAILSAFE\_WATCHDOG);

}

}

}

## flip

// 2010 Jose Julio

// 2011 Adapted and updated for AC2 by Jason Short

//

// Automatic Acrobatic Procedure (AAP) v1 : Roll flip

// State machine aproach:

// Some states are fixed commands (for a fixed time)

// Some states are fixed commands (until some IMU condition)

// Some states include controls inside

uint8\_t flip\_timer;

uint8\_t flip\_state;

#define AAP\_THR\_INC 170

#define AAP\_THR\_DEC 120

#define AAP\_ROLL\_OUT 2000

static int8\_t flip\_dir;

void init\_flip()

{

if(false == ap.do\_flip) {

ap.do\_flip = true;

flip\_state = 0;

flip\_dir = (ahrs.roll\_sensor >= 0) ? 1 : -1;

Log\_Write\_Event(DATA\_BEGIN\_FLIP);

}

}

void roll\_flip()

{

// Pitch

//g.rc\_2.servo\_out = get\_stabilize\_pitch(g.rc\_2.control\_in);

get\_stabilize\_pitch(g.rc\_2.control\_in);

int32\_t roll = ahrs.roll\_sensor \* flip\_dir;

// Roll State machine

switch (flip\_state) {

case 0:

if (roll < 4500) {

// Roll control

g.rc\_1.servo\_out = AAP\_ROLL\_OUT \* flip\_dir;

set\_throttle\_out(g.rc\_3.control\_in + AAP\_THR\_INC, false);

}else{

flip\_state++;

}

break;

case 1:

if((roll >= 4500) || (roll < -9000)) {

#if FRAME\_CONFIG == HELI\_FRAME

g.rc\_1.servo\_out = get\_heli\_rate\_roll(40000 \* flip\_dir);

#else

g.rc\_1.servo\_out = get\_rate\_roll(40000 \* flip\_dir);

#endif // HELI\_FRAME

set\_throttle\_out(g.rc\_3.control\_in - AAP\_THR\_DEC, false);

}else{

flip\_state++;

flip\_timer = 0;

}

break;

case 2:

if (flip\_timer < 100) {

//g.rc\_1.servo\_out = get\_stabilize\_roll(g.rc\_1.control\_in);

get\_stabilize\_roll(g.rc\_1.control\_in);

set\_throttle\_out(g.rc\_3.control\_in + AAP\_THR\_INC, false);

flip\_timer++;

}else{

Log\_Write\_Event(DATA\_END\_FLIP);

ap.do\_flip = false;

flip\_state = 0;

}

break;

}

}

## inertia

/// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

// read\_inertia - read inertia in from accelerometers

static void read\_inertia()

{

#if INERTIAL\_NAV\_XY == ENABLED || INERTIAL\_NAV\_Z == ENABLED

static uint8\_t log\_counter\_inav = 0;

// inertial altitude estimates

inertial\_nav.update(G\_Dt);

if( motors.armed() && g.log\_bitmask & MASK\_LOG\_INAV ) {

log\_counter\_inav++;

if( log\_counter\_inav >= 10 ) {

log\_counter\_inav = 0;

Log\_Write\_INAV();

}

}

#endif

}

## leds

/// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

static void update\_lights()

{

switch(led\_mode) {

case NORMAL\_LEDS:

update\_motor\_light();

update\_GPS\_light();

break;

case SAVE\_TRIM\_LEDS:

dancing\_light();

break;

}

}

static void update\_GPS\_light(void)

{

// GPS LED on if we have a fix or Blink GPS LED if we are receiving data

// ---------------------------------------------------------------------

switch (g\_gps->status()) {

case (2):

if(ap.home\_is\_set) { // JLN update

digitalWriteFast(C\_LED\_PIN, LED\_ON); //Turn LED C on when gps has valid fix AND home is set.

} else {

digitalWriteFast(C\_LED\_PIN, LED\_OFF);

}

break;

case (1):

if (g\_gps->valid\_read == true) {

ap\_system.GPS\_light = !ap\_system.GPS\_light; // Toggle light on and off to indicate gps messages being received, but no GPS fix lock

if (ap\_system.GPS\_light) {

digitalWriteFast(C\_LED\_PIN, LED\_OFF);

}else{

digitalWriteFast(C\_LED\_PIN, LED\_ON);

}

g\_gps->valid\_read = false;

}

break;

default:

digitalWriteFast(C\_LED\_PIN, LED\_OFF);

break;

}

}

static void update\_motor\_light(void)

{

if(motors.armed() == false) {

ap\_system.motor\_light = !ap\_system.motor\_light;

// blink

if(ap\_system.motor\_light) {

digitalWriteFast(A\_LED\_PIN, LED\_ON);

}else{

digitalWriteFast(A\_LED\_PIN, LED\_OFF);

}

}else{

if(!ap\_system.motor\_light) {

ap\_system.motor\_light = true;

digitalWriteFast(A\_LED\_PIN, LED\_ON);

}

}

}

static void dancing\_light()

{

static byte step;

if (step++ == 3)

step = 0;

switch(step)

{

case 0:

digitalWriteFast(C\_LED\_PIN, LED\_OFF);

digitalWriteFast(A\_LED\_PIN, LED\_ON);

break;

case 1:

digitalWriteFast(A\_LED\_PIN, LED\_OFF);

digitalWriteFast(B\_LED\_PIN, LED\_ON);

break;

case 2:

digitalWriteFast(B\_LED\_PIN, LED\_OFF);

digitalWriteFast(C\_LED\_PIN, LED\_ON);

break;

}

}

static void clear\_leds()

{

digitalWriteFast(A\_LED\_PIN, LED\_OFF);

digitalWriteFast(B\_LED\_PIN, LED\_OFF);

digitalWriteFast(C\_LED\_PIN, LED\_OFF);

ap\_system.motor\_light = false;

led\_mode = NORMAL\_LEDS;

}

/////////////////////////////////////////////////////////////////////////////////////////////

// Copter LEDS by Robert Lefebvre

// Based on the work of U4eake, Bill Sanford, Max Levine, and Oliver

// g.copter\_leds\_mode controls the copter leds function via bitmath

// Zeroeth bit turns motor leds on and off: 00000001

// First bit turns GPS function on and off: 00000010

// Second bit turns Aux function on and off: 00000100

// Third bit turns on Beeper (legacy Piezo) function: 00001000

// Fourth bit toggles between Fast Flash or Oscillate on Low Battery: 00010000 (0) does Fast Flash, (1) does Oscillate

// Fifth bit causes motor LEDs to Nav Blink: 00100000

// Sixth bit causes GPS LEDs to Nav Blink: 01000000

// This code is written in order to be backwards compatible with the old Motor\_LEDS code

// I hope to include at least some of the Show\_LEDS code in the future

// copter\_leds\_GPS\_blink controls the blinking of the GPS LEDS

// copter\_leds\_motor\_blink controls the blinking of the motor LEDS

// Piezo Code and beeps once on Startup to verify operation

// Piezo Enables Tone on reaching low battery or current alert

/////////////////////////////////////////////////////////////////////////////////////////////

#if COPTER\_LEDS == ENABLED

static void update\_copter\_leds(void)

{

if (g.copter\_leds\_mode == 0) {

copter\_leds\_reset(); //method of reintializing LED state

}

if ( bitRead(g.copter\_leds\_mode, 0) ) {

if (motors.armed() == true) {

if (ap.low\_battery == true) {

if ( bitRead(g.copter\_leds\_mode, 4 )) {

copter\_leds\_oscillate(); //if motors are armed, but battery level is low, motor leds fast blink

} else {

copter\_leds\_fast\_blink(); //if motors are armed, but battery level is low, motor leds oscillate

}

} else if ( !bitRead(g.copter\_leds\_mode, 5 ) ) {

copter\_leds\_on(); //if motors are armed, battery level OK, all motor leds ON

} else if ( bitRead(g.copter\_leds\_mode, 5 ) ) {

if ( copter\_leds\_nav\_blink >0 ) {

copter\_leds\_slow\_blink(); //if nav command was seen, blink LEDs.

} else {

copter\_leds\_on();

}

}

} else {

copter\_leds\_slow\_blink(); //if motors are not armed, blink motor leds

}

}

if ( bitRead(g.copter\_leds\_mode, 1) ) {

// GPS LED on if we have a fix or Blink GPS LED if we are receiving data

// ---------------------------------------------------------------------

switch (g\_gps->status()) {

case (2):

if(ap.home\_is\_set) {

if ( !bitRead(g.copter\_leds\_mode, 6 ) ) {

copter\_leds\_GPS\_on(); //Turn GPS LEDs on when gps has valid fix AND home is set

} else if (bitRead(g.copter\_leds\_mode, 6 ) ) {

if ( copter\_leds\_nav\_blink >0 ) {

copter\_leds\_GPS\_slow\_blink(); //if nav command was seen, blink LEDs.

} else {

copter\_leds\_GPS\_on();

}

}

} else {

copter\_leds\_GPS\_fast\_blink(); //if GPS has fix, but home is not set, blink GPS LED fast

}

break;

case (1):

copter\_leds\_GPS\_slow\_blink(); //if GPS has valid reads, but no fix, blink GPS LED slow

break;

default:

copter\_leds\_GPS\_off(); //if no valid GPS signal, turn GPS LED off

break;

}

}

if ( bitRead(g.copter\_leds\_mode, 2) ) {

if (200 <= g.rc\_7.control\_in && g.rc\_7.control\_in < 400) {

copter\_leds\_aux\_on(); //if sub-control of Ch7 is high, turn Aux LED on

} else if (g.rc\_7.control\_in < 200) {

copter\_leds\_aux\_off(); //if sub-control of Ch7 is low, turn Aux LED off

}

}

}

static void copter\_leds\_reset(void) {

digitalWriteFast(COPTER\_LED\_1, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_2, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_3, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_4, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_5, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_6, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_7, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_8, COPTER\_LED\_OFF);

}

static void copter\_leds\_on(void) {

if ( !bitRead(g.copter\_leds\_mode, 2) ) {

digitalWriteFast(COPTER\_LED\_1, COPTER\_LED\_ON);

}

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

if ( !bitRead(g.copter\_leds\_mode, 3) ) {

digitalWriteFast(COPTER\_LED\_2, COPTER\_LED\_ON);

}

#else

digitalWriteFast(COPTER\_LED\_2, COPTER\_LED\_ON);

#endif

if ( !bitRead(g.copter\_leds\_mode, 1) ) {

digitalWriteFast(COPTER\_LED\_3, COPTER\_LED\_ON);

}

digitalWriteFast(COPTER\_LED\_4, COPTER\_LED\_ON);

digitalWriteFast(COPTER\_LED\_5, COPTER\_LED\_ON);

digitalWriteFast(COPTER\_LED\_6, COPTER\_LED\_ON);

digitalWriteFast(COPTER\_LED\_7, COPTER\_LED\_ON);

digitalWriteFast(COPTER\_LED\_8, COPTER\_LED\_ON);

}

static void copter\_leds\_off(void) {

if ( !bitRead(g.copter\_leds\_mode, 2) ) {

digitalWriteFast(COPTER\_LED\_1, COPTER\_LED\_OFF);

}

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

if ( !bitRead(g.copter\_leds\_mode, 3) ) {

digitalWriteFast(COPTER\_LED\_2, COPTER\_LED\_OFF);

}

#else

digitalWriteFast(COPTER\_LED\_2, COPTER\_LED\_OFF);

#endif

if ( !bitRead(g.copter\_leds\_mode, 1) ) {

digitalWriteFast(COPTER\_LED\_3, COPTER\_LED\_OFF);

}

digitalWriteFast(COPTER\_LED\_4, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_5, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_6, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_7, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_8, COPTER\_LED\_OFF);

}

static void copter\_leds\_slow\_blink(void) {

copter\_leds\_motor\_blink++; // this increments once every 1/10 second because it is in the 10hz loop

if ( 0 < copter\_leds\_motor\_blink && copter\_leds\_motor\_blink < 6 ) { // when the counter reaches 5 (1/2 sec), then toggle the leds

copter\_leds\_off();

if ( bitRead(g.copter\_leds\_mode, 5 ) && !bitRead(g.copter\_leds\_mode, 6 ) && copter\_leds\_nav\_blink >0 ) { // if blinking is called by the Nav Blinker...

copter\_leds\_nav\_blink--; // decrement the Nav Blink counter

}

}else if (5 < copter\_leds\_motor\_blink && copter\_leds\_motor\_blink < 11) {

copter\_leds\_on();

}

else copter\_leds\_motor\_blink = 0; // start blink cycle again

}

static void copter\_leds\_fast\_blink(void) {

copter\_leds\_motor\_blink++; // this increments once every 1/10 second because it is in the 10hz loop

if ( 0 < copter\_leds\_motor\_blink && copter\_leds\_motor\_blink < 3 ) { // when the counter reaches 3 (1/5 sec), then toggle the leds

copter\_leds\_on();

}else if (2 < copter\_leds\_motor\_blink && copter\_leds\_motor\_blink < 5) {

copter\_leds\_off();

}

else copter\_leds\_motor\_blink = 0; // start blink cycle again

}

static void copter\_leds\_oscillate(void) {

copter\_leds\_motor\_blink++; // this increments once every 1/10 second because it is in the 10hz loop

if ( 0 < copter\_leds\_motor\_blink && copter\_leds\_motor\_blink < 3 ) { // when the counter reaches 3 (1/5 sec), then toggle the leds

if ( !bitRead(g.copter\_leds\_mode, 2) ) {

digitalWriteFast(COPTER\_LED\_1, COPTER\_LED\_ON);

}

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

if ( !bitRead(g.copter\_leds\_mode, 3) ) {

digitalWriteFast(COPTER\_LED\_2, COPTER\_LED\_ON);

}

#else

digitalWriteFast(COPTER\_LED\_2, COPTER\_LED\_ON);

#endif

if ( !bitRead(g.copter\_leds\_mode, 1) ) {

digitalWriteFast(COPTER\_LED\_3, COPTER\_LED\_OFF);

}

digitalWriteFast(COPTER\_LED\_4, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_5, COPTER\_LED\_ON);

digitalWriteFast(COPTER\_LED\_6, COPTER\_LED\_ON);

digitalWriteFast(COPTER\_LED\_7, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_8, COPTER\_LED\_OFF);

}else if (2 < copter\_leds\_motor\_blink && copter\_leds\_motor\_blink < 5) {

if ( !bitRead(g.copter\_leds\_mode, 2) ) {

digitalWriteFast(COPTER\_LED\_1, COPTER\_LED\_OFF);

}

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

if ( !bitRead(g.copter\_leds\_mode, 3) ) {

digitalWriteFast(COPTER\_LED\_2, COPTER\_LED\_OFF);

}

#else

digitalWriteFast(COPTER\_LED\_2, COPTER\_LED\_OFF);

#endif

if ( !bitRead(g.copter\_leds\_mode, 1) ) {

digitalWriteFast(COPTER\_LED\_3, COPTER\_LED\_ON);

}

digitalWriteFast(COPTER\_LED\_4, COPTER\_LED\_ON);

digitalWriteFast(COPTER\_LED\_5, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_6, COPTER\_LED\_OFF);

digitalWriteFast(COPTER\_LED\_7, COPTER\_LED\_ON);

digitalWriteFast(COPTER\_LED\_8, COPTER\_LED\_ON);

}

else copter\_leds\_motor\_blink = 0; // start blink cycle again

}

static void copter\_leds\_GPS\_on(void) {

digitalWriteFast(COPTER\_LED\_3, COPTER\_LED\_ON);

}

static void copter\_leds\_GPS\_off(void) {

digitalWriteFast(COPTER\_LED\_3, COPTER\_LED\_OFF);

}

static void copter\_leds\_GPS\_slow\_blink(void) {

copter\_leds\_GPS\_blink++; // this increments once every 1/10 second because it is in the 10hz loop

if ( 0 < copter\_leds\_GPS\_blink && copter\_leds\_GPS\_blink < 6 ) { // when the counter reaches 5 (1/2 sec), then toggle the leds

copter\_leds\_GPS\_off();

if ( bitRead(g.copter\_leds\_mode, 6 ) && copter\_leds\_nav\_blink >0 ) { // if blinking is called by the Nav Blinker...

copter\_leds\_nav\_blink--; // decrement the Nav Blink counter

}

}else if (5 < copter\_leds\_GPS\_blink && copter\_leds\_GPS\_blink < 11) {

copter\_leds\_GPS\_on();

}

else copter\_leds\_GPS\_blink = 0; // start blink cycle again

}

static void copter\_leds\_GPS\_fast\_blink(void) {

copter\_leds\_GPS\_blink++; // this increments once every 1/10 second because it is in the 10hz loop

if ( 0 < copter\_leds\_GPS\_blink && copter\_leds\_GPS\_blink < 3 ) { // when the counter reaches 3 (1/5 sec), then toggle the leds

copter\_leds\_GPS\_off();

}else if (2 < copter\_leds\_GPS\_blink && copter\_leds\_GPS\_blink < 5) {

copter\_leds\_GPS\_on();

}

else copter\_leds\_GPS\_blink = 0; // start blink cycle again

}

static void copter\_leds\_aux\_off(void){

digitalWriteFast(COPTER\_LED\_1, COPTER\_LED\_OFF);

}

static void copter\_leds\_aux\_on(void){

digitalWriteFast(COPTER\_LED\_1, COPTER\_LED\_ON);

}

void piezo\_on(){

digitalWriteFast(PIEZO\_PIN,HIGH);

}

void piezo\_off(){

digitalWriteFast(PIEZO\_PIN,LOW);

}

void piezo\_beep(){ // Note! This command should not be used in time sensitive loops

piezo\_on();

delay(100);

piezo\_off();

}

#endif //COPTER\_LEDS

## limits

// Main state machine loop for AP\_Limits. Called from slow or superslow loop.

#if AP\_LIMITS == ENABLED

uint8\_t lim\_state = 0, lim\_old\_state = 0;

void set\_recovery\_home\_alt() {

uint32\_t return\_altitude\_cm\_ahl = 0; // in centimeters above home level.

uint32\_t amin\_meters\_ahl, amax\_meters\_ahl;

// for flying vehicles only

if (altitude\_limit.enabled()) {

amin\_meters\_ahl = (uint32\_t) (altitude\_limit.min\_alt());

amax\_meters\_ahl = (uint32\_t) (altitude\_limit.max\_alt());

// See if we have a meaningful setting

if (amax\_meters\_ahl && ((amax\_meters\_ahl - amin\_meters\_ahl) > 1)) {

// there is a max\_alt set

// set a return altitude that is halfway between the minimum and maximum altitude setting.

// return\_altitude is in centimeters, not meters, so we multiply

return\_altitude\_cm\_ahl = (uint32\_t) (home.alt + (100 \* (uint16\_t) ((amax\_meters\_ahl - amin\_meters\_ahl) / 2)));

}

} else {

return\_altitude\_cm\_ahl = (uint32\_t) (home.alt + g.rtl\_altitude);

}

// final sanity check

// if our return is less than 4 meters from ground, set it to 4m, to clear "people" height.

if ((return\_altitude\_cm\_ahl - (uint32\_t) home.alt) < 400) {

return\_altitude\_cm\_ahl = home.alt + 400;

}

guided\_WP.id = 0;

guided\_WP.p1 = 0;

guided\_WP.options = 0;

guided\_WP.lat = home.lat;

guided\_WP.lng = home.lng;

guided\_WP.alt = return\_altitude\_cm\_ahl;

}

static void limits\_loop() {

lim\_state = limits.state();

// Use limits channel to determine LIMITS\_ENABLED or LIMITS\_DISABLED state

if (lim\_state != LIMITS\_DISABLED && limits.channel() !=0 && APM\_RC.InputCh(limits.channel()-1) < LIMITS\_ENABLE\_PWM) {

limits.set\_state(LIMITS\_DISABLED);

}

else if (lim\_state == LIMITS\_DISABLED && limits.channel() !=0 && APM\_RC.InputCh(limits.channel()-1) >= LIMITS\_ENABLE\_PWM) {

limits.set\_state(LIMITS\_ENABLED);

}

if ((uint32\_t) millis() - (uint32\_t) limits.last\_status\_update > 1000) { // more than a second has passed - time for an update

gcs\_send\_message(MSG\_LIMITS\_STATUS);

}

if (lim\_state != lim\_old\_state) { // state changed

lim\_old\_state = lim\_state; // we only use lim\_oldstate here, for reporting purposes. So, reset it.

gcs\_send\_message(MSG\_LIMITS\_STATUS);

if (limits.debug()) switch (lim\_state) {

case LIMITS\_INIT: gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits - State change: INIT")); break;

case LIMITS\_DISABLED: gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits - State change: DISABLED")); break;

case LIMITS\_ENABLED: gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits - State change: ENABLED")); break;

case LIMITS\_TRIGGERED: gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits - State change: TRIGGERED")); break;

case LIMITS\_RECOVERING: gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits - State change: RECOVERING")); break;

case LIMITS\_RECOVERED: gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits - State change: RECOVERED")); break;

default: gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits - State change: UNKNOWN")); break;

}

}

switch (limits.state()) {

// have not initialized yet

case LIMITS\_INIT:

if (limits.init()) { // initialize system

// See what the "master" on/off swith is and go to the appropriate start state

if (!limits.enabled()) {

limits.set\_state(LIMITS\_DISABLED);

}

else {

limits.set\_state(LIMITS\_ENABLED);

}

}

break;

// We have been switched off

case LIMITS\_DISABLED:

// check if we have been switched on

if (limits.enabled()) {

limits.set\_state(LIMITS\_ENABLED);

break;

}

break;

// Limits module is enabled

case LIMITS\_ENABLED:

// check if we've been switched off

if (!limits.enabled()) {

limits.set\_state(LIMITS\_DISABLED);

break;

}

// Until motors are armed, do nothing, just wait in ENABLED state

if (!motors.armed()) {

// we are waiting for motors to arm

// do nothing

break;

}

bool required\_only;

required\_only = (limits.last\_clear == 0); // if we haven't yet 'cleared' all limits, check required limits only

// check if any limits have been breached and trigger if they have

if (limits.check\_triggered(required\_only)) {

//

// TRIGGER - BREACH OF LIMITS

//

// make a note of which limits triggered, so if we know if we recovered them

limits.mods\_recovering = limits.mods\_triggered;

limits.last\_action = 0;

limits.last\_trigger = millis();

limits.breach\_count++;

limits.set\_state(LIMITS\_TRIGGERED);

break;

}

if (motors.armed() && limits.enabled() && !limits.mods\_triggered) {

// All clear.

if (limits.debug()) gcs\_send\_text\_P(SEVERITY\_LOW, PSTR("Limits - All Clear"));

limits.last\_clear = millis();

}

break;

// Limits have been triggered

case LIMITS\_TRIGGERED:

// check if we've been switched off

if (!limits.enabled()) {

limits.set\_state(LIMITS\_DISABLED);

break;

}

#if LIMITS\_TRIGGERED\_PIN > 0

digitalWrite(LIMITS\_TRIGGERED\_PIN, HIGH);

#endif

if (limits.debug()) {

if (limits.mods\_triggered & LIMIT\_GPSLOCK) gcs\_send\_text\_P(SEVERITY\_LOW, PSTR("!GPSLock"));

if (limits.mods\_triggered & LIMIT\_GEOFENCE) gcs\_send\_text\_P(SEVERITY\_LOW, PSTR("!Geofence"));

if (limits.mods\_triggered & LIMIT\_ALTITUDE) gcs\_send\_text\_P(SEVERITY\_LOW, PSTR("!Altitude"));

}

// If the motors are not armed, we have triggered pre-arm checks. Do nothing

if (motors.armed() == false) {

limits.set\_state(LIMITS\_ENABLED); // go back to checking limits

break;

}

// If we are triggered but no longer in breach, that means we recovered

// somehow, via auto recovery or pilot action

if (!limits.check\_all()) {

limits.last\_recovery = millis();

limits.set\_state(LIMITS\_RECOVERED);

break;

}

else {

limits.set\_state(LIMITS\_RECOVERING);

limits.last\_action = 0; // reset timer

// We are about to take action on a real breach. Make sure we notify immediately

gcs\_send\_message(MSG\_LIMITS\_STATUS);

break;

}

break;

// Take action to recover

case LIMITS\_RECOVERING:

// If the motors are not armed, we have triggered pre-arm checks. Do nothing

if (motors.armed() == false) {

limits.set\_state(LIMITS\_ENABLED); // go back to checking limits

break;

}

// check if we've been switched off

if (!limits.enabled() && limits.old\_mode\_switch == oldSwitchPosition) {

limits.old\_mode\_switch = 0;

reset\_control\_switch();

limits.set\_state(LIMITS\_DISABLED);

break;

}

// Still need action?

if (limits.check\_all() == 0) { // all triggers clear

limits.set\_state(LIMITS\_RECOVERED);

break;

}

if (limits.mods\_triggered != limits.mods\_recovering) { // if any \*new\* triggers, hit the trigger again

//

// TRIGGER - BREACH OF LIMITS

//

// make a note of which limits triggered, so if we know if we recovered them

limits.mods\_recovering = limits.mods\_triggered;

limits.last\_action = 0;

limits.last\_trigger = millis();

limits.breach\_count++;

limits.set\_state(LIMITS\_TRIGGERED);

limits.set\_state(LIMITS\_TRIGGERED);

break;

}

// Recovery Action

// if there was no previous action, take action, take note of time send GCS.

if (limits.last\_action == 0) {

// save mode switch

limits.old\_mode\_switch = oldSwitchPosition;

// // Take action

// // This ensures no "radical" RTL, like a full throttle take-off,happens if something triggers at ground level

// if ((uint32\_t) current\_loc.alt < ((uint32\_t)home.alt \* 200) ) { // we're under 2m (200cm), already at "people" height or on the ground

// if (limits.debug()) gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits Action: near ground - do nothing"));

// // TODO: Will this work for a plane? Does it make sense in general?

//

// //set\_mode(LAND);

// limits.last\_action = millis(); // start counter

// gcs\_send\_message(MSG\_LIMITS\_STATUS);

//

// break;

// }

// TODO: This applies only to planes - hold for porting

// if (control\_mode == MANUAL && g.auto\_trim) {

// // make sure we don't auto trim the surfaces on this change

// control\_mode = STABILIZE;

// }

switch (limits.recmode()) {

case 0: // RTL mode

if (limits.debug()) gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits Action - RTL"));

set\_mode(RTL);

limits.last\_action = millis();

gcs\_send\_message(MSG\_LIMITS\_STATUS);

break;

case 1: // Bounce mode

if (limits.debug()) gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits Action - bounce mode, POSITION"));

// ALT\_HOLD gives us yaw hold, roll& pitch hold and throttle hold.

// It is like position hold, but without manual throttle control.

//set\_recovery\_home\_alt();

set\_mode(POSITION);

set\_throttle\_mode(THROTTLE\_AUTO);

limits.last\_action = millis();

gcs\_send\_message(MSG\_LIMITS\_STATUS);

break;

}

break;

}

// In bounce mode, take control for 3 seconds, and then wait for the pilot to make us "safe".

// If the vehicle does not recover, the escalation action will trigger.

if (limits.recmode() == 1) {

if (control\_mode == POSITION && ((uint32\_t)millis() - (uint32\_t)limits.last\_action) > 3000) {

if (limits.debug()) gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits Recovery Bounce: Returning control to pilot"));

set\_mode(STABILIZE);

} else if (control\_mode == STABILIZE && ((uint32\_t)millis() - (uint32\_t)limits.last\_action) > 6000) {

// after 3 more seconds, reset action counter to take action again

limits.last\_action = 0;

}

}

// ESCALATE We have not recovered after 2 minutes of recovery action

if (((uint32\_t)millis() - (uint32\_t)limits.last\_action) > 120000 ) {

// TODO: Secondary recovery

if (limits.debug()) gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits Recovery Escalation: RTL"));

set\_mode(RTL);

limits.last\_action = millis();

break;

}

break;

// Have recovered, relinquish control and re-enable

case LIMITS\_RECOVERED:

// check if we've been switched off

if (!limits.enabled()) {

limits.set\_state(LIMITS\_DISABLED);

break;

}

#if LIMITS\_TRIGGERED\_PIN > 0

digitalWrite(LIMITS\_TRIGGERED\_PIN, LOW);

#endif

// Reset action counter

limits.last\_action = 0;

if (((uint32\_t)millis() - (uint32\_t)limits.last\_recovery) > (uint32\_t)(limits.safetime() \* 1000)) { // Wait "safetime" seconds of recovery before we give back control

// Our recovery action worked.

limits.set\_state(LIMITS\_ENABLED);

// Switch to stabilize

if (limits.debug()) gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits - Returning controls"));

set\_mode(STABILIZE); limits.last\_recovery = millis();

break;

}

break;

default:

if (limits.debug()) gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits: unknown state"));

break;

}

}

// This function below, should really be in the AP\_Limits class, but it is impossible to untangle the mavlink includes.

void limits\_send\_mavlink\_status(mavlink\_channel\_t chan) {

limits.last\_status\_update = millis();

if (limits.enabled()) {

mavlink\_msg\_limits\_status\_send(chan,

(uint8\_t) limits.state(),

(uint32\_t) limits.last\_trigger,

(uint32\_t) limits.last\_action,

(uint32\_t) limits.last\_recovery,

(uint32\_t) limits.last\_clear,

(uint16\_t) limits.breach\_count,

(LimitModuleBits) limits.mods\_enabled,

(LimitModuleBits) limits.mods\_required,

(LimitModuleBits) limits.mods\_triggered);

}

}

#endif

## motors

/// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

// 10 = 1 second

#define ARM\_DELAY 20

#define DISARM\_DELAY 20

#define AUTO\_TRIM\_DELAY 100

// called at 10hz

static void arm\_motors()

{

static int16\_t arming\_counter;

// don't allow arming/disarming in anything but manual

if (g.rc\_3.control\_in > 0) {

arming\_counter = 0;

return;

}

if ((control\_mode > ACRO) && ((control\_mode != TOY\_A) && (control\_mode != TOY\_M))) {

arming\_counter = 0;

return;

}

#if FRAME\_CONFIG == HELI\_FRAME

if ((motors.rsc\_mode > 0) && (g.rc\_8.control\_in >= 10)){

arming\_counter = 0;

return;

}

#endif // HELI\_FRAME

#if TOY\_EDF == ENABLED

int16\_t tmp = g.rc\_1.control\_in;

#else

int16\_t tmp = g.rc\_4.control\_in;

#endif

// full right

if (tmp > 4000) {

// increase the arming counter to a maximum of 1 beyond the auto trim counter

if( arming\_counter <= AUTO\_TRIM\_DELAY ) {

arming\_counter++;

}

// arm the motors and configure for flight

if (arming\_counter == ARM\_DELAY && !motors.armed()) {

////////////////////////////////////////////////////////////////////////////////

// Experimental AP\_Limits library - set constraints, limits, fences, minima, maxima on various parameters

////////////////////////////////////////////////////////////////////////////////

#if AP\_LIMITS == ENABLED

if (limits.enabled() && limits.required()) {

gcs\_send\_text\_P(SEVERITY\_LOW, PSTR("Limits - Running pre-arm checks"));

// check only pre-arm required modules

if (limits.check\_required()) {

gcs\_send\_text\_P(SEVERITY\_LOW, PSTR("ARMING PREVENTED - Limit Breached"));

limits.set\_state(LIMITS\_TRIGGERED);

gcs\_send\_message(MSG\_LIMITS\_STATUS);

arming\_counter++; // restart timer by cycling

}else{

init\_arm\_motors();

}

}else{

init\_arm\_motors();

}

#else // without AP\_LIMITS, just arm motors

init\_arm\_motors();

#endif //AP\_LIMITS\_ENABLED

}

// arm the motors and configure for flight

if (arming\_counter == AUTO\_TRIM\_DELAY && motors.armed()) {

auto\_trim\_counter = 250;

}

// full left

}else if (tmp < -4000) {

// increase the counter to a maximum of 1 beyond the disarm delay

if( arming\_counter <= DISARM\_DELAY ) {

arming\_counter++;

}

// disarm the motors

if (arming\_counter == DISARM\_DELAY && motors.armed()) {

init\_disarm\_motors();

}

// Yaw is centered so reset arming counter

}else{

arming\_counter = 0;

}

}

static void init\_arm\_motors()

{

// arming marker

// Flag used to track if we have armed the motors the first time.

// This is used to decide if we should run the ground\_start routine

// which calibrates the IMU

static bool did\_ground\_start = false;

// disable failsafe because initialising everything takes a while

failsafe\_disable();

//cliSerial->printf("\nARM\n");

#if HIL\_MODE != HIL\_MODE\_DISABLED || defined(DESKTOP\_BUILD)

gcs\_send\_text\_P(SEVERITY\_HIGH, PSTR("ARMING MOTORS"));

#endif

// we don't want writes to the serial port to cause us to pause

// mid-flight, so set the serial ports non-blocking once we arm

// the motors

cliSerial->set\_blocking\_writes(false);

if (gcs3.initialised) {

Serial3.set\_blocking\_writes(false);

}

#if COPTER\_LEDS == ENABLED

if ( bitRead(g.copter\_leds\_mode, 3) ) {

piezo\_beep();

delay(50);

piezo\_beep();

}

#endif

// Remember Orientation

// --------------------

init\_simple\_bearing();

// Reset home position

// -------------------

if(ap.home\_is\_set)

init\_home();

// all I terms are invalid

// -----------------------

reset\_I\_all();

if(did\_ground\_start == false) {

did\_ground\_start = true;

startup\_ground();

}

#if HIL\_MODE != HIL\_MODE\_ATTITUDE

// read Baro pressure at ground -

// this resets Baro for more accuracy

//-----------------------------------

init\_barometer();

#endif

// temp hack

ap\_system.motor\_light = true;

digitalWrite(A\_LED\_PIN, LED\_ON);

// go back to normal AHRS gains

ahrs.set\_fast\_gains(false);

#if SECONDARY\_DMP\_ENABLED == ENABLED

ahrs2.set\_fast\_gains(false);

#endif

// finally actually arm the motors

motors.armed(true);

set\_armed(true);

// reenable failsafe

failsafe\_enable();

}

static void init\_disarm\_motors()

{

#if HIL\_MODE != HIL\_MODE\_DISABLED || defined(DESKTOP\_BUILD)

gcs\_send\_text\_P(SEVERITY\_HIGH, PSTR("DISARMING MOTORS"));

#endif

motors.armed(false);

set\_armed (false);

motors.auto\_armed(false);

set\_auto\_armed(false);

compass.save\_offsets();

g.throttle\_cruise.save();

#if INERTIAL\_NAV\_XY == ENABLED || INERTIAL\_NAV\_Z == ENABLED

inertial\_nav.save\_params();

#endif

// we are not in the air

set\_takeoff\_complete(false);

#if COPTER\_LEDS == ENABLED

if ( bitRead(g.copter\_leds\_mode, 3) ) {

piezo\_beep();

}

#endif

// setup fast AHRS gains to get right attitude

ahrs.set\_fast\_gains(true);

#if SECONDARY\_DMP\_ENABLED == ENABLED

ahrs2.set\_fast\_gains(true);

#endif

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Set the flight control servos based on the current calculated values

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void

set\_servos\_4()

{

#if FRAME\_CONFIG == TRI\_FRAME

// To-Do: implement improved stability patch for tri so that we do not need to limit throttle input to motors

g.rc\_3.servo\_out = min(g.rc\_3.servo\_out, 800);

#endif

motors.output();

}

## navigation

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

// update\_navigation - checks for new GPS updates and invokes navigation routines

static void update\_navigation()

{

static uint32\_t nav\_last\_gps\_update = 0; // the system time of the last gps update

static uint32\_t nav\_last\_gps\_time = 0; // the time according to the gps

bool pos\_updated = false;

bool log\_output = false;

// check for new gps data

if( g\_gps->fix && g\_gps->time != nav\_last\_gps\_time ) {

// used to calculate speed in X and Y, iterms

// ------------------------------------------

dTnav = (float)(millis() - nav\_last\_gps\_update)/ 1000.0;

nav\_last\_gps\_update = millis();

// prevent runup from bad GPS

dTnav = min(dTnav, 1.0);

// save GPS time

nav\_last\_gps\_time = g\_gps->time;

// signal to run nav controllers

pos\_updated = true;

// signal to create log entry

log\_output = true;

}

#if INERTIAL\_NAV\_XY == ENABLED

// TO-DO: clean this up because inertial nav is overwriting the dTnav and pos\_updated from above

// check for inertial nav updates

if( inertial\_nav.position\_ok() ) {

// 50hz

dTnav = 0.02; // To-Do: calculate the time from the mainloop or INS readings?

// signal to run nav controllers

pos\_updated = true;

}

#endif

// calc various navigation values and run controllers if we've received a position update

if( pos\_updated ) {

// calculate velocity

calc\_velocity\_and\_position();

// calculate distance, angles to target

calc\_distance\_and\_bearing();

// run navigation controllers

run\_navigation\_contollers();

// Rotate the nav\_lon and nav\_lat vectors based on Yaw

calc\_nav\_pitch\_roll();

// update log

if (log\_output && (g.log\_bitmask & MASK\_LOG\_NTUN) && motors.armed()) {

Log\_Write\_Nav\_Tuning();

}

}

// reduce nav outputs to zero if we have not received a gps update in 2 seconds

if( millis() - nav\_last\_gps\_update > 2000 ) {

// after 12 reads we guess we may have lost GPS signal, stop navigating

// we have lost GPS signal for a moment. Reduce our error to avoid flyaways

auto\_roll >>= 1;

auto\_pitch >>= 1;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// calc\_velocity\_and\_filtered\_position - velocity in lon and lat directions calculated from GPS position

// and accelerometer data

// lon\_speed expressed in cm/s. positive numbers mean moving east

// lat\_speed expressed in cm/s. positive numbers when moving north

// Note: we use gps locations directly to calculate velocity instead of asking gps for velocity because

// this is more accurate below 1.5m/s

// Note: even though the positions are projected using a lead filter, the velocities are calculated

// from the unaltered gps locations. We do not want noise from our lead filter affecting velocity

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static void calc\_velocity\_and\_position(){

static int32\_t last\_gps\_longitude = 0;

static int32\_t last\_gps\_latitude = 0;

// initialise last\_longitude and last\_latitude

if( last\_gps\_longitude == 0 && last\_gps\_latitude == 0 ) {

last\_gps\_longitude = g\_gps->longitude;

last\_gps\_latitude = g\_gps->latitude;

}

// this speed is ~ in cm because we are using 10^7 numbers from GPS

float tmp = 1.0/dTnav;

#if INERTIAL\_NAV\_XY == ENABLED

if( inertial\_nav.position\_ok() ) {

// pull velocity from interial nav library

lon\_speed = inertial\_nav.get\_longitude\_velocity();

lat\_speed = inertial\_nav.get\_latitude\_velocity();

// pull position from interial nav library

current\_loc.lng = inertial\_nav.get\_longitude();

current\_loc.lat = inertial\_nav.get\_latitude();

}else{

// calculate velocity

lon\_speed = (float)(g\_gps->longitude - last\_gps\_longitude) \* scaleLongDown \* tmp;

lat\_speed = (float)(g\_gps->latitude - last\_gps\_latitude) \* tmp;

// calculate position from gps + expected travel during gps\_lag

current\_loc.lng = xLeadFilter.get\_position(g\_gps->longitude, lon\_speed, g\_gps->get\_lag());

current\_loc.lat = yLeadFilter.get\_position(g\_gps->latitude, lat\_speed, g\_gps->get\_lag());

}

#else

// calculate velocity

lon\_speed = (float)(g\_gps->longitude - last\_gps\_longitude) \* scaleLongDown \* tmp;

lat\_speed = (float)(g\_gps->latitude - last\_gps\_latitude) \* tmp;

// calculate position from gps + expected travel during gps\_lag

current\_loc.lng = xLeadFilter.get\_position(g\_gps->longitude, lon\_speed, g\_gps->get\_lag());

current\_loc.lat = yLeadFilter.get\_position(g\_gps->latitude, lat\_speed, g\_gps->get\_lag());

#endif

// store gps lat and lon values for next iteration

last\_gps\_longitude = g\_gps->longitude;

last\_gps\_latitude = g\_gps->latitude;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Function that will calculate the desired direction to fly and distance

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static void calc\_distance\_and\_bearing()

{

// waypoint distance from plane in cm

// ---------------------------------------

wp\_distance = get\_distance\_cm(&current\_loc, &next\_WP);

home\_distance = get\_distance\_cm(&current\_loc, &home);

// wp\_bearing is bearing to next waypoint

// --------------------------------------------

wp\_bearing = get\_bearing\_cd(&current\_loc, &next\_WP);

home\_bearing = get\_bearing\_cd(&current\_loc, &home);

// bearing to target (used when yaw\_mode = YAW\_LOOK\_AT\_LOCATION)

yaw\_look\_at\_WP\_bearing = get\_bearing\_cd(&current\_loc, &yaw\_look\_at\_WP);

}

static void calc\_location\_error(struct Location \*next\_loc)

{

/\*

\* Becuase we are using lat and lon to do our distance errors here's a quick chart:

\* 100 = 1m

\* 1000 = 11m = 36 feet

\* 1800 = 19.80m = 60 feet

\* 3000 = 33m

\* 10000 = 111m

\*/

// X Error

long\_error = (float)(next\_loc->lng - current\_loc.lng) \* scaleLongDown; // 500 - 0 = 500 Go East

// Y Error

lat\_error = next\_loc->lat - current\_loc.lat; // 500 - 0 = 500 Go North

}

// called after a GPS read

static void run\_navigation\_contollers()

{

// wp\_distance is in CM

// --------------------

switch(control\_mode) {

case AUTO:

// note: wp\_control is handled by commands\_logic

verify\_commands();

// calculates the desired Roll and Pitch

update\_nav\_wp();

break;

case GUIDED:

// switch to loiter once we've reached the target location and altitude

if(verify\_nav\_wp()) {

set\_mode(LOITER);

}

update\_nav\_wp();

break;

case RTL:

// execute the RTL state machine

verify\_RTL();

// calculates the desired Roll and Pitch

update\_nav\_wp();

break;

#if LOITER\_REPOSITIONING == ENABLED // Robert Lefebvre 16/12/2012

// switch passthrough to LOITER

case LOITER:

case POSITION:

// This feature allows us to reposition the quad when the user lets

// go of the sticks

if((abs(g.rc\_2.control\_in) + abs(g.rc\_1.control\_in)) > 100) {

ap.loiter\_override = true;

}

// Allow the user to take control temporarily,

if(ap.loiter\_override){

// reset LOITER to current position

next\_WP.lat += LOITER\_REPOSITION\_RATE \* dTnav \* ((sin\_yaw\_y \* g.rc\_1.control\_in) + (cos\_yaw\_x \* g.rc\_2.control\_in))/4500.0;

next\_WP.lng += LOITER\_REPOSITION\_RATE \* dTnav \* ((sin\_yaw\_y \* g.rc\_2.control\_in) + (cos\_yaw\_x \* g.rc\_1.control\_in))/4500.0;

if((abs(g.rc\_2.control\_in) + abs(g.rc\_1.control\_in)) < 100) {

next\_WP.lat = current\_loc.lat;

next\_WP.lng = current\_loc.lng;

ap.loiter\_override = false;

}

}

wp\_control = LOITER\_MODE;

// calculates the desired Roll and Pitch

update\_nav\_wp();

break;

#else // LOITER\_REPOSITIONING

// switch passthrough to LOITER

case LOITER:

case POSITION:

// This feature allows us to reposition the quad when the user lets

// go of the sticks

if((abs(g.rc\_2.control\_in) + abs(g.rc\_1.control\_in)) > 500) {

if(wp\_distance > 500){

ap.loiter\_override = true;

}

}

// Allow the user to take control temporarily,

if(ap.loiter\_override) {

// this sets the copter to not try and nav while we control it

wp\_control = NO\_NAV\_MODE;

// reset LOITER to current position

next\_WP.lat = current\_loc.lat;

next\_WP.lng = current\_loc.lng;

if(g.rc\_2.control\_in == 0 && g.rc\_1.control\_in == 0) {

wp\_control = LOITER\_MODE;

ap.loiter\_override = false;

}

}else{

wp\_control = LOITER\_MODE;

}

// calculates the desired Roll and Pitch

update\_nav\_wp();

break;

#endif // LOITER\_REPOSITIONING

case LAND:

verify\_land();

// calculates the desired Roll and Pitch

update\_nav\_wp();

break;

case CIRCLE:

wp\_control = CIRCLE\_MODE;

update\_nav\_wp();

break;

case STABILIZE:

case TOY\_A:

case TOY\_M:

wp\_control = NO\_NAV\_MODE;

update\_nav\_wp();

break;

}

// are we in SIMPLE mode?

if(ap.simple\_mode && g.super\_simple) {

// get distance to home

if(home\_distance > SUPER\_SIMPLE\_RADIUS) { // 10m from home

// we reset the angular offset to be a vector from home to the quad

initial\_simple\_bearing = wrap\_360(home\_bearing+18000);

}

}

}

// update\_nav\_wp - high level calculation of nav\_lat and nav\_lon based on wp\_control

// called after gps read from run\_navigation\_controller

static void update\_nav\_wp()

{

int16\_t loiter\_delta;

int16\_t speed;

switch( wp\_control ) {

case LOITER\_MODE:

// calc error to target

calc\_location\_error(&next\_WP);

// use error as the desired rate towards the target

calc\_loiter(long\_error, lat\_error);

break;

case CIRCLE\_MODE:

// check if we have missed the WP

loiter\_delta = (wp\_bearing - old\_wp\_bearing)/100;

// reset the old value

old\_wp\_bearing = wp\_bearing;

// wrap values

if (loiter\_delta > 180) loiter\_delta -= 360;

if (loiter\_delta < -180) loiter\_delta += 360;

// sum the angle around the WP

loiter\_sum += loiter\_delta;

circle\_angle += (circle\_rate \* dTnav);

//1 degree = 0.0174532925 radians

// wrap

if (circle\_angle > 6.28318531)

circle\_angle -= 6.28318531;

next\_WP.lng = circle\_WP.lng + (g.circle\_radius \* 100 \* cos(1.57 - circle\_angle) \* scaleLongUp);

next\_WP.lat = circle\_WP.lat + (g.circle\_radius \* 100 \* sin(1.57 - circle\_angle));

// use error as the desired rate towards the target

// nav\_lon, nav\_lat is calculated

if(wp\_distance > 400) {

calc\_nav\_rate(get\_desired\_speed(g.waypoint\_speed\_max));

}else{

// calc the lat and long error to the target

calc\_location\_error(&next\_WP);

calc\_loiter(long\_error, lat\_error);

}

break;

case WP\_MODE:

// calc error to target

calc\_location\_error(&next\_WP);

speed = get\_desired\_speed(g.waypoint\_speed\_max);

// use error as the desired rate towards the target

calc\_nav\_rate(speed);

break;

case NO\_NAV\_MODE:

// clear out our nav so we can do things like land straight down

// or change Loiter position

// We bring copy over our Iterms for wind control, but we don't navigate

nav\_lon = g.pid\_loiter\_rate\_lon.get\_integrator();

nav\_lat = g.pid\_loiter\_rate\_lon.get\_integrator();

nav\_lon = constrain(nav\_lon, -2000, 2000); // 20 degrees

nav\_lat = constrain(nav\_lat, -2000, 2000); // 20 degrees

break;

}

}

static bool check\_missed\_wp()

{

int32\_t temp;

temp = wp\_bearing - original\_wp\_bearing;

temp = wrap\_180(temp);

return (labs(temp) > 9000); // we passed the waypoint by 100 degrees

}

#define NAV\_ERR\_MAX 600

#define NAV\_RATE\_ERR\_MAX 250

#if LOITER\_REPOSITIONING == ENABLED // Robert Lefebvre 16/12/2012

static void calc\_loiter(int16\_t x\_error, int16\_t y\_error)

{

int32\_t p,i,d; // used to capture pid values for logging

int32\_t output;

int32\_t x\_target\_speed, y\_target\_speed;

// East / West

x\_target\_speed = g.pi\_loiter\_lon.get\_p(x\_error); // calculate desired speed from lon error

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_LOITER\_KP || g.radio\_tuning == CH6\_LOITER\_KI) ) {

Log\_Write\_PID(CH6\_LOITER\_KP, x\_error, x\_target\_speed, 0, 0, x\_target\_speed, tuning\_value);

}

#endif

// calculate rate error

x\_rate\_error = x\_target\_speed - lon\_speed; // calc the speed error

p = g.pid\_loiter\_rate\_lon.get\_p(x\_rate\_error);

i = g.pid\_loiter\_rate\_lon.get\_i(x\_rate\_error, dTnav);

d = g.pid\_loiter\_rate\_lon.get\_d(x\_rate\_error, dTnav);

d = constrain(d, -2000, 2000);

// get rid of noise

if(abs(lon\_speed) < 50) {

d = 0;

}

output = p + i + d;

nav\_lon = constrain(output, -4500, 4500); // constrain max angle to 45 degrees

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_LOITER\_RATE\_KP || g.radio\_tuning == CH6\_LOITER\_RATE\_KI || g.radio\_tuning == CH6\_LOITER\_RATE\_KD) ) {

Log\_Write\_PID(CH6\_LOITER\_RATE\_KP, x\_rate\_error, p, i, d, nav\_lon, tuning\_value);

}

#endif

// North / South

y\_target\_speed = g.pi\_loiter\_lat.get\_p(y\_error); // calculate desired speed from lat error

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_LOITER\_KP || g.radio\_tuning == CH6\_LOITER\_KI) ) {

Log\_Write\_PID(CH6\_LOITER\_KP+100, y\_error, y\_target\_speed, 0, 0, y\_target\_speed, tuning\_value);

}

#endif

// calculate rate error

y\_rate\_error = y\_target\_speed - lat\_speed; // calc the speed error

p = g.pid\_loiter\_rate\_lat.get\_p(y\_rate\_error);

i = g.pid\_loiter\_rate\_lat.get\_i(y\_rate\_error, dTnav);

d = g.pid\_loiter\_rate\_lat.get\_d(y\_rate\_error, dTnav);

d = constrain(d, -2000, 2000);

// get rid of noise

if(abs(lat\_speed) < 50) {

d = 0;

}

output = p + i + d;

nav\_lat = constrain(output, -4500, 4500); // constrain max angle to 45 degrees

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_LOITER\_RATE\_KP || g.radio\_tuning == CH6\_LOITER\_RATE\_KI || g.radio\_tuning == CH6\_LOITER\_RATE\_KD) ) {

Log\_Write\_PID(CH6\_LOITER\_RATE\_KP+100, y\_rate\_error, p, i, d, nav\_lat, tuning\_value);

}

#endif

// copy over I term to Nav\_Rate

g.pid\_nav\_lon.set\_integrator(g.pid\_loiter\_rate\_lon.get\_integrator());

g.pid\_nav\_lat.set\_integrator(g.pid\_loiter\_rate\_lat.get\_integrator());

}

#else // LOITER\_REPOSITIONING

static void calc\_loiter(int16\_t x\_error, int16\_t y\_error)

{

int32\_t p,i,d; // used to capture pid values for logging

int32\_t output;

int32\_t x\_target\_speed, y\_target\_speed;

// East / West

x\_target\_speed = g.pi\_loiter\_lon.get\_p(x\_error); // calculate desired speed from lon error

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_LOITER\_KP || g.radio\_tuning == CH6\_LOITER\_KI) ) {

Log\_Write\_PID(CH6\_LOITER\_KP, x\_error, x\_target\_speed, 0, 0, x\_target\_speed, tuning\_value);

}

#endif

// calculate rate error

x\_rate\_error = x\_target\_speed - lon\_speed; // calc the speed error

p = g.pid\_loiter\_rate\_lon.get\_p(x\_rate\_error);

i = g.pid\_loiter\_rate\_lon.get\_i(x\_rate\_error + x\_error, dTnav);

d = g.pid\_loiter\_rate\_lon.get\_d(x\_error, dTnav);

d = constrain(d, -2000, 2000);

// get rid of noise

if(abs(lon\_speed) < 50) {

d = 0;

}

output = p + i + d;

nav\_lon = constrain(output, -4500, 4500); // constrain max angle to 45 degrees

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_LOITER\_RATE\_KP || g.radio\_tuning == CH6\_LOITER\_RATE\_KI || g.radio\_tuning == CH6\_LOITER\_RATE\_KD) ) {

Log\_Write\_PID(CH6\_LOITER\_RATE\_KP, x\_rate\_error, p, i, d, nav\_lon, tuning\_value);

}

#endif

// North / South

y\_target\_speed = g.pi\_loiter\_lat.get\_p(y\_error); // calculate desired speed from lat error

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_LOITER\_KP || g.radio\_tuning == CH6\_LOITER\_KI) ) {

Log\_Write\_PID(CH6\_LOITER\_KP+100, y\_error, y\_target\_speed, 0, 0, y\_target\_speed, tuning\_value);

}

#endif

// calculate rate error

y\_rate\_error = y\_target\_speed - lat\_speed; // calc the speed error

p = g.pid\_loiter\_rate\_lat.get\_p(y\_rate\_error);

i = g.pid\_loiter\_rate\_lat.get\_i(y\_rate\_error + y\_error, dTnav);

d = g.pid\_loiter\_rate\_lat.get\_d(y\_error, dTnav);

d = constrain(d, -2000, 2000);

// get rid of noise

if(abs(lat\_speed) < 50) {

d = 0;

}

output = p + i + d;

nav\_lat = constrain(output, -4500, 4500); // constrain max angle to 45 degrees

#if LOGGING\_ENABLED == ENABLED

// log output if PID logging is on and we are tuning the yaw

if( g.log\_bitmask & MASK\_LOG\_PID && (g.radio\_tuning == CH6\_LOITER\_RATE\_KP || g.radio\_tuning == CH6\_LOITER\_RATE\_KI || g.radio\_tuning == CH6\_LOITER\_RATE\_KD) ) {

Log\_Write\_PID(CH6\_LOITER\_RATE\_KP+100, y\_rate\_error, p, i, d, nav\_lat, tuning\_value);

}

#endif

// copy over I term to Nav\_Rate

g.pid\_nav\_lon.set\_integrator(g.pid\_loiter\_rate\_lon.get\_integrator());

g.pid\_nav\_lat.set\_integrator(g.pid\_loiter\_rate\_lat.get\_integrator());

}

#endif // LOITER\_REPOSITIONING

static void calc\_nav\_rate(int16\_t max\_speed)

{

float temp, temp\_x, temp\_y;

// push us towards the original track

update\_crosstrack();

int16\_t cross\_speed = crosstrack\_error \* -g.crosstrack\_gain; // scale down crosstrack\_error in cm

cross\_speed = constrain(cross\_speed, -150, 150);

// rotate by 90 to deal with trig functions

temp = (9000l - wp\_bearing) \* RADX100;

temp\_x = cos(temp);

temp\_y = sin(temp);

// rotate desired spped vector:

int32\_t x\_target\_speed = max\_speed \* temp\_x - cross\_speed \* temp\_y;

int32\_t y\_target\_speed = cross\_speed \* temp\_x + max\_speed \* temp\_y;

// East / West

// calculate rate error

x\_rate\_error = x\_target\_speed - lon\_speed;

x\_rate\_error = constrain(x\_rate\_error, -500, 500);

nav\_lon = g.pid\_nav\_lon.get\_pid(x\_rate\_error, dTnav);

int32\_t tilt = (x\_target\_speed \* x\_target\_speed \* (int32\_t)g.tilt\_comp) / 10000;

if(x\_target\_speed < 0) tilt = -tilt;

nav\_lon += tilt;

// North / South

// calculate rate error

y\_rate\_error = y\_target\_speed - lat\_speed;

y\_rate\_error = constrain(y\_rate\_error, -500, 500); // added a rate error limit to keep pitching down to a minimum

nav\_lat = g.pid\_nav\_lat.get\_pid(y\_rate\_error, dTnav);

tilt = (y\_target\_speed \* y\_target\_speed \* (int32\_t)g.tilt\_comp) / 10000;

if(y\_target\_speed < 0) tilt = -tilt;

nav\_lat += tilt;

// copy over I term to Loiter\_Rate

g.pid\_loiter\_rate\_lon.set\_integrator(g.pid\_nav\_lon.get\_integrator());

g.pid\_loiter\_rate\_lat.set\_integrator(g.pid\_nav\_lat.get\_integrator());

}

// this calculation rotates our World frame of reference to the copter's frame of reference

// We use the DCM's matrix to precalculate these trig values at 50hz

static void calc\_nav\_pitch\_roll()

{

// rotate the vector

auto\_roll = (float)nav\_lon \* sin\_yaw\_y - (float)nav\_lat \* cos\_yaw\_x;

auto\_pitch = (float)nav\_lon \* cos\_yaw\_x + (float)nav\_lat \* sin\_yaw\_y;

// flip pitch because forward is negative

auto\_pitch = -auto\_pitch;

// constrain maximum roll and pitch angles to 45 degrees

auto\_roll = constrain(auto\_roll, -4500, 4500);

auto\_pitch = constrain(auto\_pitch, -4500, 4500);

}

static int16\_t get\_desired\_speed(int16\_t max\_speed)

{

/\*

Based on Equation by Bill Premerlani & Robert Lefebvre

(sq(V2)-sq(V1))/2 = A(X2-X1)

derives to:

V1 = sqrt(sq(V2) - 2\*A\*(X2-X1))

\*/

if(ap.fast\_corner) {

// don't slow down

}else{

if(wp\_distance < 20000){ // limit the size of numbers we're dealing with to avoid overflow

// go slower

int32\_t temp = 2 \* 100 \* (int32\_t)(wp\_distance - g.waypoint\_radius \* 100);

int32\_t s\_min = WAYPOINT\_SPEED\_MIN;

temp += s\_min \* s\_min;

if( temp < 0 ) temp = 0; // check to ensure we don't try to take the sqrt of a negative number

max\_speed = sqrt((float)temp);

max\_speed = min(max\_speed, g.waypoint\_speed\_max);

}

}

max\_speed = min(max\_speed, max\_speed\_old + (100 \* dTnav));// limit going faster

max\_speed = max(max\_speed, WAYPOINT\_SPEED\_MIN); // don't go too slow

max\_speed\_old = max\_speed;

return max\_speed;

}

static void reset\_desired\_speed()

{

max\_speed\_old = 0;

}

static void update\_crosstrack(void)

{

// Crosstrack Error

// ----------------

if (wp\_distance >= (g.crosstrack\_min\_distance \* 100) &&

abs(wrap\_180(wp\_bearing - original\_wp\_bearing)) < 4500) {

float temp = (wp\_bearing - original\_wp\_bearing) \* RADX100;

crosstrack\_error = sin(temp) \* wp\_distance; // Meters we are off track line

}else{

// fade out crosstrack

crosstrack\_error >>= 1;

}

}

static void force\_new\_altitude(int32\_t new\_alt)

{

next\_WP.alt = new\_alt;

set\_alt\_change(REACHED\_ALT);

}

static void set\_new\_altitude(int32\_t new\_alt)

{

// if no change exit immediately

if(new\_alt == next\_WP.alt) {

return;

}

// update new target altitude

next\_WP.alt = new\_alt;

if(next\_WP.alt > (current\_loc.alt + 80)) {

// we are below, going up

set\_alt\_change(ASCENDING);

}else if(next\_WP.alt < (current\_loc.alt - 80)) {

// we are above, going down

set\_alt\_change(DESCENDING);

}else{

// No Change

set\_alt\_change(REACHED\_ALT);

}

}

static void verify\_altitude()

{

if(alt\_change\_flag == ASCENDING) {

// we are below, going up

if(current\_loc.alt > next\_WP.alt - 50) {

set\_alt\_change(REACHED\_ALT);

}

}else if (alt\_change\_flag == DESCENDING) {

// we are above, going down

if(current\_loc.alt <= next\_WP.alt + 50){

set\_alt\_change(REACHED\_ALT);

}

}

}

// Keeps old data out of our calculation / logs

static void reset\_nav\_params(void)

{

// always start Circle mode at same angle

circle\_angle = 0;

// We must be heading to a new WP, so XTrack must be 0

crosstrack\_error = 0;

// Will be set by new command

wp\_bearing = 0;

// Will be set by new command

wp\_distance = 0;

// Will be set by new command, used by loiter

long\_error = 0;

lat\_error = 0;

nav\_lon = 0;

nav\_lat = 0;

nav\_roll = 0;

nav\_pitch = 0;

auto\_roll = 0;

auto\_pitch = 0;

}

static int32\_t wrap\_360(int32\_t error)

{

if (error > 36000) error -= 36000;

if (error < 0) error += 36000;

return error;

}

static int32\_t wrap\_180(int32\_t error)

{

if (error > 18000) error -= 36000;

if (error < -18000) error += 36000;

return error;

}

// get\_yaw\_slew - reduces rate of change of yaw to a maximum

// assumes it is called at 100hz so centi-degrees and update rate cancel each other out

static int32\_t get\_yaw\_slew(int32\_t current\_yaw, int32\_t desired\_yaw, int16\_t deg\_per\_sec)

{

return wrap\_360(current\_yaw + constrain(wrap\_180(desired\_yaw - current\_yaw), -deg\_per\_sec, deg\_per\_sec));

}

## perf\_info

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

//

// high level performance monitoring

//

// we measure the main loop time

//

#define PERF\_INFO\_OVERTIME\_THRESHOLD\_MICROS 10500

uint16\_t perf\_info\_loop\_count;

uint32\_t perf\_info\_max\_time;

uint16\_t perf\_info\_long\_running;

// perf\_info\_reset - reset all records of loop time to zero

void perf\_info\_reset()

{

perf\_info\_loop\_count = 0;

perf\_info\_max\_time = 0;

perf\_info\_long\_running = 0;

}

// perf\_info\_check\_loop\_time - check latest loop time vs min, max and overtime threshold

void perf\_info\_check\_loop\_time(uint32\_t time\_in\_micros)

{

perf\_info\_loop\_count++;

if( time\_in\_micros > perf\_info\_max\_time) {

perf\_info\_max\_time = time\_in\_micros;

}

if( time\_in\_micros > PERF\_INFO\_OVERTIME\_THRESHOLD\_MICROS ) {

perf\_info\_long\_running++;

}

}

// perf\_info\_get\_long\_running\_percentage - get number of long running loops as a percentage of the total number of loops

uint16\_t perf\_info\_get\_num\_loops()

{

return perf\_info\_loop\_count;

}

// perf\_info\_get\_max\_time - return maximum loop time (in microseconds)

uint32\_t perf\_info\_get\_max\_time()

{

return perf\_info\_max\_time;

}

// perf\_info\_get\_num\_long\_running - get number of long running loops

uint16\_t perf\_info\_get\_num\_long\_running()

{

return perf\_info\_long\_running;

}

## radio

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

// Function that will read the radio data, limit servos and trigger a failsafe

// ----------------------------------------------------------------------------

extern RC\_Channel\* rc\_ch[NUM\_CHANNELS];

static void default\_dead\_zones()

{

g.rc\_1.set\_dead\_zone(60);

g.rc\_2.set\_dead\_zone(60);

#if FRAME\_CONFIG == HELI\_FRAME

g.rc\_3.set\_dead\_zone(20);

g.rc\_4.set\_dead\_zone(30);

#else

g.rc\_3.set\_dead\_zone(60);

g.rc\_4.set\_dead\_zone(80);

#endif

}

static void init\_rc\_in()

{

// set rc channel ranges

g.rc\_1.set\_angle(MAX\_INPUT\_ROLL\_ANGLE);

g.rc\_2.set\_angle(MAX\_INPUT\_PITCH\_ANGLE);

#if FRAME\_CONFIG == HELI\_FRAME

// we do not want to limit the movment of the heli's swash plate

g.rc\_3.set\_range(0, 1000);

#else

g.rc\_3.set\_range(g.throttle\_min, g.throttle\_max);

#endif

g.rc\_4.set\_angle(4500);

// reverse: CW = left

// normal: CW = left???

g.rc\_1.set\_type(RC\_CHANNEL\_TYPE\_ANGLE\_RAW);

g.rc\_2.set\_type(RC\_CHANNEL\_TYPE\_ANGLE\_RAW);

g.rc\_4.set\_type(RC\_CHANNEL\_TYPE\_ANGLE\_RAW);

rc\_ch[CH\_1] = &g.rc\_1;

rc\_ch[CH\_2] = &g.rc\_2;

rc\_ch[CH\_3] = &g.rc\_3;

rc\_ch[CH\_4] = &g.rc\_4;

rc\_ch[CH\_5] = &g.rc\_5;

rc\_ch[CH\_6] = &g.rc\_6;

rc\_ch[CH\_7] = &g.rc\_7;

rc\_ch[CH\_8] = &g.rc\_8;

//set auxiliary ranges

g.rc\_5.set\_range(0,1000);

g.rc\_6.set\_range(0,1000);

g.rc\_7.set\_range(0,1000);

g.rc\_8.set\_range(0,1000);

#if MOUNT == ENABLED

update\_aux\_servo\_function(&g.rc\_5, &g.rc\_6, &g.rc\_7, &g.rc\_8, &g.rc\_10, &g.rc\_11);

#endif

}

static void init\_rc\_out()

{

APM\_RC.Init( &isr\_registry ); // APM Radio initialization

motors.set\_update\_rate(g.rc\_speed);

motors.set\_frame\_orientation(g.frame\_orientation);

motors.Init(); // motor initialisation

motors.set\_min\_throttle(g.throttle\_min);

motors.set\_max\_throttle(g.throttle\_max);

for(byte i = 0; i < 5; i++) {

delay(20);

read\_radio();

}

// we want the input to be scaled correctly

g.rc\_3.set\_range\_out(0,1000);

// sanity check - prevent unconfigured radios from outputting

if(g.rc\_3.radio\_min >= 1300) {

g.rc\_3.radio\_min = g.rc\_3.radio\_in;

}

// we are full throttle

if(g.rc\_3.control\_in >= (MAXIMUM\_THROTTLE - 50)) {

if(g.esc\_calibrate == 0) {

// we will enter esc\_calibrate mode on next reboot

g.esc\_calibrate.set\_and\_save(1);

// send minimum throttle out to ESC

motors.output\_min();

// display message on console

cliSerial->printf\_P(PSTR("Entering ESC Calibration: please restart APM.\n"));

// block until we restart

while(1) {

delay(200);

dancing\_light();

}

}else{

cliSerial->printf\_P(PSTR("ESC Calibration active: passing throttle through to ESCs.\n"));

// clear esc flag

g.esc\_calibrate.set\_and\_save(0);

// pass through user throttle to escs

init\_esc();

}

}else{

// did we abort the calibration?

if(g.esc\_calibrate == 1)

g.esc\_calibrate.set\_and\_save(0);

// send miinimum throttle out to ESC

output\_min();

}

#if TOY\_EDF == ENABLED

// add access to CH8 and CH6

APM\_RC.enable\_out(CH\_8);

APM\_RC.enable\_out(CH\_6);

#endif

}

void output\_min()

{

// enable motors

motors.enable();

motors.output\_min();

}

#define RADIO\_FS\_TIMEOUT\_MS 2000 // 2 seconds

static void read\_radio()

{

if (APM\_RC.GetState() == 1) {

ap\_system.new\_radio\_frame = true;

g.rc\_1.set\_pwm(APM\_RC.InputCh(CH\_1));

g.rc\_2.set\_pwm(APM\_RC.InputCh(CH\_2));

set\_throttle\_and\_failsafe(APM\_RC.InputCh(CH\_3));

g.rc\_4.set\_pwm(APM\_RC.InputCh(CH\_4));

g.rc\_5.set\_pwm(APM\_RC.InputCh(CH\_5));

g.rc\_6.set\_pwm(APM\_RC.InputCh(CH\_6));

g.rc\_7.set\_pwm(APM\_RC.InputCh(CH\_7));

g.rc\_8.set\_pwm(APM\_RC.InputCh(CH\_8));

#if FRAME\_CONFIG != HELI\_FRAME

// limit our input to 800 so we can still pitch and roll

g.rc\_3.control\_in = min(g.rc\_3.control\_in, MAXIMUM\_THROTTLE);

#endif

}else{

// turn on throttle failsafe if no update from ppm encoder for 2 seconds

uint32\_t last\_rc\_update = APM\_RC.get\_last\_update();

if ((millis() - last\_rc\_update >= RADIO\_FS\_TIMEOUT\_MS) && g.failsafe\_throttle && motors.armed() && !ap.failsafe) {

Log\_Write\_Error(ERROR\_SUBSYSTEM\_RADIO, ERROR\_CODE\_RADIO\_LATE\_FRAME);

set\_failsafe(true);

}

}

}

#define FS\_COUNTER 3

static void set\_throttle\_and\_failsafe(uint16\_t throttle\_pwm)

{

static int8\_t failsafe\_counter = 0;

// if failsafe not enabled pass through throttle and exit

if(g.failsafe\_throttle == FS\_THR\_DISABLED) {

g.rc\_3.set\_pwm(throttle\_pwm);

return;

}

//check for low throttle value

if (throttle\_pwm < (uint16\_t)g.failsafe\_throttle\_value) {

// if we are already in failsafe or motors not armed pass through throttle and exit

if (ap.failsafe || !motors.armed()) {

g.rc\_3.set\_pwm(throttle\_pwm);

return;

}

// check for 3 low throttle values

// Note: we do not pass through the low throttle until 3 low throttle values are recieved

failsafe\_counter++;

if( failsafe\_counter >= FS\_COUNTER ) {

failsafe\_counter = FS\_COUNTER; // check to ensure we don't overflow the counter

set\_failsafe(true);

g.rc\_3.set\_pwm(throttle\_pwm); // pass through failsafe throttle

}

}else{

// we have a good throttle so reduce failsafe counter

failsafe\_counter--;

if( failsafe\_counter <= 0 ) {

failsafe\_counter = 0; // check to ensure we don't underflow the counter

// disengage failsafe after three (nearly) consecutive valid throttle values

if (ap.failsafe) {

set\_failsafe(false);

}

}

// pass through throttle

g.rc\_3.set\_pwm(throttle\_pwm);

}

}

static void trim\_radio()

{

for (byte i = 0; i < 30; i++) {

read\_radio();

}

g.rc\_1.trim(); // roll

g.rc\_2.trim(); // pitch

g.rc\_4.trim(); // yaw

g.rc\_1.save\_eeprom();

g.rc\_2.save\_eeprom();

g.rc\_4.save\_eeprom();

}

## sensors

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

// Sensors are not available in HIL\_MODE\_ATTITUDE

#if HIL\_MODE != HIL\_MODE\_ATTITUDE

static void ReadSCP1000(void) {

}

#if CONFIG\_SONAR == ENABLED

static void init\_sonar(void)

{

#if CONFIG\_SONAR\_SOURCE == SONAR\_SOURCE\_ADC

sonar.calculate\_scaler(g.sonar\_type, 3.3);

#else

sonar.calculate\_scaler(g.sonar\_type, 5.0);

#endif

}

#endif

static void init\_barometer(void)

{

barometer.calibrate(mavlink\_delay);

ahrs.set\_barometer(&barometer);

gcs\_send\_text\_P(SEVERITY\_LOW, PSTR("barometer calibration complete"));

}

// return barometric altitude in centimeters

static int32\_t read\_barometer(void)

{

barometer.read();

return baro\_filter.apply(barometer.get\_altitude() \* 100.0);

}

// return sonar altitude in centimeters

static int16\_t read\_sonar(void)

{

#if CONFIG\_SONAR == ENABLED

// exit immediately if sonar is disabled

if( !g.sonar\_enabled ) {

sonar\_alt\_health = 0;

return 0;

}

int16\_t temp\_alt = sonar.read();

if(temp\_alt >= sonar.min\_distance && temp\_alt <= sonar.max\_distance \* 0.70) {

if( sonar\_alt\_health < SONAR\_ALT\_HEALTH\_MAX ) {

sonar\_alt\_health++;

}

}else{

sonar\_alt\_health = 0;

}

#if SONAR\_TILT\_CORRECTION == 1

// correct alt for angle of the sonar

float temp = cos\_pitch\_x \* cos\_roll\_x;

temp = max(temp, 0.707);

temp\_alt = (float)temp\_alt \* temp;

#endif

return temp\_alt;

#else

return 0;

#endif

}

#endif // HIL\_MODE != HIL\_MODE\_ATTITUDE

static void init\_compass()

{

compass.set\_orientation(MAG\_ORIENTATION); // set compass's orientation on aircraft

if (!compass.init() || !compass.read()) {

// make sure we don't pass a broken compass to DCM

cliSerial->println\_P(PSTR("COMPASS INIT ERROR"));

Log\_Write\_Error(ERROR\_SUBSYSTEM\_COMPASS,ERROR\_CODE\_FAILED\_TO\_INITIALISE);

return;

}

ahrs.set\_compass(&compass);

#if SECONDARY\_DMP\_ENABLED == ENABLED

ahrs2.set\_compass(&compass);

#endif

}

static void init\_optflow()

{

#if OPTFLOW == ENABLED

if( optflow.init(false, &timer\_scheduler, &spi\_semaphore, &spi3\_semaphore) == false ) {

g.optflow\_enabled = false;

cliSerial->print\_P(PSTR("\nFailed to Init OptFlow "));

Log\_Write\_Error(ERROR\_SUBSYSTEM\_OPTFLOW,ERROR\_CODE\_FAILED\_TO\_INITIALISE);

}else{

// suspend timer while we set-up SPI communication

timer\_scheduler.suspend\_timer();

optflow.set\_orientation(OPTFLOW\_ORIENTATION); // set optical flow sensor's orientation on aircraft

optflow.set\_frame\_rate(2000); // set minimum update rate (which should lead to maximum low light performance

optflow.set\_resolution(OPTFLOW\_RESOLUTION); // set optical flow sensor's resolution

optflow.set\_field\_of\_view(OPTFLOW\_FOV); // set optical flow sensor's field of view

// resume timer

timer\_scheduler.resume\_timer();

}

#endif // OPTFLOW == ENABLED

}

// read\_battery - check battery voltage and current and invoke failsafe if necessary

// called at 10hz

#define BATTERY\_FS\_COUNTER 100 // 100 iterations at 10hz is 10 seconds

static void read\_battery(void)

{

static uint8\_t low\_battery\_counter = 0;

if(g.battery\_monitoring == 0) {

battery\_voltage1 = 0;

return;

}

if(g.battery\_monitoring == 3 || g.battery\_monitoring == 4) {

static AP\_AnalogSource\_Arduino batt\_volt\_pin(g.battery\_volt\_pin);

batt\_volt\_pin.set\_pin(g.battery\_volt\_pin);

battery\_voltage1 = BATTERY\_VOLTAGE(batt\_volt\_pin.read\_average());

}

if(g.battery\_monitoring == 4) {

static AP\_AnalogSource\_Arduino batt\_curr\_pin(g.battery\_curr\_pin);

batt\_curr\_pin.set\_pin(g.battery\_curr\_pin);

current\_amps1 = CURRENT\_AMPS(batt\_curr\_pin.read\_average());

current\_total1 += current\_amps1 \* 0.02778; // called at 100ms on average, .0002778 is 1/3600 (conversion to hours)

}

// check for low voltage or current if the low voltage check hasn't already been triggered

if (!ap.low\_battery && ( battery\_voltage1 < g.low\_voltage || (g.battery\_monitoring == 4 && current\_total1 > g.pack\_capacity))) {

low\_battery\_counter++;

if( low\_battery\_counter >= BATTERY\_FS\_COUNTER ) {

low\_battery\_counter = BATTERY\_FS\_COUNTER; // ensure counter does not overflow

low\_battery\_event();

}

}else{

// reset low\_battery\_counter in case it was a temporary voltage dip

low\_battery\_counter = 0;

}

}

// read the receiver RSSI as an 8 bit number for MAVLink

// RC\_CHANNELS\_SCALED message

void read\_receiver\_rssi(void)

{

RSSI\_pin.set\_pin(g.rssi\_pin);

float ret = RSSI\_pin.read();

receiver\_rssi = constrain(ret, 0, 255);

}

## setup

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

#if CLI\_ENABLED == ENABLED

// Functions called from the setup menu

static int8\_t setup\_radio (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_motors (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_accel (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_accel\_scale (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_frame (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_factory (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_erase (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_flightmodes (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_batt\_monitor (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_sonar (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_compass (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_tune (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_range (uint8\_t argc, const Menu::arg \*argv);

//static int8\_t setup\_mag\_offset (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_declination (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_optflow (uint8\_t argc, const Menu::arg \*argv);

#if FRAME\_CONFIG == HELI\_FRAME

static int8\_t setup\_heli (uint8\_t argc, const Menu::arg \*argv);

static int8\_t setup\_gyro (uint8\_t argc, const Menu::arg \*argv);

#endif

// Command/function table for the setup menu

const struct Menu::command setup\_menu\_commands[] PROGMEM = {

// command function called

// ======= ===============

{"erase", setup\_erase},

{"reset", setup\_factory},

{"radio", setup\_radio},

{"frame", setup\_frame},

{"motors", setup\_motors},

{"level", setup\_accel},

{"accel", setup\_accel\_scale},

{"modes", setup\_flightmodes},

{"battery", setup\_batt\_monitor},

{"sonar", setup\_sonar},

{"compass", setup\_compass},

{"tune", setup\_tune},

{"range", setup\_range},

// {"offsets", setup\_mag\_offset},

{"declination", setup\_declination},

{"optflow", setup\_optflow},

#if FRAME\_CONFIG == HELI\_FRAME

{"heli", setup\_heli},

{"gyro", setup\_gyro},

#endif

{"show", setup\_show}

};

// Create the setup menu object.

MENU(setup\_menu, "setup", setup\_menu\_commands);

// Called from the top-level menu to run the setup menu.

static int8\_t

setup\_mode(uint8\_t argc, const Menu::arg \*argv)

{

// Give the user some guidance

cliSerial->printf\_P(PSTR("Setup Mode\n\n\n"));

//"\n"

//"IMPORTANT: if you have not previously set this system up, use the\n"

//"'reset' command to initialize the EEPROM to sensible default values\n"

//"and then the 'radio' command to configure for your radio.\n"

//"\n"));

if(g.rc\_1.radio\_min >= 1300) {

delay(1000);

cliSerial->printf\_P(PSTR("\n!Warning, radio not configured!"));

delay(1000);

cliSerial->printf\_P(PSTR("\n Type 'radio' now.\n\n"));

}

// Run the setup menu. When the menu exits, we will return to the main menu.

setup\_menu.run();

return 0;

}

// Print the current configuration.

// Called by the setup menu 'show' command.

static int8\_t

setup\_show(uint8\_t argc, const Menu::arg \*argv)

{

// clear the area

print\_blanks(8);

report\_version();

report\_radio();

report\_frame();

report\_batt\_monitor();

report\_sonar();

//report\_gains();

//report\_xtrack();

//report\_throttle();

report\_flight\_modes();

report\_ins();

report\_compass();

report\_optflow();

#if FRAME\_CONFIG == HELI\_FRAME

report\_heli();

report\_gyro();

#endif

AP\_Param::show\_all();

return(0);

}

// Initialise the EEPROM to 'factory' settings (mostly defined in APM\_Config.h or via defaults).

// Called by the setup menu 'factoryreset' command.

static int8\_t

setup\_factory(uint8\_t argc, const Menu::arg \*argv)

{

int16\_t c;

cliSerial->printf\_P(PSTR("\n'Y' = factory reset, any other key to abort:\n"));

do {

c = cliSerial->read();

} while (-1 == c);

if (('y' != c) && ('Y' != c))

return(-1);

AP\_Param::erase\_all();

cliSerial->printf\_P(PSTR("\nReboot APM"));

delay(1000);

//default\_gains();

for (;; ) {

}

// note, cannot actually return here

return(0);

}

// Perform radio setup.

// Called by the setup menu 'radio' command.

static int8\_t

setup\_radio(uint8\_t argc, const Menu::arg \*argv)

{

cliSerial->println\_P(PSTR("\n\nRadio Setup:"));

uint8\_t i;

for(i = 0; i < 100; i++) {

delay(20);

read\_radio();

}

if(g.rc\_1.radio\_in < 500) {

while(1) {

//cliSerial->printf\_P(PSTR("\nNo radio; Check connectors."));

delay(1000);

// stop here

}

}

g.rc\_1.radio\_min = g.rc\_1.radio\_in;

g.rc\_2.radio\_min = g.rc\_2.radio\_in;

g.rc\_3.radio\_min = g.rc\_3.radio\_in;

g.rc\_4.radio\_min = g.rc\_4.radio\_in;

g.rc\_5.radio\_min = g.rc\_5.radio\_in;

g.rc\_6.radio\_min = g.rc\_6.radio\_in;

g.rc\_7.radio\_min = g.rc\_7.radio\_in;

g.rc\_8.radio\_min = g.rc\_8.radio\_in;

g.rc\_1.radio\_max = g.rc\_1.radio\_in;

g.rc\_2.radio\_max = g.rc\_2.radio\_in;

g.rc\_3.radio\_max = g.rc\_3.radio\_in;

g.rc\_4.radio\_max = g.rc\_4.radio\_in;

g.rc\_5.radio\_max = g.rc\_5.radio\_in;

g.rc\_6.radio\_max = g.rc\_6.radio\_in;

g.rc\_7.radio\_max = g.rc\_7.radio\_in;

g.rc\_8.radio\_max = g.rc\_8.radio\_in;

g.rc\_1.radio\_trim = g.rc\_1.radio\_in;

g.rc\_2.radio\_trim = g.rc\_2.radio\_in;

g.rc\_4.radio\_trim = g.rc\_4.radio\_in;

// 3 is not trimed

g.rc\_5.radio\_trim = 1500;

g.rc\_6.radio\_trim = 1500;

g.rc\_7.radio\_trim = 1500;

g.rc\_8.radio\_trim = 1500;

cliSerial->printf\_P(PSTR("\nMove all controls to extremes. Enter to save: "));

while(1) {

delay(20);

// Filters radio input - adjust filters in the radio.pde file

// ----------------------------------------------------------

read\_radio();

g.rc\_1.update\_min\_max();

g.rc\_2.update\_min\_max();

g.rc\_3.update\_min\_max();

g.rc\_4.update\_min\_max();

g.rc\_5.update\_min\_max();

g.rc\_6.update\_min\_max();

g.rc\_7.update\_min\_max();

g.rc\_8.update\_min\_max();

if(cliSerial->available() > 0) {

delay(20);

cliSerial->flush();

g.rc\_1.save\_eeprom();

g.rc\_2.save\_eeprom();

g.rc\_3.save\_eeprom();

g.rc\_4.save\_eeprom();

g.rc\_5.save\_eeprom();

g.rc\_6.save\_eeprom();

g.rc\_7.save\_eeprom();

g.rc\_8.save\_eeprom();

print\_done();

break;

}

}

report\_radio();

return(0);

}

static int8\_t

setup\_motors(uint8\_t argc, const Menu::arg \*argv)

{

cliSerial->printf\_P(PSTR(

"Now connect the main lipo and follow the instruction on the wiki for your frame setup.\n"

"For security remember to disconnect the main lipo after the test, then hit any key to exit.\n"

"Any key to exit.\n"));

while(1) {

delay(20);

read\_radio();

motors.output\_test();

if(cliSerial->available() > 0) {

g.esc\_calibrate.set\_and\_save(0);

return(0);

}

}

}

static int8\_t

setup\_accel(uint8\_t argc, const Menu::arg \*argv)

{

ins.init(AP\_InertialSensor::COLD\_START,

ins\_sample\_rate,

delay, flash\_leds, &timer\_scheduler);

ins.init\_accel(delay, flash\_leds); // level accelerometer values

ahrs.set\_trim(Vector3f(0,0,0)); // clear out saved trim

report\_ins();

return(0);

}

/\*

handle full accelerometer calibration via user dialog

\*/

static void setup\_printf\_P(const prog\_char\_t \*fmt, ...)

{

va\_list arg\_list;

va\_start(arg\_list, fmt);

cliSerial->vprintf\_P(fmt, arg\_list);

va\_end(arg\_list);

}

static void setup\_wait\_key(void)

{

// wait for user input

while (!cliSerial->available()) {

delay(20);

}

// clear input buffer

while( cliSerial->available() ) {

cliSerial->read();

}

}

static int8\_t

setup\_accel\_scale(uint8\_t argc, const Menu::arg \*argv)

{

float trim\_roll, trim\_pitch;

cliSerial->println\_P(PSTR("Initialising gyros"));

ins.init(AP\_InertialSensor::COLD\_START,

ins\_sample\_rate,

delay, flash\_leds, &timer\_scheduler);

ins.calibrate\_accel(delay, flash\_leds, setup\_printf\_P, setup\_wait\_key, trim\_roll, trim\_pitch);

// reset ahrs's trim to suggested values from calibration routine

trim\_roll = constrain(trim\_roll, ToRad(-10.0f), ToRad(10.0f));

trim\_pitch = constrain(trim\_pitch, ToRad(-10.0f), ToRad(10.0f));

ahrs.set\_trim(Vector3f(trim\_roll, trim\_pitch, 0));

report\_ins();

return(0);

}

static int8\_t

setup\_frame(uint8\_t argc, const Menu::arg \*argv)

{

if (!strcmp\_P(argv[1].str, PSTR("x"))) {

g.frame\_orientation.set\_and\_save(X\_FRAME);

} else if (!strcmp\_P(argv[1].str, PSTR("p"))) {

g.frame\_orientation.set\_and\_save(PLUS\_FRAME);

} else if (!strcmp\_P(argv[1].str, PSTR("+"))) {

g.frame\_orientation.set\_and\_save(PLUS\_FRAME);

} else if (!strcmp\_P(argv[1].str, PSTR("v"))) {

g.frame\_orientation.set\_and\_save(V\_FRAME);

}else{

cliSerial->printf\_P(PSTR("\nOp:[x,+,v]\n"));

report\_frame();

return 0;

}

report\_frame();

return 0;

}

static int8\_t

setup\_flightmodes(uint8\_t argc, const Menu::arg \*argv)

{

byte \_switchPosition = 0;

byte \_oldSwitchPosition = 0;

int8\_t mode = 0;

cliSerial->printf\_P(PSTR("\nMode switch to edit, aileron: select modes, rudder: Simple on/off\n"));

print\_hit\_enter();

while(1) {

delay(20);

read\_radio();

\_switchPosition = readSwitch();

// look for control switch change

if (\_oldSwitchPosition != \_switchPosition) {

mode = flight\_modes[\_switchPosition];

mode = constrain(mode, 0, NUM\_MODES-1);

// update the user

print\_switch(\_switchPosition, mode, (g.simple\_modes & (1<<\_switchPosition)));

// Remember switch position

\_oldSwitchPosition = \_switchPosition;

}

// look for stick input

if (abs(g.rc\_1.control\_in) > 3000) {

mode++;

if(mode >= NUM\_MODES)

mode = 0;

// save new mode

flight\_modes[\_switchPosition] = mode;

// print new mode

print\_switch(\_switchPosition, mode, (g.simple\_modes & (1<<\_switchPosition)));

delay(500);

}

// look for stick input

if (g.rc\_4.control\_in > 3000) {

g.simple\_modes |= (1<<\_switchPosition);

// print new mode

print\_switch(\_switchPosition, mode, (g.simple\_modes & (1<<\_switchPosition)));

delay(500);

}

// look for stick input

if (g.rc\_4.control\_in < -3000) {

g.simple\_modes &= ~(1<<\_switchPosition);

// print new mode

print\_switch(\_switchPosition, mode, (g.simple\_modes & (1<<\_switchPosition)));

delay(500);

}

// escape hatch

if(cliSerial->available() > 0) {

for (mode = 0; mode < 6; mode++)

flight\_modes[mode].save();

g.simple\_modes.save();

print\_done();

report\_flight\_modes();

return (0);

}

}

}

static int8\_t

setup\_declination(uint8\_t argc, const Menu::arg \*argv)

{

compass.set\_declination(radians(argv[1].f));

report\_compass();

return 0;

}

static int8\_t

setup\_tune(uint8\_t argc, const Menu::arg \*argv)

{

g.radio\_tuning.set\_and\_save(argv[1].i);

//g.radio\_tuning\_high.set\_and\_save(1000);

//g.radio\_tuning\_low.set\_and\_save(0);

report\_tuning();

return 0;

}

static int8\_t

setup\_range(uint8\_t argc, const Menu::arg \*argv)

{

cliSerial->printf\_P(PSTR("\nCH 6 Ranges are divided by 1000: [low, high]\n"));

g.radio\_tuning\_low.set\_and\_save(argv[1].i);

g.radio\_tuning\_high.set\_and\_save(argv[2].i);

report\_tuning();

return 0;

}

static int8\_t

setup\_erase(uint8\_t argc, const Menu::arg \*argv)

{

zero\_eeprom();

return 0;

}

static int8\_t

setup\_compass(uint8\_t argc, const Menu::arg \*argv)

{

if (!strcmp\_P(argv[1].str, PSTR("on"))) {

g.compass\_enabled.set\_and\_save(true);

init\_compass();

} else if (!strcmp\_P(argv[1].str, PSTR("off"))) {

clear\_offsets();

g.compass\_enabled.set\_and\_save(false);

}else{

cliSerial->printf\_P(PSTR("\nOp:[on,off]\n"));

report\_compass();

return 0;

}

g.compass\_enabled.save();

report\_compass();

return 0;

}

static int8\_t

setup\_batt\_monitor(uint8\_t argc, const Menu::arg \*argv)

{

if (!strcmp\_P(argv[1].str, PSTR("off"))) {

g.battery\_monitoring.set\_and\_save(0);

} else if(argv[1].i > 0 && argv[1].i <= 4) {

g.battery\_monitoring.set\_and\_save(argv[1].i);

} else {

cliSerial->printf\_P(PSTR("\nOp: off, 3-4"));

}

report\_batt\_monitor();

return 0;

}

static int8\_t

setup\_sonar(uint8\_t argc, const Menu::arg \*argv)

{

if (!strcmp\_P(argv[1].str, PSTR("on"))) {

g.sonar\_enabled.set\_and\_save(true);

} else if (!strcmp\_P(argv[1].str, PSTR("off"))) {

g.sonar\_enabled.set\_and\_save(false);

} else if (argc > 1 && (argv[1].i >= 0 && argv[1].i <= 3)) {

g.sonar\_enabled.set\_and\_save(true); // if you set the sonar type, surely you want it on

g.sonar\_type.set\_and\_save(argv[1].i);

}else{

cliSerial->printf\_P(PSTR("\nOp:[on, off, 0-3]\n"));

report\_sonar();

return 0;

}

report\_sonar();

return 0;

}

#if FRAME\_CONFIG == HELI\_FRAME

// Perform heli setup.

// Called by the setup menu 'radio' command.

static int8\_t

setup\_heli(uint8\_t argc, const Menu::arg \*argv)

{

uint8\_t active\_servo = 0;

int16\_t value = 0;

int16\_t temp;

int16\_t state = 0; // 0 = set rev+pos, 1 = capture min/max

int16\_t max\_roll=0, max\_pitch=0, min\_collective=0, max\_collective=0, min\_tail=0, max\_tail=0;

// initialise swash plate

motors.init\_swash();

// source swash plate movements directly from radio

motors.servo\_manual = true;

// display initial settings

report\_heli();

// display help

cliSerial->printf\_P(PSTR("Instructions:"));

print\_divider();

cliSerial->printf\_P(PSTR("\td\t\tdisplay settings\n"));

cliSerial->printf\_P(PSTR("\t1~4\t\tselect servo\n"));

cliSerial->printf\_P(PSTR("\ta or z\t\tmove mid up/down\n"));

cliSerial->printf\_P(PSTR("\tc\t\tset coll when blade pitch zero\n"));

cliSerial->printf\_P(PSTR("\tm\t\tset roll, pitch, coll min/max\n"));

cliSerial->printf\_P(PSTR("\tp<angle>\tset pos (i.e. p0 = front, p90 = right)\n"));

cliSerial->printf\_P(PSTR("\tr\t\treverse servo\n"));

cliSerial->printf\_P(PSTR("\tu a|d\t\tupdate rate (a=analog servo, d=digital)\n"));

cliSerial->printf\_P(PSTR("\tt<angle>\tset trim (-500 ~ 500)\n"));

cliSerial->printf\_P(PSTR("\tx\t\texit & save\n"));

// start capturing

while( value != 'x' ) {

// read radio although we don't use it yet

read\_radio();

// allow swash plate to move

motors.output\_armed();

// record min/max

if( state == 1 ) {

if( abs(g.rc\_1.control\_in) > max\_roll )

max\_roll = abs(g.rc\_1.control\_in);

if( abs(g.rc\_2.control\_in) > max\_pitch )

max\_pitch = abs(g.rc\_2.control\_in);

if( g.rc\_3.radio\_out < min\_collective )

min\_collective = g.rc\_3.radio\_out;

if( g.rc\_3.radio\_out > max\_collective )

max\_collective = g.rc\_3.radio\_out;

min\_tail = min(g.rc\_4.radio\_out, min\_tail);

max\_tail = max(g.rc\_4.radio\_out, max\_tail);

}

if( cliSerial->available() ) {

value = cliSerial->read();

// process the user's input

switch( value ) {

case '1':

active\_servo = CH\_1;

break;

case '2':

active\_servo = CH\_2;

break;

case '3':

active\_servo = CH\_3;

break;

case '4':

active\_servo = CH\_4;

break;

case 'a':

case 'A':

heli\_get\_servo(active\_servo)->radio\_trim += 10;

break;

case 'c':

case 'C':

if( g.rc\_3.radio\_out >= 900 && g.rc\_3.radio\_out <= 2100 ) {

motors.collective\_mid = g.rc\_3.radio\_out;

cliSerial->printf\_P(PSTR("Collective when blade pitch at zero: %d\n"),(int)motors.collective\_mid);

}

break;

case 'd':

case 'D':

// display settings

report\_heli();

break;

case 'm':

case 'M':

if( state == 0 ) {

state = 1; // switch to capture min/max mode

cliSerial->printf\_P(PSTR("Move coll, roll, pitch and tail to extremes, press 'm' when done\n"));

// reset servo ranges

motors.roll\_max = motors.pitch\_max = 4500;

motors.collective\_min = 1000;

motors.collective\_max = 2000;

motors.\_servo\_4->radio\_min = 1000;

motors.\_servo\_4->radio\_max = 2000;

// set sensible values in temp variables

max\_roll = abs(g.rc\_1.control\_in);

max\_pitch = abs(g.rc\_2.control\_in);

min\_collective = 2000;

max\_collective = 1000;

min\_tail = max\_tail = abs(g.rc\_4.radio\_out);

}else{

state = 0; // switch back to normal mode

// double check values aren't totally terrible

if( max\_roll <= 1000 || max\_pitch <= 1000 || (max\_collective - min\_collective < 200) || (max\_tail - min\_tail < 200) || min\_tail < 1000 || max\_tail > 2000 )

cliSerial->printf\_P(PSTR("Invalid min/max captured roll:%d, pitch:%d, collective min: %d max: %d, tail min:%d max:%d\n"),max\_roll,max\_pitch,min\_collective,max\_collective,min\_tail,max\_tail);

else{

motors.roll\_max = max\_roll;

motors.pitch\_max = max\_pitch;

motors.collective\_min = min\_collective;

motors.collective\_max = max\_collective;

motors.\_servo\_4->radio\_min = min\_tail;

motors.\_servo\_4->radio\_max = max\_tail;

// reinitialise swash

motors.init\_swash();

// display settings

report\_heli();

}

}

break;

case 'p':

case 'P':

temp = read\_num\_from\_serial();

if( temp >= -360 && temp <= 360 ) {

if( active\_servo == CH\_1 )

motors.servo1\_pos = temp;

if( active\_servo == CH\_2 )

motors.servo2\_pos = temp;

if( active\_servo == CH\_3 )

motors.servo3\_pos = temp;

motors.init\_swash();

cliSerial->printf\_P(PSTR("Servo %d\t\tpos:%d\n"),active\_servo+1, temp);

}

break;

case 'r':

case 'R':

heli\_get\_servo(active\_servo)->set\_reverse(!heli\_get\_servo(active\_servo)->get\_reverse());

break;

case 't':

case 'T':

temp = read\_num\_from\_serial();

if( temp > 1000 )

temp -= 1500;

if( temp > -500 && temp < 500 ) {

heli\_get\_servo(active\_servo)->radio\_trim = 1500 + temp;

motors.init\_swash();

cliSerial->printf\_P(PSTR("Servo %d\t\ttrim:%d\n"),active\_servo+1, 1500 + temp);

}

break;

case 'u':

case 'U':

temp = 0;

// delay up to 2 seconds for servo type from user

while( !cliSerial->available() && temp < 20 ) {

temp++;

delay(100);

}

if( cliSerial->available() ) {

value = cliSerial->read();

if( value == 'a' || value == 'A' ) {

g.rc\_speed.set\_and\_save(AP\_MOTORS\_HELI\_SPEED\_ANALOG\_SERVOS);

//motors.\_speed\_hz = AP\_MOTORS\_HELI\_SPEED\_ANALOG\_SERVOS; // need to force this update to take effect immediately

cliSerial->printf\_P(PSTR("Analog Servo %dhz\n"),(int)g.rc\_speed);

}

if( value == 'd' || value == 'D' ) {

g.rc\_speed.set\_and\_save(AP\_MOTORS\_HELI\_SPEED\_ANALOG\_SERVOS);

//motors.\_speed\_hz = AP\_MOTORS\_HELI\_SPEED\_ANALOG\_SERVOS; // need to force this update to take effect immediately

cliSerial->printf\_P(PSTR("Digital Servo %dhz\n"),(int)g.rc\_speed);

}

}

break;

case 'z':

case 'Z':

heli\_get\_servo(active\_servo)->radio\_trim -= 10;

break;

}

}

delay(20);

}

// display final settings

report\_heli();

// save to eeprom

motors.\_servo\_1->save\_eeprom();

motors.\_servo\_2->save\_eeprom();

motors.\_servo\_3->save\_eeprom();

motors.\_servo\_4->save\_eeprom();

motors.servo1\_pos.save();

motors.servo2\_pos.save();

motors.servo3\_pos.save();

motors.roll\_max.save();

motors.pitch\_max.save();

motors.collective\_min.save();

motors.collective\_max.save();

motors.collective\_mid.save();

// return swash plate movements to attitude controller

motors.servo\_manual = false;

return(0);

}

// setup for external tail gyro (for heli only)

static int8\_t

setup\_gyro(uint8\_t argc, const Menu::arg \*argv)

{

if (!strcmp\_P(argv[1].str, PSTR("on"))) {

motors.ext\_gyro\_enabled.set\_and\_save(true);

// optionally capture the gain

if( argc >= 2 && argv[2].i >= 1000 && argv[2].i <= 2000 ) {

motors.ext\_gyro\_gain = argv[2].i;

motors.ext\_gyro\_gain.save();

}

} else if (!strcmp\_P(argv[1].str, PSTR("off"))) {

motors.ext\_gyro\_enabled.set\_and\_save(false);

// capture gain if user simply provides a number

} else if( argv[1].i >= 1000 && argv[1].i <= 2000 ) {

motors.ext\_gyro\_enabled.set\_and\_save(true);

motors.ext\_gyro\_gain = argv[1].i;

motors.ext\_gyro\_gain.save();

}else{

cliSerial->printf\_P(PSTR("\nOp:[on, off] gain\n"));

}

report\_gyro();

return 0;

}

#endif // FRAME\_CONFIG == HELI

static void clear\_offsets()

{

Vector3f \_offsets(0.0,0.0,0.0);

compass.set\_offsets(\_offsets);

compass.save\_offsets();

}

static int8\_t

setup\_optflow(uint8\_t argc, const Menu::arg \*argv)

{

#if OPTFLOW == ENABLED

if (!strcmp\_P(argv[1].str, PSTR("on"))) {

g.optflow\_enabled = true;

init\_optflow();

} else if (!strcmp\_P(argv[1].str, PSTR("off"))) {

g.optflow\_enabled = false;

}else{

cliSerial->printf\_P(PSTR("\nOp:[on, off]\n"));

report\_optflow();

return 0;

}

g.optflow\_enabled.save();

report\_optflow();

#endif // OPTFLOW == ENABLED

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// CLI reports

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void report\_batt\_monitor()

{

cliSerial->printf\_P(PSTR("\nBatt Mon:\n"));

print\_divider();

if(g.battery\_monitoring == 0) print\_enabled(false);

if(g.battery\_monitoring == 3) cliSerial->printf\_P(PSTR("volts"));

if(g.battery\_monitoring == 4) cliSerial->printf\_P(PSTR("volts and cur"));

print\_blanks(2);

}

static void report\_wp(byte index = 255)

{

if(index == 255) {

for(byte i = 0; i < g.command\_total; i++) {

struct Location temp = get\_cmd\_with\_index(i);

print\_wp(&temp, i);

}

}else{

struct Location temp = get\_cmd\_with\_index(index);

print\_wp(&temp, index);

}

}

static void report\_sonar()

{

cliSerial->printf\_P(PSTR("Sonar\n"));

print\_divider();

print\_enabled(g.sonar\_enabled.get());

cliSerial->printf\_P(PSTR("Type: %d (0=XL, 1=LV, 2=XLL, 3=HRLV)"), (int)g.sonar\_type);

print\_blanks(2);

}

static void report\_frame()

{

cliSerial->printf\_P(PSTR("Frame\n"));

print\_divider();

#if FRAME\_CONFIG == QUAD\_FRAME

cliSerial->printf\_P(PSTR("Quad frame\n"));

#elif FRAME\_CONFIG == TRI\_FRAME

cliSerial->printf\_P(PSTR("TRI frame\n"));

#elif FRAME\_CONFIG == HEXA\_FRAME

cliSerial->printf\_P(PSTR("Hexa frame\n"));

#elif FRAME\_CONFIG == Y6\_FRAME

cliSerial->printf\_P(PSTR("Y6 frame\n"));

#elif FRAME\_CONFIG == OCTA\_FRAME

cliSerial->printf\_P(PSTR("Octa frame\n"));

#elif FRAME\_CONFIG == HELI\_FRAME

cliSerial->printf\_P(PSTR("Heli frame\n"));

#endif

#if FRAME\_CONFIG != HELI\_FRAME

if(g.frame\_orientation == X\_FRAME)

cliSerial->printf\_P(PSTR("X mode\n"));

else if(g.frame\_orientation == PLUS\_FRAME)

cliSerial->printf\_P(PSTR("+ mode\n"));

else if(g.frame\_orientation == V\_FRAME)

cliSerial->printf\_P(PSTR("V mode\n"));

#endif

print\_blanks(2);

}

static void report\_radio()

{

cliSerial->printf\_P(PSTR("Radio\n"));

print\_divider();

// radio

print\_radio\_values();

print\_blanks(2);

}

static void report\_ins()

{

cliSerial->printf\_P(PSTR("INS\n"));

print\_divider();

print\_gyro\_offsets();

print\_accel\_offsets\_and\_scaling();

print\_blanks(2);

}

static void report\_compass()

{

cliSerial->printf\_P(PSTR("Compass\n"));

print\_divider();

print\_enabled(g.compass\_enabled);

// mag declination

cliSerial->printf\_P(PSTR("Mag Dec: %4.4f\n"),

degrees(compass.get\_declination()));

Vector3f offsets = compass.get\_offsets();

// mag offsets

cliSerial->printf\_P(PSTR("Mag off: %4.4f, %4.4f, %4.4f"),

offsets.x,

offsets.y,

offsets.z);

print\_blanks(2);

}

static void report\_flight\_modes()

{

cliSerial->printf\_P(PSTR("Flight modes\n"));

print\_divider();

for(int16\_t i = 0; i < 6; i++ ) {

print\_switch(i, flight\_modes[i], (g.simple\_modes & (1<<i)));

}

print\_blanks(2);

}

void report\_optflow()

{

#if OPTFLOW == ENABLED

cliSerial->printf\_P(PSTR("OptFlow\n"));

print\_divider();

print\_enabled(g.optflow\_enabled);

// field of view

//cliSerial->printf\_P(PSTR("FOV: %4.0f\n"),

// degrees(g.optflow\_fov));

print\_blanks(2);

#endif // OPTFLOW == ENABLED

}

#if FRAME\_CONFIG == HELI\_FRAME

static void report\_heli()

{

cliSerial->printf\_P(PSTR("Heli\n"));

print\_divider();

// main servo settings

cliSerial->printf\_P(PSTR("Servo \tpos \tmin \tmax \trev\n"));

cliSerial->printf\_P(PSTR("1:\t%d \t%d \t%d \t%d\n"),(int)motors.servo1\_pos, (int)motors.\_servo\_1->radio\_min, (int)motors.\_servo\_1->radio\_max, (int)motors.\_servo\_1->get\_reverse());

cliSerial->printf\_P(PSTR("2:\t%d \t%d \t%d \t%d\n"),(int)motors.servo2\_pos, (int)motors.\_servo\_2->radio\_min, (int)motors.\_servo\_2->radio\_max, (int)motors.\_servo\_2->get\_reverse());

cliSerial->printf\_P(PSTR("3:\t%d \t%d \t%d \t%d\n"),(int)motors.servo3\_pos, (int)motors.\_servo\_3->radio\_min, (int)motors.\_servo\_3->radio\_max, (int)motors.\_servo\_3->get\_reverse());

cliSerial->printf\_P(PSTR("tail:\t\t%d \t%d \t%d\n"), (int)motors.\_servo\_4->radio\_min, (int)motors.\_servo\_4->radio\_max, (int)motors.\_servo\_4->get\_reverse());

cliSerial->printf\_P(PSTR("roll max: \t%d\n"), (int)motors.roll\_max);

cliSerial->printf\_P(PSTR("pitch max: \t%d\n"), (int)motors.pitch\_max);

cliSerial->printf\_P(PSTR("coll min:\t%d\t mid:%d\t max:%d\n"),(int)motors.collective\_min, (int)motors.collective\_mid, (int)motors.collective\_max);

// calculate and print servo rate

cliSerial->printf\_P(PSTR("servo rate:\t%d hz\n"),(int)g.rc\_speed);

print\_blanks(2);

}

static void report\_gyro()

{

cliSerial->printf\_P(PSTR("Gyro:\n"));

print\_divider();

print\_enabled( motors.ext\_gyro\_enabled );

if( motors.ext\_gyro\_enabled )

cliSerial->printf\_P(PSTR("gain: %d"),(int)motors.ext\_gyro\_gain);

print\_blanks(2);

}

#endif // FRAME\_CONFIG == HELI\_FRAME

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// CLI utilities

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*static void

\* print\_PID(PI \* pid)

\* {

\* cliSerial->printf\_P(PSTR("P: %4.2f, I:%4.2f, IMAX:%ld\n"),

\* pid->kP(),

\* pid->kI(),

\* (long)pid->imax());

\* }

\*/

static void

print\_radio\_values()

{

cliSerial->printf\_P(PSTR("CH1: %d | %d\n"), (int)g.rc\_1.radio\_min, (int)g.rc\_1.radio\_max);

cliSerial->printf\_P(PSTR("CH2: %d | %d\n"), (int)g.rc\_2.radio\_min, (int)g.rc\_2.radio\_max);

cliSerial->printf\_P(PSTR("CH3: %d | %d\n"), (int)g.rc\_3.radio\_min, (int)g.rc\_3.radio\_max);

cliSerial->printf\_P(PSTR("CH4: %d | %d\n"), (int)g.rc\_4.radio\_min, (int)g.rc\_4.radio\_max);

cliSerial->printf\_P(PSTR("CH5: %d | %d\n"), (int)g.rc\_5.radio\_min, (int)g.rc\_5.radio\_max);

cliSerial->printf\_P(PSTR("CH6: %d | %d\n"), (int)g.rc\_6.radio\_min, (int)g.rc\_6.radio\_max);

cliSerial->printf\_P(PSTR("CH7: %d | %d\n"), (int)g.rc\_7.radio\_min, (int)g.rc\_7.radio\_max);

//cliSerial->printf\_P(PSTR("CH8: %d | %d\n"), (int)g.rc\_8.radio\_min, (int)g.rc\_8.radio\_max);

}

static void

print\_switch(byte p, byte m, bool b)

{

cliSerial->printf\_P(PSTR("Pos %d:\t"),p);

print\_flight\_mode(m);

cliSerial->printf\_P(PSTR(",\t\tSimple: "));

if(b)

cliSerial->printf\_P(PSTR("ON\n"));

else

cliSerial->printf\_P(PSTR("OFF\n"));

}

static void

print\_done()

{

cliSerial->printf\_P(PSTR("\nSaved\n"));

}

static void zero\_eeprom(void)

{

byte b = 0;

cliSerial->printf\_P(PSTR("\nErasing EEPROM\n"));

for (uintptr\_t i = 0; i < EEPROM\_MAX\_ADDR; i++) {

eeprom\_write\_byte((uint8\_t \*) i, b);

}

cliSerial->printf\_P(PSTR("done\n"));

}

static void

print\_accel\_offsets\_and\_scaling(void)

{

Vector3f accel\_offsets = ins.get\_accel\_offsets();

Vector3f accel\_scale = ins.get\_accel\_scale();

cliSerial->printf\_P(PSTR("A\_off: %4.2f, %4.2f, %4.2f\tA\_scale: %4.2f, %4.2f, %4.2f\n"),

(float)accel\_offsets.x, // Pitch

(float)accel\_offsets.y, // Roll

(float)accel\_offsets.z, // YAW

(float)accel\_scale.x, // Pitch

(float)accel\_scale.y, // Roll

(float)accel\_scale.z); // YAW

}

static void

print\_gyro\_offsets(void)

{

Vector3f gyro\_offsets = ins.get\_gyro\_offsets();

cliSerial->printf\_P(PSTR("G\_off: %4.2f, %4.2f, %4.2f\n"),

(float)gyro\_offsets.x,

(float)gyro\_offsets.y,

(float)gyro\_offsets.z);

}

#if FRAME\_CONFIG == HELI\_FRAME

static RC\_Channel \*

heli\_get\_servo(int16\_t servo\_num){

if( servo\_num == CH\_1 )

return motors.\_servo\_1;

if( servo\_num == CH\_2 )

return motors.\_servo\_2;

if( servo\_num == CH\_3 )

return motors.\_servo\_3;

if( servo\_num == CH\_4 )

return motors.\_servo\_4;

return NULL;

}

// Used to read integer values from the serial port

static int16\_t read\_num\_from\_serial() {

byte index = 0;

byte timeout = 0;

char data[5] = "";

do {

if (cliSerial->available() == 0) {

delay(10);

timeout++;

}else{

data[index] = cliSerial->read();

timeout = 0;

index++;

}

} while (timeout < 5 && index < 5);

return atoi(data);

}

#endif

#endif // CLI\_ENABLED

static void

print\_blanks(int16\_t num)

{

while(num > 0) {

num--;

cliSerial->println("");

}

}

static void

print\_divider(void)

{

for (int i = 0; i < 40; i++) {

cliSerial->print\_P(PSTR("-"));

}

cliSerial->println();

}

static void print\_enabled(bool b)

{

if(b)

cliSerial->print\_P(PSTR("en"));

else

cliSerial->print\_P(PSTR("dis"));

cliSerial->print\_P(PSTR("abled\n"));

}

static void

init\_esc()

{

// reduce update rate to motors to 50Hz

motors.set\_update\_rate(50);

motors.enable();

motors.armed(true);

while(1) {

read\_radio();

delay(100);

dancing\_light();

motors.throttle\_pass\_through();

}

}

static void print\_wp(struct Location \*cmd, byte index)

{

//float t1 = (float)cmd->lat / t7;

//float t2 = (float)cmd->lng / t7;

cliSerial->printf\_P(PSTR("cmd#: %d | %d, %d, %d, %ld, %ld, %ld\n"),

index,

cmd->id,

cmd->options,

cmd->p1,

cmd->alt,

cmd->lat,

cmd->lng);

/\*

cliSerial->printf\_P(PSTR("cmd#: %d id:%d op:%d p1:%d p2:%ld p3:%4.7f p4:%4.7f \n"),

(int)index,

(int)cmd->id,

(int)cmd->options,

(int)cmd->p1,

(long)cmd->alt,

t1,

t2);

\*/

}

static void report\_version()

{

cliSerial->printf\_P(PSTR("FW Ver: %d\n"),(int)g.k\_format\_version);

print\_divider();

print\_blanks(2);

}

static void report\_tuning()

{

cliSerial->printf\_P(PSTR("\nTUNE:\n"));

print\_divider();

if (g.radio\_tuning == 0) {

print\_enabled(g.radio\_tuning.get());

}else{

float low = (float)g.radio\_tuning\_low.get() / 1000;

float high = (float)g.radio\_tuning\_high.get() / 1000;

cliSerial->printf\_P(PSTR(" %d, Low:%1.4f, High:%1.4f\n"),(int)g.radio\_tuning.get(), low, high);

}

print\_blanks(2);

}

## system

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The init\_ardupilot function processes everything we need for an in - air restart

\* We will determine later if we are actually on the ground and process a

\* ground start in that case.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#if CLI\_ENABLED == ENABLED

// Functions called from the top-level menu

static int8\_t process\_logs(uint8\_t argc, const Menu::arg \*argv); // in Log.pde

static int8\_t setup\_mode(uint8\_t argc, const Menu::arg \*argv); // in setup.pde

static int8\_t test\_mode(uint8\_t argc, const Menu::arg \*argv); // in test.cpp

static int8\_t reboot\_board(uint8\_t argc, const Menu::arg \*argv);

// This is the help function

// PSTR is an AVR macro to read strings from flash memory

// printf\_P is a version of print\_f that reads from flash memory

static int8\_t main\_menu\_help(uint8\_t argc, const Menu::arg \*argv)

{

cliSerial->printf\_P(PSTR("Commands:\n"

" logs\n"

" setup\n"

" test\n"

" reboot\n"

"\n"));

return(0);

}

// Command/function table for the top-level menu.

const struct Menu::command main\_menu\_commands[] PROGMEM = {

// command function called

// ======= ===============

{"logs", process\_logs},

{"setup", setup\_mode},

{"test", test\_mode},

{"reboot", reboot\_board},

{"help", main\_menu\_help},

};

// Create the top-level menu object.

MENU(main\_menu, THISFIRMWARE, main\_menu\_commands);

static int8\_t reboot\_board(uint8\_t argc, const Menu::arg \*argv)

{

reboot\_apm();

return 0;

}

// the user wants the CLI. It never exits

static void run\_cli(FastSerial \*port)

{

cliSerial = port;

Menu::set\_port(port);

port->set\_blocking\_writes(true);

while (1) {

main\_menu.run();

}

}

#endif // CLI\_ENABLED

static void init\_ardupilot()

{

#if USB\_MUX\_PIN > 0

// on the APM2 board we have a mux thet switches UART0 between

// USB and the board header. If the right ArduPPM firmware is

// installed we can detect if USB is connected using the

// USB\_MUX\_PIN

pinMode(USB\_MUX\_PIN, INPUT);

ap\_system.usb\_connected = !digitalReadFast(USB\_MUX\_PIN);

if (!ap\_system.usb\_connected) {

// USB is not connected, this means UART0 may be a Xbee, with

// its darned bricking problem. We can't write to it for at

// least one second after powering up. Simplest solution for

// now is to delay for 1 second. Something more elegant may be

// added later

delay(1000);

}

#endif

// Console serial port

//

// The console port buffers are defined to be sufficiently large to support

// the MAVLink protocol efficiently

//

cliSerial->begin(SERIAL0\_BAUD, 128, 256);

// GPS serial port.

//

#if GPS\_PROTOCOL != GPS\_PROTOCOL\_IMU

// standard gps running. Note that we need a 256 byte buffer for some

// GPS types (eg. UBLOX)

Serial1.begin(38400, 256, 16);

#endif

cliSerial->printf\_P(PSTR("\n\nInit " THISFIRMWARE

"\n\nFree RAM: %u\n"),

memcheck\_available\_memory());

//

// Initialize Wire and SPI libraries

//

#ifndef DESKTOP\_BUILD

I2c.begin();

I2c.timeOut(5);

// initially set a fast I2c speed, and drop it on first failures

I2c.setSpeed(true);

#endif

SPI.begin();

SPI.setClockDivider(SPI\_CLOCK\_DIV16); // 1MHZ SPI rate

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

SPI3.begin();

SPI3.setSpeed(SPI3\_SPEED\_2MHZ);

#endif

//

// Initialize the isr\_registry.

//

isr\_registry.init();

//

// Report firmware version code expect on console (check of actual EEPROM format version is done in load\_parameters function)

//

report\_version();

// setup IO pins

pinMode(A\_LED\_PIN, OUTPUT); // GPS status LED

digitalWrite(A\_LED\_PIN, LED\_OFF);

pinMode(B\_LED\_PIN, OUTPUT); // GPS status LED

digitalWrite(B\_LED\_PIN, LED\_OFF);

pinMode(C\_LED\_PIN, OUTPUT); // GPS status LED

digitalWrite(C\_LED\_PIN, LED\_OFF);

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

// Set Port B, pin 7 as output for an external relay (A9 Enclosure's label)

pinMode(RELAY\_APM2\_PIN, OUTPUT);

#else

// Set Port L, pin 2 as output for the onboard relay

pinMode(RELAY\_APM1\_PIN, OUTPUT);

#endif

#if SLIDE\_SWITCH\_PIN > 0

pinMode(SLIDE\_SWITCH\_PIN, INPUT); // To enter interactive mode

#endif

#if CONFIG\_PUSHBUTTON == ENABLED

pinMode(PUSHBUTTON\_PIN, INPUT); // unused

#endif

#if COPTER\_LEDS == ENABLED

pinMode(COPTER\_LED\_1, OUTPUT); //Motor LED

pinMode(COPTER\_LED\_2, OUTPUT); //Motor LED

pinMode(COPTER\_LED\_3, OUTPUT); //Motor LED

pinMode(COPTER\_LED\_4, OUTPUT); //Motor LED

pinMode(COPTER\_LED\_5, OUTPUT); //Motor or Aux LED

pinMode(COPTER\_LED\_6, OUTPUT); //Motor or Aux LED

pinMode(COPTER\_LED\_7, OUTPUT); //Motor or GPS LED

pinMode(COPTER\_LED\_8, OUTPUT); //Motor or GPS LED

if ( !bitRead(g.copter\_leds\_mode, 3) ) {

piezo\_beep();

}

#endif

// load parameters from EEPROM

load\_parameters();

// init the GCS

gcs0.init(&Serial);

#if USB\_MUX\_PIN > 0

if (!ap\_system.usb\_connected) {

// we are not connected via USB, re-init UART0 with right

// baud rate

cliSerial->begin(map\_baudrate(g.serial3\_baud, SERIAL3\_BAUD));

}

#else

// we have a 2nd serial port for telemetry

Serial3.begin(map\_baudrate(g.serial3\_baud, SERIAL3\_BAUD), 128, 256);

gcs3.init(&Serial3);

#endif

// identify ourselves correctly with the ground station

mavlink\_system.sysid = g.sysid\_this\_mav;

mavlink\_system.type = 2; //MAV\_QUADROTOR;

#if LOGGING\_ENABLED == ENABLED

DataFlash.Init();

if (!DataFlash.CardInserted()) {

gcs\_send\_text\_P(SEVERITY\_LOW, PSTR("No dataflash inserted"));

g.log\_bitmask.set(0);

} else if (DataFlash.NeedErase()) {

gcs\_send\_text\_P(SEVERITY\_LOW, PSTR("ERASING LOGS"));

do\_erase\_logs();

}

if (g.log\_bitmask != 0) {

DataFlash.start\_new\_log();

}

#endif

#if FRAME\_CONFIG == HELI\_FRAME

motors.servo\_manual = false;

motors.init\_swash(); // heli initialisation

#endif

RC\_Channel::set\_apm\_rc(&APM\_RC);

init\_rc\_in(); // sets up rc channels from radio

init\_rc\_out(); // sets up the timer libs

timer\_scheduler.init( &isr\_registry );

/\*

\* setup the 'main loop is dead' check. Note that this relies on

\* the RC library being initialised.

\*/

timer\_scheduler.set\_failsafe(failsafe\_check);

// initialise the analog port reader

AP\_AnalogSource\_Arduino::init\_timer(&timer\_scheduler);

#if HIL\_MODE != HIL\_MODE\_ATTITUDE

#if CONFIG\_ADC == ENABLED

// begin filtering the ADC Gyros

adc.Init(&timer\_scheduler); // APM ADC library initialization

#endif // CONFIG\_ADC

barometer.init(&timer\_scheduler);

#endif // HIL\_MODE

// Do GPS init

g\_gps = &g\_gps\_driver;

// GPS Initialization

g\_gps->init(GPS::GPS\_ENGINE\_AIRBORNE\_1G);

if(g.compass\_enabled)

init\_compass();

// init the optical flow sensor

if(g.optflow\_enabled) {

init\_optflow();

}

#if INERTIAL\_NAV\_XY == ENABLED || INERTIAL\_NAV\_Z == ENABLED

// initialise inertial nav

inertial\_nav.init();

#endif

// agmatthews USERHOOKS

#ifdef USERHOOK\_INIT

USERHOOK\_INIT

#endif

#if CLI\_ENABLED == ENABLED && CLI\_SLIDER\_ENABLED == ENABLED

// If the switch is in 'menu' mode, run the main menu.

//

// Since we can't be sure that the setup or test mode won't leave

// the system in an odd state, we don't let the user exit the top

// menu; they must reset in order to fly.

//

if (check\_startup\_for\_CLI()) {

digitalWrite(A\_LED\_PIN, LED\_ON); // turn on setup-mode LED

cliSerial->printf\_P(PSTR("\nCLI:\n\n"));

run\_cli(cliSerial);

}

#else

const prog\_char\_t \*msg = PSTR("\nPress ENTER 3 times to start interactive setup\n");

cliSerial->println\_P(msg);

#if USB\_MUX\_PIN == 0

Serial3.println\_P(msg);

#endif

#endif // CLI\_ENABLED

#if HIL\_MODE != HIL\_MODE\_ATTITUDE

// read Baro pressure at ground

//-----------------------------

init\_barometer();

#endif

// initialise sonar

#if CONFIG\_SONAR == ENABLED

init\_sonar();

#endif

#if FRAME\_CONIG == HELI\_FRAME

// initialise controller filters

init\_rate\_controllers();

#endif // HELI\_FRAME

// initialize commands

// -------------------

init\_commands();

// set the correct flight mode

// ---------------------------

reset\_control\_switch();

startup\_ground();

// now that initialisation of IMU has occurred increase SPI to 2MHz

SPI.setClockDivider(SPI\_CLOCK\_DIV8); // 2MHZ SPI rate

#if LOGGING\_ENABLED == ENABLED

Log\_Write\_Startup();

#endif

///////////////////////////////////////////////////////////////////////////////

// Experimental AP\_Limits library - set constraints, limits, fences, minima, maxima on various parameters

////////////////////////////////////////////////////////////////////////////////

#if AP\_LIMITS == ENABLED

// AP\_Limits modules are stored as a \_linked list\_. That allows us to define an infinite number of modules

// and also to allocate no space until we actually need to.

// The linked list looks (logically) like this

// [limits module] -> [first limit module] -> [second limit module] -> [third limit module] -> NULL

// The details of the linked list are handled by the methods

// modules\_first, modules\_current, modules\_next, modules\_last, modules\_add

// in limits

limits.modules\_add(&gpslock\_limit);

limits.modules\_add(&geofence\_limit);

limits.modules\_add(&altitude\_limit);

if (limits.debug()) {

gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("Limits Modules Loaded"));

AP\_Limit\_Module \*m = limits.modules\_first();

while (m) {

gcs\_send\_text\_P(SEVERITY\_LOW, get\_module\_name(m->get\_module\_id()));

m = limits.modules\_next();

}

}

#endif

cliSerial->print\_P(PSTR("\nReady to FLY "));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//This function does all the calibrations, etc. that we need during a ground start

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static void startup\_ground(void)

{

gcs\_send\_text\_P(SEVERITY\_LOW,PSTR("GROUND START"));

// Warm up and read Gyro offsets

// -----------------------------

ins.init(AP\_InertialSensor::COLD\_START,

ins\_sample\_rate,

mavlink\_delay, flash\_leds, &timer\_scheduler);

#if CLI\_ENABLED == ENABLED

report\_ins();

#endif

// initialise ahrs (may push imu calibration into the mpu6000 if using that device).

ahrs.init(&timer\_scheduler);

// setup fast AHRS gains to get right attitude

ahrs.set\_fast\_gains(true);

#if SECONDARY\_DMP\_ENABLED == ENABLED

ahrs2.init(&timer\_scheduler);

ahrs2.set\_as\_secondary(true);

ahrs2.set\_fast\_gains(true);

#endif

// reset the leds

// ---------------------------

clear\_leds();

// when we re-calibrate the gyros,

// all previous I values are invalid

reset\_I\_all();

}

// set\_mode - change flight mode and perform any necessary initialisation

static void set\_mode(byte mode)

{

// Switch to stabilize mode if requested mode requires a GPS lock

if(!ap.home\_is\_set) {

if (mode > ALT\_HOLD && mode != TOY\_A && mode != TOY\_M && mode != OF\_LOITER && mode != LAND) {

mode = STABILIZE;

}

}

// Switch to stabilize if OF\_LOITER requested but no optical flow sensor

if (mode == OF\_LOITER && !g.optflow\_enabled ) {

mode = STABILIZE;

}

control\_mode = mode;

control\_mode = constrain(control\_mode, 0, NUM\_MODES - 1);

// used to stop fly\_aways

// set to false if we have low throttle

motors.auto\_armed(g.rc\_3.control\_in > 0 || ap.failsafe);

set\_auto\_armed(g.rc\_3.control\_in > 0 || ap.failsafe);

// if we change modes, we must clear landed flag

set\_land\_complete(false);

// debug to Serial terminal

//cliSerial->println(flight\_mode\_strings[control\_mode]);

ap.loiter\_override = false;

// report the GPS and Motor arming status

led\_mode = NORMAL\_LEDS;

switch(control\_mode)

{

case ACRO:

ap.manual\_throttle = true;

ap.manual\_attitude = true;

set\_yaw\_mode(YAW\_ACRO);

set\_roll\_pitch\_mode(ROLL\_PITCH\_ACRO);

set\_throttle\_mode(THROTTLE\_MANUAL);

// reset acro axis targets to current attitude

if(g.axis\_enabled){

roll\_axis = ahrs.roll\_sensor;

pitch\_axis = ahrs.pitch\_sensor;

nav\_yaw = ahrs.yaw\_sensor;

}

break;

case STABILIZE:

ap.manual\_throttle = true;

ap.manual\_attitude = true;

set\_yaw\_mode(YAW\_HOLD);

set\_roll\_pitch\_mode(ROLL\_PITCH\_STABLE);

set\_throttle\_mode(STABILIZE\_THROTTLE);

break;

case ALT\_HOLD:

ap.manual\_throttle = false;

ap.manual\_attitude = true;

set\_yaw\_mode(ALT\_HOLD\_YAW);

set\_roll\_pitch\_mode(ALT\_HOLD\_RP);

set\_throttle\_mode(ALT\_HOLD\_THR);

break;

case AUTO:

ap.manual\_throttle = false;

ap.manual\_attitude = false;

set\_yaw\_mode(AUTO\_YAW);

set\_roll\_pitch\_mode(AUTO\_RP);

set\_throttle\_mode(AUTO\_THR);

// loads the commands from where we left off

init\_commands();

break;

case CIRCLE:

ap.manual\_throttle = false;

ap.manual\_attitude = false;

// start circling around current location

set\_next\_WP(&current\_loc);

circle\_WP = next\_WP;

// set yaw to point to center of circle

yaw\_look\_at\_WP = circle\_WP;

set\_yaw\_mode(YAW\_LOOK\_AT\_LOCATION);

set\_roll\_pitch\_mode(CIRCLE\_RP);

set\_throttle\_mode(CIRCLE\_THR);

circle\_angle = 0;

break;

case LOITER:

ap.manual\_throttle = false;

ap.manual\_attitude = false;

set\_yaw\_mode(LOITER\_YAW);

set\_roll\_pitch\_mode(LOITER\_RP);

set\_throttle\_mode(LOITER\_THR);

set\_next\_WP(&current\_loc);

break;

case POSITION:

ap.manual\_throttle = true;

ap.manual\_attitude = false;

set\_yaw\_mode(YAW\_HOLD);

set\_roll\_pitch\_mode(LOITER\_RP);

set\_throttle\_mode(THROTTLE\_MANUAL);

set\_next\_WP(&current\_loc);

break;

case GUIDED:

ap.manual\_throttle = false;

ap.manual\_attitude = false;

set\_yaw\_mode(GUIDED\_YAW);

set\_roll\_pitch\_mode(GUIDED\_RP);

set\_throttle\_mode(GUIDED\_THR);

wp\_control = WP\_MODE;

wp\_verify\_byte = 0;

set\_next\_WP(&guided\_WP);

break;

case LAND:

if( ap.home\_is\_set ) {

// switch to loiter if we have gps

ap.manual\_attitude = false;

set\_yaw\_mode(LOITER\_YAW);

set\_roll\_pitch\_mode(LOITER\_RP);

}else{

// otherwise remain with stabilize roll and pitch

ap.manual\_attitude = true;

set\_yaw\_mode(YAW\_HOLD);

set\_roll\_pitch\_mode(ROLL\_PITCH\_STABLE);

}

ap.manual\_throttle = false;

do\_land();

break;

case RTL:

ap.manual\_throttle = false;

ap.manual\_attitude = false;

do\_RTL();

break;

case OF\_LOITER:

ap.manual\_throttle = false;

ap.manual\_attitude = false;

set\_yaw\_mode(OF\_LOITER\_YAW);

set\_roll\_pitch\_mode(OF\_LOITER\_RP);

set\_throttle\_mode(OF\_LOITER\_THR);

set\_next\_WP(&current\_loc);

break;

// THOR

// These are the flight modes for Toy mode

// See the defines for the enumerated values

case TOY\_A:

ap.manual\_throttle = false;

ap.manual\_attitude = true;

set\_yaw\_mode(YAW\_TOY);

set\_roll\_pitch\_mode(ROLL\_PITCH\_TOY);

set\_throttle\_mode(THROTTLE\_AUTO);

wp\_control = NO\_NAV\_MODE;

// save throttle for fast exit of Alt hold

saved\_toy\_throttle = g.rc\_3.control\_in;

break;

case TOY\_M:

ap.manual\_throttle = false;

ap.manual\_attitude = true;

set\_yaw\_mode(YAW\_TOY);

set\_roll\_pitch\_mode(ROLL\_PITCH\_TOY);

wp\_control = NO\_NAV\_MODE;

set\_throttle\_mode(THROTTLE\_HOLD);

break;

default:

break;

}

if(ap.manual\_attitude) {

// We are under manual attitude control

// remove the navigation from roll and pitch command

reset\_nav\_params();

// remove the wind compenstaion

reset\_wind\_I();

}

Log\_Write\_Mode(control\_mode);

}

static void

init\_simple\_bearing()

{

initial\_simple\_bearing = ahrs.yaw\_sensor;

Log\_Write\_Data(DATA\_INIT\_SIMPLE\_BEARING, initial\_simple\_bearing);

}

#if CLI\_SLIDER\_ENABLED == ENABLED && CLI\_ENABLED == ENABLED

static boolean

check\_startup\_for\_CLI()

{

return (digitalReadFast(SLIDE\_SWITCH\_PIN) == 0);

}

#endif // CLI\_ENABLED

/\*

\* map from a 8 bit EEPROM baud rate to a real baud rate

\*/

static uint32\_t map\_baudrate(int8\_t rate, uint32\_t default\_baud)

{

switch (rate) {

case 1: return 1200;

case 2: return 2400;

case 4: return 4800;

case 9: return 9600;

case 19: return 19200;

case 38: return 38400;

case 57: return 57600;

case 111: return 111100;

case 115: return 115200;

}

//cliSerial->println\_P(PSTR("Invalid SERIAL3\_BAUD"));

return default\_baud;

}

#if USB\_MUX\_PIN > 0

static void check\_usb\_mux(void)

{

bool usb\_check = !digitalReadFast(USB\_MUX\_PIN);

if (usb\_check == ap\_system.usb\_connected) {

return;

}

// the user has switched to/from the telemetry port

ap\_system.usb\_connected = usb\_check;

if (ap\_system.usb\_connected) {

cliSerial->begin(SERIAL0\_BAUD);

} else {

cliSerial->begin(map\_baudrate(g.serial3\_baud, SERIAL3\_BAUD));

}

}

#endif

/\*

\* called by gyro/accel init to flash LEDs so user

\* has some mesmerising lights to watch while waiting

\*/

void flash\_leds(bool on)

{

digitalWrite(A\_LED\_PIN, on ? LED\_OFF : LED\_ON);

digitalWrite(C\_LED\_PIN, on ? LED\_ON : LED\_OFF);

}

#ifndef DESKTOP\_BUILD

/\*

\* Read Vcc vs 1.1v internal reference

\*/

uint16\_t board\_voltage(void)

{

static AP\_AnalogSource\_Arduino vcc(ANALOG\_PIN\_VCC);

return vcc.read\_vcc();

}

#endif

/\*

force a software reset of the APM

\*/

static void reboot\_apm(void)

{

cliSerial->printf\_P(PSTR("REBOOTING\n"));

delay(100); // let serial flush

// see http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1250663814/

// for the method

#if CONFIG\_APM\_HARDWARE == APM\_HARDWARE\_APM2

// this relies on the bootloader resetting the watchdog, which

// APM1 doesn't do

cli();

wdt\_enable(WDTO\_15MS);

#else

// this works on APM1

void (\*fn)(void) = NULL;

fn();

#endif

while (1);

}

//

// print\_flight\_mode - prints flight mode to serial port.

//

static void

print\_flight\_mode(uint8\_t mode)

{

switch (mode) {

case STABILIZE:

cliSerial->print\_P(PSTR("STABILIZE"));

break;

case ACRO:

cliSerial->print\_P(PSTR("ACRO"));

break;

case ALT\_HOLD:

cliSerial->print\_P(PSTR("ALT\_HOLD"));

break;

case AUTO:

cliSerial->print\_P(PSTR("AUTO"));

break;

case GUIDED:

cliSerial->print\_P(PSTR("GUIDED"));

break;

case LOITER:

cliSerial->print\_P(PSTR("LOITER"));

break;

case RTL:

cliSerial->print\_P(PSTR("RTL"));

break;

case CIRCLE:

cliSerial->print\_P(PSTR("CIRCLE"));

break;

case POSITION:

cliSerial->print\_P(PSTR("POSITION"));

break;

case LAND:

cliSerial->print\_P(PSTR("LAND"));

break;

case OF\_LOITER:

cliSerial->print\_P(PSTR("OF\_LOITER"));

break;

case TOY\_M:

cliSerial->print\_P(PSTR("TOY\_M"));

break;

case TOY\_A:

cliSerial->print\_P(PSTR("TOY\_A"));

break;

default:

cliSerial->print\_P(PSTR("---"));

break;

}

}

## test

// -\*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -\*-

#if CLI\_ENABLED == ENABLED

// These are function definitions so the Menu can be constructed before the functions

// are defined below. Order matters to the compiler.

static int8\_t test\_radio\_pwm(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_radio(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_failsafe(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_stabilize(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_gps(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_tri(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_adc(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_ins(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_imu(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_dcm\_eulers(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_dcm(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_omega(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_stab\_d(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_battery(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_toy(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_wp\_nav(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_reverse(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_tuning(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_relay(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_wp(uint8\_t argc, const Menu::arg \*argv);

#if HIL\_MODE != HIL\_MODE\_ATTITUDE

static int8\_t test\_baro(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_sonar(uint8\_t argc, const Menu::arg \*argv);

#endif

static int8\_t test\_mag(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_optflow(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_logging(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_xbee(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_eedump(uint8\_t argc, const Menu::arg \*argv);

static int8\_t test\_rawgps(uint8\_t argc, const Menu::arg \*argv);

//static int8\_t test\_mission(uint8\_t argc, const Menu::arg \*argv);

// this is declared here to remove compiler errors

extern void print\_latlon(BetterStream \*s, int32\_t lat\_or\_lon); // in Log.pde

// This is the help function

// PSTR is an AVR macro to read strings from flash memory

// printf\_P is a version of printf that reads from flash memory

/\*static int8\_t help\_test(uint8\_t argc, const Menu::arg \*argv)

\* {

\* cliSerial->printf\_P(PSTR("\n"

\* "Commands:\n"

\* " radio\n"

\* " servos\n"

\* " g\_gps\n"

\* " imu\n"

\* " battery\n"

\* "\n"));

\* }\*/

// Creates a constant array of structs representing menu options

// and stores them in Flash memory, not RAM.

// User enters the string in the console to call the functions on the right.

// See class Menu in AP\_Coommon for implementation details

const struct Menu::command test\_menu\_commands[] PROGMEM = {

{"pwm", test\_radio\_pwm},

{"radio", test\_radio},

// {"failsafe", test\_failsafe},

// {"stabilize", test\_stabilize},

{"gps", test\_gps},

// {"adc", test\_adc},

{"ins", test\_ins},

// {"dcm", test\_dcm\_eulers},

//{"omega", test\_omega},

// {"stab\_d", test\_stab\_d},

{"battery", test\_battery},

{"tune", test\_tuning},

//{"tri", test\_tri},

{"relay", test\_relay},

{"wp", test\_wp},

// {"toy", test\_toy},

#if HIL\_MODE != HIL\_MODE\_ATTITUDE

{"altitude", test\_baro},

{"sonar", test\_sonar},

#endif

{"compass", test\_mag},

{"optflow", test\_optflow},

//{"xbee", test\_xbee},

{"eedump", test\_eedump},

{"logging", test\_logging},

// {"rawgps", test\_rawgps},

// {"mission", test\_mission},

//{"reverse", test\_reverse},

{"nav", test\_wp\_nav},

};

// A Macro to create the Menu

MENU(test\_menu, "test", test\_menu\_commands);

static int8\_t

test\_mode(uint8\_t argc, const Menu::arg \*argv)

{

//cliSerial->printf\_P(PSTR("Test Mode\n\n"));

test\_menu.run();

return 0;

}

static int8\_t

test\_eedump(uint8\_t argc, const Menu::arg \*argv)

{

uintptr\_t i, j;

// hexdump the EEPROM

for (i = 0; i < EEPROM\_MAX\_ADDR; i += 16) {

cliSerial->printf\_P(PSTR("%04x:"), i);

for (j = 0; j < 16; j++)

cliSerial->printf\_P(PSTR(" %02x"), eeprom\_read\_byte((const uint8\_t \*)(i + j)));

cliSerial->println();

}

return(0);

}

static int8\_t

test\_radio\_pwm(uint8\_t argc, const Menu::arg \*argv)

{

#if defined( \_\_AVR\_ATmega1280\_\_ ) // test disabled to save code size for 1280

print\_test\_disabled();

return (0);

#else

print\_hit\_enter();

delay(1000);

while(1) {

delay(20);

// Filters radio input - adjust filters in the radio.pde file

// ----------------------------------------------------------

read\_radio();

// servo Yaw

//APM\_RC.OutputCh(CH\_7, g.rc\_4.radio\_out);

cliSerial->printf\_P(PSTR("IN: 1: %d\t2: %d\t3: %d\t4: %d\t5: %d\t6: %d\t7: %d\t8: %d\n"),

g.rc\_1.radio\_in,

g.rc\_2.radio\_in,

g.rc\_3.radio\_in,

g.rc\_4.radio\_in,

g.rc\_5.radio\_in,

g.rc\_6.radio\_in,

g.rc\_7.radio\_in,

g.rc\_8.radio\_in);

if(cliSerial->available() > 0) {

return (0);

}

}

#endif

}

/\*

\* //static int8\_t

\* //test\_tri(uint8\_t argc, const Menu::arg \*argv)

\* {

\* print\_hit\_enter();

\* delay(1000);

\*

\* while(1){

\* delay(20);

\*

\* // Filters radio input - adjust filters in the radio.pde file

\* // ----------------------------------------------------------

\* read\_radio();

\* g.rc\_4.servo\_out = g.rc\_4.control\_in;

\* g.rc\_4.calc\_pwm();

\*

\* cliSerial->printf\_P(PSTR("input: %d\toutput%d\n"),

\* g.rc\_4.control\_in,

\* g.rc\_4.radio\_out);

\*

\* APM\_RC.OutputCh(CH\_TRI\_YAW, g.rc\_4.radio\_out);

\*

\* if(cliSerial->available() > 0){

\* return (0);

\* }

\* }

\* }\*/

/\*

//static int8\_t

//test\_toy(uint8\_t argc, const Menu::arg \*argv)

{

set\_alt\_change(ASCENDING)

for(altitude\_error = 2000; altitude\_error > -100; altitude\_error--){

int16\_t temp = get\_desired\_climb\_rate();

cliSerial->printf("%ld, %d\n", altitude\_error, temp);

}

return 0;

}

{ wp\_distance = 0;

int16\_t max\_speed = 0;

for(int16\_t i = 0; i < 200; i++){

int32\_t temp = 2 \* 100 \* (wp\_distance - g.waypoint\_radius \* 100);

max\_speed = sqrt((float)temp);

max\_speed = min(max\_speed, g.waypoint\_speed\_max);

cliSerial->printf("Zspeed: %ld, %d, %ld\n", temp, max\_speed, wp\_distance);

wp\_distance += 100;

}

return 0;

}

//\*/

/\*static int8\_t

\* //test\_toy(uint8\_t argc, const Menu::arg \*argv)

\* {

\* int16\_t yaw\_rate;

\* int16\_t roll\_rate;

\* g.rc\_1.control\_in = -2500;

\* g.rc\_2.control\_in = 2500;

\*

\* g.toy\_yaw\_rate = 3;

\* yaw\_rate = g.rc\_1.control\_in / g.toy\_yaw\_rate;

\* roll\_rate = ((int32\_t)g.rc\_2.control\_in \* (yaw\_rate/100)) /40;

\* cliSerial->printf("yaw\_rate, %d, roll\_rate, %d\n", yaw\_rate, roll\_rate);

\*

\* g.toy\_yaw\_rate = 2;

\* yaw\_rate = g.rc\_1.control\_in / g.toy\_yaw\_rate;

\* roll\_rate = ((int32\_t)g.rc\_2.control\_in \* (yaw\_rate/100)) /40;

\* cliSerial->printf("yaw\_rate, %d, roll\_rate, %d\n", yaw\_rate, roll\_rate);

\*

\* g.toy\_yaw\_rate = 1;

\* yaw\_rate = g.rc\_1.control\_in / g.toy\_yaw\_rate;

\* roll\_rate = ((int32\_t)g.rc\_2.control\_in \* (yaw\_rate/100)) /40;

\* cliSerial->printf("yaw\_rate, %d, roll\_rate, %d\n", yaw\_rate, roll\_rate);

\* }\*/

static int8\_t

test\_radio(uint8\_t argc, const Menu::arg \*argv)

{

print\_hit\_enter();

delay(1000);

while(1) {

delay(20);

read\_radio();

cliSerial->printf\_P(PSTR("IN 1: %d\t2: %d\t3: %d\t4: %d\t5: %d\t6: %d\t7: %d\n"),

g.rc\_1.control\_in,

g.rc\_2.control\_in,

g.rc\_3.control\_in,

g.rc\_4.control\_in,

g.rc\_5.control\_in,

g.rc\_6.control\_in,

g.rc\_7.control\_in);

//cliSerial->printf\_P(PSTR("OUT 1: %d\t2: %d\t3: %d\t4: %d\n"), (g.rc\_1.servo\_out / 100), (g.rc\_2.servo\_out / 100), g.rc\_3.servo\_out, (g.rc\_4.servo\_out / 100));

/\*cliSerial->printf\_P(PSTR( "min: %d"

\* "\t in: %d"

\* "\t pwm\_in: %d"

\* "\t sout: %d"

\* "\t pwm\_out %d\n"),

\* g.rc\_3.radio\_min,

\* g.rc\_3.control\_in,

\* g.rc\_3.radio\_in,

\* g.rc\_3.servo\_out,

\* g.rc\_3.pwm\_out

\* );

\*/

if(cliSerial->available() > 0) {

return (0);

}

}

}

/\*

\* //static int8\_t

\* //test\_failsafe(uint8\_t argc, const Menu::arg \*argv)

\* {

\*

\* #if THROTTLE\_FAILSAFE

\* byte fail\_test;

\* print\_hit\_enter();

\* for(int16\_t i = 0; i < 50; i++){

\* delay(20);

\* read\_radio();

\* }

\*

\* oldSwitchPosition = readSwitch();

\*

\* cliSerial->printf\_P(PSTR("Unplug battery, throttle in neutral, turn off radio.\n"));

\* while(g.rc\_3.control\_in > 0){

\* delay(20);

\* read\_radio();

\* }

\*

\* while(1){

\* delay(20);

\* read\_radio();

\*

\* if(g.rc\_3.control\_in > 0){

\* cliSerial->printf\_P(PSTR("THROTTLE CHANGED %d \n"), g.rc\_3.control\_in);

\* fail\_test++;

\* }

\*

\* if(oldSwitchPosition != readSwitch()){

\* cliSerial->printf\_P(PSTR("CONTROL MODE CHANGED: "));

\* cliSerial->println(flight\_mode\_strings[readSwitch()]);

\* fail\_test++;

\* }

\*

\* if(g.failsafe\_throttle && g.rc\_3.get\_failsafe()){

\* cliSerial->printf\_P(PSTR("THROTTLE FAILSAFE ACTIVATED: %d, "), g.rc\_3.radio\_in);

\* cliSerial->println(flight\_mode\_strings[readSwitch()]);

\* fail\_test++;

\* }

\*

\* if(fail\_test > 0){

\* return (0);

\* }

\* if(cliSerial->available() > 0){

\* cliSerial->printf\_P(PSTR("LOS caused no change in ACM.\n"));

\* return (0);

\* }

\* }

\* #else

\* return (0);

\* #endif

\* }

\*/

/\*

\* //static int8\_t

\* //test\_stabilize(uint8\_t argc, const Menu::arg \*argv)

\* {

\* static byte ts\_num;

\*

\*

\* print\_hit\_enter();

\* delay(1000);

\*

\* // setup the radio

\* // ---------------

\* init\_rc\_in();

\*

\* control\_mode = STABILIZE;

\* cliSerial->printf\_P(PSTR("g.pi\_stabilize\_roll.kP: %4.4f\n"), g.pi\_stabilize\_roll.kP());

\* cliSerial->printf\_P(PSTR("max\_stabilize\_dampener:%d\n\n "), max\_stabilize\_dampener);

\*

\* motors.auto\_armed(false);

\* motors.armed(true);

\*

\* while(1){

\* // 50 hz

\* if (millis() - fast\_loopTimer > 19) {

\* delta\_ms\_fast\_loop = millis() - fast\_loopTimer;

\* fast\_loopTimer = millis();

\* G\_Dt = (float)delta\_ms\_fast\_loop / 1000.f;

\*

\* if(g.compass\_enabled){

\* medium\_loopCounter++;

\* if(medium\_loopCounter == 5){

\* Matrix3f m = dcm.get\_dcm\_matrix();

\* compass.read(); // Read magnetometer

\* compass.null\_offsets();

\* medium\_loopCounter = 0;

\* }

\* }

\*

\* // for trim features

\* read\_trim\_switch();

\*

\* // Filters radio input - adjust filters in the radio.pde file

\* // ----------------------------------------------------------

\* read\_radio();

\*

\* // IMU

\* // ---

\* read\_AHRS();

\*

\* // allow us to zero out sensors with control switches

\* if(g.rc\_5.control\_in < 600){

\* dcm.roll\_sensor = dcm.pitch\_sensor = 0;

\* }

\*

\* // custom code/exceptions for flight modes

\* // ---------------------------------------

\* update\_current\_flight\_mode();

\*

\* // write out the servo PWM values

\* // ------------------------------

\* set\_servos\_4();

\*

\* ts\_num++;

\* if (ts\_num > 10){

\* ts\_num = 0;

\* cliSerial->printf\_P(PSTR("r: %d, p:%d, rc1:%d, "),

\* (int)(dcm.roll\_sensor/100),

\* (int)(dcm.pitch\_sensor/100),

\* g.rc\_1.pwm\_out);

\*

\* print\_motor\_out();

\* }

\* // R: 1417, L: 1453 F: 1453 B: 1417

\*

\* //cliSerial->printf\_P(PSTR("timer: %d, r: %d\tp: %d\t y: %d\n"), (int)delta\_ms\_fast\_loop, ((int)dcm.roll\_sensor/100), ((int)dcm.pitch\_sensor/100), ((uint16\_t)dcm.yaw\_sensor/100));

\* //cliSerial->printf\_P(PSTR("timer: %d, r: %d\tp: %d\t y: %d\n"), (int)delta\_ms\_fast\_loop, ((int)dcm.roll\_sensor/100), ((int)dcm.pitch\_sensor/100), ((uint16\_t)dcm.yaw\_sensor/100));

\*

\* if(cliSerial->available() > 0){

\* if(g.compass\_enabled){

\* compass.save\_offsets();

\* report\_compass();

\* }

\* return (0);

\* }

\*

\* }

\* }

\* }

\*/

/\*

\* #if HIL\_MODE != HIL\_MODE\_ATTITUDE && CONFIG\_ADC == ENABLED

\* //static int8\_t

\* //test\_adc(uint8\_t argc, const Menu::arg \*argv)

\* {

\* print\_hit\_enter();

\* cliSerial->printf\_P(PSTR("ADC\n"));

\* delay(1000);

\*

\* adc.Init(&timer\_scheduler);

\*

\* delay(50);

\*

\* while(1){

\* for(int16\_t i = 0; i < 9; i++){

\* cliSerial->printf\_P(PSTR("%.1f,"),adc.Ch(i));

\* }

\* cliSerial->println();

\* delay(20);

\* if(cliSerial->available() > 0){

\* return (0);

\* }

\* }

\* }

\* #endif

\*/

static int8\_t

test\_ins(uint8\_t argc, const Menu::arg \*argv)

{

#if defined( \_\_AVR\_ATmega1280\_\_ ) // test disabled to save code size for 1280

print\_test\_disabled();

return (0);

#else

Vector3f gyro, accel;

float temp;

print\_hit\_enter();

cliSerial->printf\_P(PSTR("INS\n"));

delay(1000);

ins.init(AP\_InertialSensor::COLD\_START,

ins\_sample\_rate,

delay, flash\_leds, &timer\_scheduler);

delay(50);

while(1) {

ins.update();

gyro = ins.get\_gyro();

accel = ins.get\_accel();

temp = ins.temperature();

float test = sqrt(sq(accel.x) + sq(accel.y) + sq(accel.z)) / 9.80665;

cliSerial->printf\_P(PSTR("a %7.4f %7.4f %7.4f g %7.4f %7.4f %7.4f t %74f | %7.4f\n"),

accel.x, accel.y, accel.z,

gyro.x, gyro.y, gyro.z,

temp, test);

delay(40);

if(cliSerial->available() > 0) {

return (0);

}

}

#endif

}

static int8\_t

test\_gps(uint8\_t argc, const Menu::arg \*argv)

{

// test disabled to save code size for 1280

#if defined( \_\_AVR\_ATmega1280\_\_ ) || HIL\_MODE != HIL\_MODE\_DISABLED

print\_test\_disabled();

return (0);

#else

print\_hit\_enter();

delay(1000);

while(1) {

delay(333);

// Blink GPS LED if we don't have a fix

// ------------------------------------

update\_GPS\_light();

g\_gps->update();

if (g\_gps->new\_data) {

cliSerial->printf\_P(PSTR("Lat: "));

print\_latlon(&Serial, g\_gps->latitude);

cliSerial->printf\_P(PSTR(", Lon "));

print\_latlon(&Serial, g\_gps->longitude);

cliSerial->printf\_P(PSTR(", Alt: %ldm, #sats: %d\n"),

g\_gps->altitude/100,

g\_gps->num\_sats);

g\_gps->new\_data = false;

}else{

cliSerial->print\_P(PSTR("."));

}

if(cliSerial->available() > 0) {

return (0);

}

}

return 0;

#endif

}

/\*

\* //static int8\_t

\* //test\_dcm(uint8\_t argc, const Menu::arg \*argv)

\* {

\* print\_hit\_enter();

\* delay(1000);

\* cliSerial->printf\_P(PSTR("Gyro | Accel\n"));

\* Vector3f \_cam\_vector;

\* Vector3f \_out\_vector;

\*

\* G\_Dt = .02;

\*

\* while(1){

\* for(byte i = 0; i <= 50; i++){

\* delay(20);

\* // IMU

\* // ---

\* read\_AHRS();

\* }

\*

\* Matrix3f temp = dcm.get\_dcm\_matrix();

\* Matrix3f temp\_t = dcm.get\_dcm\_transposed();

\*

\* cliSerial->printf\_P(PSTR("dcm\n"

\* "%4.4f \t %4.4f \t %4.4f \n"

\* "%4.4f \t %4.4f \t %4.4f \n"

\* "%4.4f \t %4.4f \t %4.4f \n\n"),

\* temp.a.x, temp.a.y, temp.a.z,

\* temp.b.x, temp.b.y, temp.b.z,

\* temp.c.x, temp.c.y, temp.c.z);

\*

\* int16\_t \_pitch = degrees(-asin(temp.c.x));

\* int16\_t \_roll = degrees(atan2(temp.c.y, temp.c.z));

\* int16\_t \_yaw = degrees(atan2(temp.b.x, temp.a.x));

\* cliSerial->printf\_P(PSTR( "angles\n"

\* "%d \t %d \t %d\n\n"),

\* \_pitch,

\* \_roll,

\* \_yaw);

\*

\* //\_out\_vector = \_cam\_vector \* temp;

\* //cliSerial->printf\_P(PSTR( "cam\n"

\* // "%d \t %d \t %d\n\n"),

\* // (int)temp.a.x \* 100, (int)temp.a.y \* 100, (int)temp.a.x \* 100);

\*

\* if(cliSerial->available() > 0){

\* return (0);

\* }

\* }

\* }

\*/

/\*

\* //static int8\_t

\* //test\_dcm(uint8\_t argc, const Menu::arg \*argv)

\* {

\* print\_hit\_enter();

\* delay(1000);

\* cliSerial->printf\_P(PSTR("Gyro | Accel\n"));

\* delay(1000);

\*

\* while(1){

\* Vector3f accels = dcm.get\_accel();

\* cliSerial->print("accels.z:");

\* cliSerial->print(accels.z);

\* cliSerial->print("omega.z:");

\* cliSerial->print(omega.z);

\* delay(100);

\*

\* if(cliSerial->available() > 0){

\* return (0);

\* }

\* }

\* }

\*/

/\*static int8\_t

\* //test\_omega(uint8\_t argc, const Menu::arg \*argv)

\* {

\* static byte ts\_num;

\* float old\_yaw;

\*

\* print\_hit\_enter();

\* delay(1000);

\* cliSerial->printf\_P(PSTR("Omega"));

\* delay(1000);

\*

\* G\_Dt = .02;

\*

\* while(1){

\* delay(20);

\* // IMU

\* // ---

\* read\_AHRS();

\*

\* float my\_oz = (dcm.yaw - old\_yaw) \* 50;

\*

\* old\_yaw = dcm.yaw;

\*

\* ts\_num++;

\* if (ts\_num > 2){

\* ts\_num = 0;

\* //cliSerial->printf\_P(PSTR("R: %4.4f\tP: %4.4f\tY: %4.4f\tY: %4.4f\n"), omega.x, omega.y, omega.z, my\_oz);

\* cliSerial->printf\_P(PSTR(" Yaw: %ld\tY: %4.4f\tY: %4.4f\n"), dcm.yaw\_sensor, omega.z, my\_oz);

\* }

\*

\* if(cliSerial->available() > 0){

\* return (0);

\* }

\* }

\* return (0);

\* }

\* //\*/

static int8\_t

test\_tuning(uint8\_t argc, const Menu::arg \*argv)

{

print\_hit\_enter();

while(1) {

delay(200);

read\_radio();

tuning();

cliSerial->printf\_P(PSTR("tune: %1.3f\n"), tuning\_value);

if(cliSerial->available() > 0) {

return (0);

}

}

}

static int8\_t

test\_battery(uint8\_t argc, const Menu::arg \*argv)

{

#if defined( \_\_AVR\_ATmega1280\_\_ ) // disable this test if we are using 1280

print\_test\_disabled();

return (0);

#else

cliSerial->printf\_P(PSTR("\nCareful! Motors will spin! Press Enter to start.\n"));

cliSerial->flush();

while(!cliSerial->available()) {

delay(100);

}

cliSerial->flush();

print\_hit\_enter();

// allow motors to spin

motors.enable();

motors.armed(true);

while(1) {

delay(100);

read\_radio();

read\_battery();

if (g.battery\_monitoring == 3) {

cliSerial->printf\_P(PSTR("V: %4.4f\n"),

battery\_voltage1,

current\_amps1,

current\_total1);

} else {

cliSerial->printf\_P(PSTR("V: %4.4f, A: %4.4f, Ah: %4.4f\n"),

battery\_voltage1,

current\_amps1,

current\_total1);

}

motors.throttle\_pass\_through();

if(cliSerial->available() > 0) {

motors.armed(false);

return (0);

}

}

motors.armed(false);

return (0);

#endif

}

static int8\_t test\_relay(uint8\_t argc, const Menu::arg \*argv)

{

#if defined( \_\_AVR\_ATmega1280\_\_ ) // test disabled to save code size for 1280

print\_test\_disabled();

return (0);

#else

print\_hit\_enter();

delay(1000);

while(1) {

cliSerial->printf\_P(PSTR("Relay on\n"));

relay.on();

delay(3000);

if(cliSerial->available() > 0) {

return (0);

}

cliSerial->printf\_P(PSTR("Relay off\n"));

relay.off();

delay(3000);

if(cliSerial->available() > 0) {

return (0);

}

}

#endif

}

static int8\_t

test\_wp(uint8\_t argc, const Menu::arg \*argv)

{

delay(1000);

// save the alitude above home option

cliSerial->printf\_P(PSTR("Hold alt "));

if(g.rtl\_altitude < 0) {

cliSerial->printf\_P(PSTR("\n"));

}else{

cliSerial->printf\_P(PSTR("of %dm\n"), (int)g.rtl\_altitude / 100);

}

cliSerial->printf\_P(PSTR("%d wp\n"), (int)g.command\_total);

cliSerial->printf\_P(PSTR("Hit rad: %dm\n"), (int)g.waypoint\_radius);

//cliSerial->printf\_P(PSTR("Loiter radius: %d\n\n"), (int)g.loiter\_radius);

report\_wp();

return (0);

}

//static int8\_t test\_rawgps(uint8\_t argc, const Menu::arg \*argv) {

/\*

\* print\_hit\_enter();

\* delay(1000);

\* while(1){

\* if (Serial3.available()){

\* digitalWrite(B\_LED\_PIN, LED\_ON); // Blink Yellow LED if we are sending data to GPS

\* Serial1.write(Serial3.read());

\* digitalWrite(B\_LED\_PIN, LED\_OFF);

\* }

\* if (Serial1.available()){

\* digitalWrite(C\_LED\_PIN, LED\_ON); // Blink Red LED if we are receiving data from GPS

\* Serial3.write(Serial1.read());

\* digitalWrite(C\_LED\_PIN, LED\_OFF);

\* }

\* if(cliSerial->available() > 0){

\* return (0);

\* }

\* }

\*/

//}

/\*static int8\_t

\* //test\_xbee(uint8\_t argc, const Menu::arg \*argv)

\* {

\* print\_hit\_enter();

\* delay(1000);

\* cliSerial->printf\_P(PSTR("Begin XBee X-CTU Range and RSSI Test:\n"));

\*

\* while(1){

\* if (Serial3.available())

\* Serial3.write(Serial3.read());

\*

\* if(cliSerial->available() > 0){

\* return (0);

\* }

\* }

\* }

\*/

#if HIL\_MODE != HIL\_MODE\_ATTITUDE

static int8\_t

test\_baro(uint8\_t argc, const Menu::arg \*argv)

{

#if defined( \_\_AVR\_ATmega1280\_\_ ) // test disabled to save code size for 1280

print\_test\_disabled();

return (0);

#else

print\_hit\_enter();

init\_barometer();

while(1) {

delay(100);

int32\_t alt = read\_barometer(); // calls barometer.read()

int32\_t pres = barometer.get\_pressure();

int16\_t temp = barometer.get\_temperature();

int32\_t raw\_pres = barometer.get\_raw\_pressure();

int32\_t raw\_temp = barometer.get\_raw\_temp();

cliSerial->printf\_P(PSTR("alt: %ldcm, pres: %ldmbar, temp: %d/100degC,"

" raw pres: %ld, raw temp: %ld\n"),

alt, pres,temp, raw\_pres, raw\_temp);

if(cliSerial->available() > 0) {

return (0);

}

}

return 0;

#endif

}

#endif

static int8\_t

test\_mag(uint8\_t argc, const Menu::arg \*argv)

{

#if defined( \_\_AVR\_ATmega1280\_\_ ) // test disabled to save code size for 1280

print\_test\_disabled();

return (0);

#else

if(g.compass\_enabled) {

print\_hit\_enter();

while(1) {

delay(100);

if (compass.read()) {

float heading = compass.calculate\_heading(ahrs.get\_dcm\_matrix());

cliSerial->printf\_P(PSTR("Heading: %ld, XYZ: %d, %d, %d\n"),

(wrap\_360(ToDeg(heading) \* 100)) /100,

compass.mag\_x,

compass.mag\_y,

compass.mag\_z);

} else {

cliSerial->println\_P(PSTR("not healthy"));

}

if(cliSerial->available() > 0) {

return (0);

}

}

} else {

cliSerial->printf\_P(PSTR("Compass: "));

print\_enabled(false);

return (0);

}

return (0);

#endif

}

/\*

\* //static int8\_t

\* //test\_reverse(uint8\_t argc, const Menu::arg \*argv)

\* {

\* print\_hit\_enter();

\* delay(1000);

\*

\* while(1){

\* delay(20);

\*

\* // Filters radio input - adjust filters in the radio.pde file

\* // ----------------------------------------------------------

\* g.rc\_4.set\_reverse(0);

\* g.rc\_4.set\_pwm(APM\_RC.InputCh(CH\_4));

\* g.rc\_4.servo\_out = g.rc\_4.control\_in;

\* g.rc\_4.calc\_pwm();

\* cliSerial->printf\_P(PSTR("PWM:%d input: %d\toutput%d "),

\* APM\_RC.InputCh(CH\_4),

\* g.rc\_4.control\_in,

\* g.rc\_4.radio\_out);

\* APM\_RC.OutputCh(CH\_6, g.rc\_4.radio\_out);

\*

\*

\* g.rc\_4.set\_reverse(1);

\* g.rc\_4.set\_pwm(APM\_RC.InputCh(CH\_4));

\* g.rc\_4.servo\_out = g.rc\_4.control\_in;

\* g.rc\_4.calc\_pwm();

\* cliSerial->printf\_P(PSTR("\trev input: %d\toutput%d\n"),

\* g.rc\_4.control\_in,

\* g.rc\_4.radio\_out);

\*

\* APM\_RC.OutputCh(CH\_7, g.rc\_4.radio\_out);

\*

\* if(cliSerial->available() > 0){

\* g.rc\_4.set\_reverse(0);

\* return (0);

\* }

\* }

\* }\*/

#if HIL\_MODE != HIL\_MODE\_ATTITUDE

/\*

\* test the sonar

\*/

static int8\_t

test\_sonar(uint8\_t argc, const Menu::arg \*argv)

{

if(g.sonar\_enabled == false) {

cliSerial->printf\_P(PSTR("Sonar disabled\n"));

return (0);

}

// make sure sonar is initialised

init\_sonar();

print\_hit\_enter();

while(1) {

delay(100);

cliSerial->printf\_P(PSTR("Sonar: %d cm\n"), sonar.read());

//cliSerial->printf\_P(PSTR("Sonar, %d, %d\n"), sonar.read(), sonar.raw\_value);

if(cliSerial->available() > 0) {

return (0);

}

}

return (0);

}

#endif

static int8\_t

test\_optflow(uint8\_t argc, const Menu::arg \*argv)

{

#if OPTFLOW == ENABLED

if(g.optflow\_enabled) {

cliSerial->printf\_P(PSTR("man id: %d\t"),optflow.read\_register(ADNS3080\_PRODUCT\_ID));

print\_hit\_enter();

while(1) {

delay(200);

optflow.update(millis());

Log\_Write\_Optflow();

cliSerial->printf\_P(PSTR("x/dx: %d/%d\t y/dy %d/%d\t squal:%d\n"),

optflow.x,

optflow.dx,

optflow.y,

optflow.dy,

optflow.surface\_quality);

if(cliSerial->available() > 0) {

return (0);

}

}

} else {

cliSerial->printf\_P(PSTR("OptFlow: "));

print\_enabled(false);

}

return (0);

#else

print\_test\_disabled();

return (0);

#endif // OPTFLOW == ENABLED

}

static int8\_t

test\_wp\_nav(uint8\_t argc, const Menu::arg \*argv)

{

current\_loc.lat = 389539260;

current\_loc.lng = -1199540200;

next\_WP.lat = 389538528;

next\_WP.lng = -1199541248;

// got 23506;, should be 22800

update\_navigation();

cliSerial->printf\_P(PSTR("bear: %ld\n"), wp\_bearing);

return 0;

}

/\*

\* test the dataflash is working

\*/

static int8\_t

test\_logging(uint8\_t argc, const Menu::arg \*argv)

{

#if defined( \_\_AVR\_ATmega1280\_\_ ) // test disabled to save code size for 1280

print\_test\_disabled();

return (0);

#else

cliSerial->println\_P(PSTR("Testing dataflash logging"));

if (!DataFlash.CardInserted()) {

cliSerial->println\_P(PSTR("ERR: No dataflash inserted"));

return 0;

}

DataFlash.ReadManufacturerID();

cliSerial->printf\_P(PSTR("Manufacturer: 0x%02x Device: 0x%04x\n"),

(unsigned)DataFlash.df\_manufacturer,

(unsigned)DataFlash.df\_device);

cliSerial->printf\_P(PSTR("NumPages: %u PageSize: %u\n"),

(unsigned)DataFlash.df\_NumPages+1,

(unsigned)DataFlash.df\_PageSize);

DataFlash.StartRead(DataFlash.df\_NumPages+1);

cliSerial->printf\_P(PSTR("Format version: %lx Expected format version: %lx\n"),

(unsigned long)DataFlash.ReadLong(), (unsigned long)DF\_LOGGING\_FORMAT);

return 0;

#endif

}

/\*

\* static int8\_t

\* //test\_mission(uint8\_t argc, const Menu::arg \*argv)

\* {

\* //write out a basic mission to the EEPROM

\*

\* //{

\* // uint8\_t id; ///< command id

\* // uint8\_t options; ///< options bitmask (1<<0 = relative altitude)

\* // uint8\_t p1; ///< param 1

\* // int32\_t alt; ///< param 2 - Altitude in centimeters (meters \* 100)

\* // int32\_t lat; ///< param 3 - Lattitude \* 10\*\*7

\* // int32\_t lng; ///< param 4 - Longitude \* 10\*\*7

\* //}

\*

\* // clear home

\* {Location t = {0, 0, 0, 0, 0, 0};

\* set\_cmd\_with\_index(t,0);}

\*

\* // CMD opt pitch alt/cm

\* {Location t = {MAV\_CMD\_NAV\_TAKEOFF, WP\_OPTION\_RELATIVE, 0, 100, 0, 0};

\* set\_cmd\_with\_index(t,1);}

\*

\* if (!strcmp\_P(argv[1].str, PSTR("wp"))) {

\*

\* // CMD opt

\* {Location t = {MAV\_CMD\_NAV\_WAYPOINT, WP\_OPTION\_RELATIVE, 15, 0, 0, 0};

\* set\_cmd\_with\_index(t,2);}

\* // CMD opt

\* {Location t = {MAV\_CMD\_NAV\_RETURN\_TO\_LAUNCH, WP\_OPTION\_YAW, 0, 0, 0, 0};

\* set\_cmd\_with\_index(t,3);}

\*

\* // CMD opt

\* {Location t = {MAV\_CMD\_NAV\_LAND, 0, 0, 0, 0, 0};

\* set\_cmd\_with\_index(t,4);}

\*

\* } else {

\* //2250 = 25 meteres

\* // CMD opt p1 //alt //NS //WE

\* {Location t = {MAV\_CMD\_NAV\_LOITER\_TIME, 0, 10, 0, 0, 0}; // 19

\* set\_cmd\_with\_index(t,2);}

\*

\* // CMD opt dir angle/deg deg/s relative

\* {Location t = {MAV\_CMD\_CONDITION\_YAW, 0, 1, 360, 60, 1};

\* set\_cmd\_with\_index(t,3);}

\*

\* // CMD opt

\* {Location t = {MAV\_CMD\_NAV\_LAND, 0, 0, 0, 0, 0};

\* set\_cmd\_with\_index(t,4);}

\*

\* }

\*

\* g.rtl\_altitude.set\_and\_save(300);

\* g.command\_total.set\_and\_save(4);

\* g.waypoint\_radius.set\_and\_save(3);

\*

\* test\_wp(NULL, NULL);

\* return (0);

\* }

\*/

static void print\_hit\_enter()

{

cliSerial->printf\_P(PSTR("Hit Enter to exit.\n\n"));

}

static void print\_test\_disabled()

{

cliSerial->printf\_P(PSTR("Sorry, not 1280 compat.\n"));

}

/\*

\* //static void fake\_out\_gps()

\* {

\* static float rads;

\* g\_gps->new\_data = true;

\* g\_gps->fix = true;

\*

\* //int length = g.rc\_6.control\_in;

\* rads += .05;

\*

\* if (rads > 6.28){

\* rads = 0;

\* }

\*

\* g\_gps->latitude = 377696000; // Y

\* g\_gps->longitude = -1224319000; // X

\* g\_gps->altitude = 9000; // meters \* 100

\*

\* //next\_WP.lng = home.lng - length \* sin(rads); // X

\* //next\_WP.lat = home.lat + length \* cos(rads); // Y

\* }

\*

\*/

/\*

\* //static void print\_motor\_out(){

\* cliSerial->printf("out: R: %d, L: %d F: %d B: %d\n",

\* (motor\_out[CH\_1] - g.rc\_3.radio\_min),

\* (motor\_out[CH\_2] - g.rc\_3.radio\_min),

\* (motor\_out[CH\_3] - g.rc\_3.radio\_min),

\* (motor\_out[CH\_4] - g.rc\_3.radio\_min));

\* }

\*/

#endif // CLI\_ENABLED

## toy

////////////////////////////////////////////////////////////////////////////////

// Toy Mode - THOR

////////////////////////////////////////////////////////////////////////////////

static boolean CH7\_toy\_flag;

#if TOY\_MIXER == TOY\_LOOKUP\_TABLE

static const int16\_t toy\_lookup[] = {

186, 373, 558, 745,

372, 745, 1117, 1490,

558, 1118, 1675, 2235,

743, 1490, 2233, 2980,

929, 1863, 2792, 3725,

1115, 2235, 3350, 4470,

1301, 2608, 3908, 4500,

1487, 2980, 4467, 4500,

1673, 3353, 4500, 4500

};

#endif

//called at 10hz

void update\_toy\_throttle()

{

/\*

\* // Disabled, now handled by TOY\_A (Alt hold) and TOY\_M (Manual throttle)

\* if (false == CH6\_toy\_flag && g.rc\_6.radio\_in >= CH\_6\_PWM\_TRIGGER){

\* CH6\_toy\_flag = true;

\* throttle\_mode = THROTTLE\_MANUAL;

\*

\* }else if (CH6\_toy\_flag && g.rc\_6.radio\_in < CH\_6\_PWM\_TRIGGER){

\* CH6\_toy\_flag = false;

\* throttle\_mode = THROTTLE\_AUTO;

\* set\_new\_altitude(current\_loc.alt);

\* saved\_toy\_throttle = g.rc\_3.control\_in;

\* }\*/

// look for a change in throttle position to exit throttle hold

if(abs(g.rc\_3.control\_in - saved\_toy\_throttle) > 40) {

throttle\_mode = THROTTLE\_MANUAL;

}

}

#define TOY\_ALT\_SMALL 25

#define TOY\_ALT\_LARGE 100

//called at 10hz

void update\_toy\_altitude()

{

int16\_t input = g.rc\_3.radio\_in; // throttle

//int16\_t input = g.rc\_7.radio\_in;

// Trigger upward alt change

if(false == CH7\_toy\_flag && input > 1666) {

CH7\_toy\_flag = true;

// go up

if(next\_WP.alt >= 400) {

force\_new\_altitude(next\_WP.alt + TOY\_ALT\_LARGE);

}else{

force\_new\_altitude(next\_WP.alt + TOY\_ALT\_SMALL);

}

// Trigger downward alt change

}else if(false == CH7\_toy\_flag && input < 1333) {

CH7\_toy\_flag = true;

// go down

if(next\_WP.alt >= (400 + TOY\_ALT\_LARGE)) {

force\_new\_altitude(next\_WP.alt - TOY\_ALT\_LARGE);

}else if(next\_WP.alt >= TOY\_ALT\_SMALL) {

force\_new\_altitude(next\_WP.alt - TOY\_ALT\_SMALL);

}else if(next\_WP.alt < TOY\_ALT\_SMALL) {

force\_new\_altitude(0);

}

// clear flag

}else if (CH7\_toy\_flag && ((input < 1666) && (input > 1333))) {

CH7\_toy\_flag = false;

}

}

// called at 50 hz from all flight modes

#if TOY\_EDF == ENABLED

void edf\_toy()

{

// EDF control:

g.rc\_8.radio\_out = 1000 + ((abs(g.rc\_2.control\_in) << 1) / 9);

if(g.rc\_8.radio\_out < 1050)

g.rc\_8.radio\_out = 1000;

// output throttle to EDF

if(motors.armed()) {

APM\_RC.OutputCh(CH\_8, g.rc\_8.radio\_out);

}else{

APM\_RC.OutputCh(CH\_8, 1000);

}

// output Servo direction

if(g.rc\_2.control\_in > 0) {

APM\_RC.OutputCh(CH\_6, 1000); // 1000 : 2000

}else{

APM\_RC.OutputCh(CH\_6, 2000); // less than 1450

}

}

#endif

// The function call for managing the flight mode Toy

void roll\_pitch\_toy()

{

#if TOY\_MIXER == TOY\_LOOKUP\_TABLE || TOY\_MIXER == TOY\_LINEAR\_MIXER

int16\_t yaw\_rate = g.rc\_1.control\_in / g.toy\_yaw\_rate;

if(g.rc\_1.control\_in != 0) { // roll

get\_acro\_yaw(yaw\_rate/2);

ap\_system.yaw\_stopped = false;

yaw\_timer = 150;

}else if (!ap\_system.yaw\_stopped) {

get\_acro\_yaw(0);

yaw\_timer--;

if((yaw\_timer == 0) || (fabs(omega.z) < .17)) {

ap\_system.yaw\_stopped = true;

nav\_yaw = ahrs.yaw\_sensor;

}

}else{

if(motors.armed() == false || g.rc\_3.control\_in == 0)

nav\_yaw = ahrs.yaw\_sensor;

get\_stabilize\_yaw(nav\_yaw);

}

#endif

// roll\_rate is the outcome of the linear equation or lookup table

// based on speed and Yaw rate

int16\_t roll\_rate = 0;

#if TOY\_MIXER == TOY\_LOOKUP\_TABLE

uint8\_t xx, yy;

// Lookup value

//xx = g\_gps->ground\_speed / 200;

xx = abs(g.rc\_2.control\_in / 1000);

yy = abs(yaw\_rate / 500);

// constrain to lookup Array range

xx = constrain(xx, 0, 3);

yy = constrain(yy, 0, 8);

roll\_rate = toy\_lookup[yy \* 4 + xx];

if(yaw\_rate == 0) {

roll\_rate = 0;

}else if(yaw\_rate < 0) {

roll\_rate = -roll\_rate;

}

int16\_t roll\_limit = 4500 / g.toy\_yaw\_rate;

roll\_rate = constrain(roll\_rate, -roll\_limit, roll\_limit);

#elif TOY\_MIXER == TOY\_LINEAR\_MIXER

roll\_rate = -((int32\_t)g.rc\_2.control\_in \* (yaw\_rate/100)) /30;

//cliSerial->printf("roll\_rate: %d\n",roll\_rate);

roll\_rate = constrain(roll\_rate, -2000, 2000);

#elif TOY\_MIXER == TOY\_EXTERNAL\_MIXER

// JKR update to allow external roll/yaw mixing

roll\_rate = g.rc\_1.control\_in;

#endif

#if TOY\_EDF == ENABLED

// Output the attitude

//g.rc\_1.servo\_out = get\_stabilize\_roll(roll\_rate);

//g.rc\_2.servo\_out = get\_stabilize\_pitch(g.rc\_6.control\_in); // use CH6 to trim pitch

get\_stabilize\_roll(roll\_rate);

get\_stabilize\_pitch(g.rc\_6.control\_in); // use CH6 to trim pitch

#else

// Output the attitude

//g.rc\_1.servo\_out = get\_stabilize\_roll(roll\_rate);

//g.rc\_2.servo\_out = get\_stabilize\_pitch(g.rc\_2.control\_in);

get\_stabilize\_roll(roll\_rate);

get\_stabilize\_pitch(g.rc\_2.control\_in);

#endif

}