MultiWii GPS assited Position Hold and RTH functions

For code release r33
Document version 1.0

Written By EOSBandi

Table of Contents

This is the latest implementation of GPS assisted position hold and Return to home (RTH) functions for MultiWii. The code that you can get from the link below is based on the MultiWii_dev_20120504 version changes after that interim release are not applied. I hope that eventually this implementation will incorporated into the main trunk and could be part of the 2.1 release. The navigational routines are based on the works of Jason Short and the Arducopter team.

Link : http://code.google.com/p/i2c-gps-nav/downloads/list

Implementations

At the moment, there are two ways to add a GPS to your MultiWii

1. If you have a spare serial port AND an AtMega1280 or 2560 microcontroller based FC, you can connect your GPS to that port and enable the serial GPS code in the code (see howto below) Due the elevated resource usage (RAM and ROM) this method is not recommended with AtMega328 based boards
2. You can use an I2C_GPS board, which contains a secondary AtMega328 processor with a serial GPS and runs the GPS parsing and navigational computations. This board communicates with the FC via the I2C bus.

GPS receivers and their configuration

For a reliable PosHold and RTH you need a precise and fast GPS receiver. I regret to say, but older SiRF III receivers with 1Hz update rate are not suitable. You need an MTK, Ublox or SiRFIV based receiver with at least 5Hz update rate and 115200bps serial speed. The best one has 10Hz refresh rate and a larger patch antenna. The most commonly used receiver is the GTop PA6B module with MTK 3329 chipset . This receiver also used on the currently available I2C nav modules. (Navigatron, RushDuino).

The GPS implementation in MultiWii (both I2c and serial) expect that your GPS receiver is

- set to 10Hz (or 5Hz) update rate
- communicates at 115200 baud speed
- and output GGA, GSA and RMC frames

If these settings are not the default ones, you have to add commands to the initialization part of the MultiWii and i2cgps code to change settings of your GPS receiver . Future versions of the code may use autodetect and autoconfig for different GPS modules, but currently it's up to you. The best solution is to configure your GPS receiver's default settings to match with the above requirements.

The PA6B receivers firmware could be updated to have these default settings. See the MTK-firmware-tools.rar in the download sections. You can follow the description in the MediaTek_programming.pdf file, but use the MTK3329_A1.5E_20110118_10Hz_115200.bin firmware file instead the AX1.30…. one. If you have a i2c-gps board (such as Navigatron) you have to upload some sketch which does not use the serial port. The easiest one is the BLINK example from the Arduino IDE, Select File->Examples->Basics->Blink. Once you uploaded the sketch connect your usb/serial dongle to the board (cross connect TX/RX) and follow the update procedure above . (Latest Navigatron modules reported to preset to 115200/10Hz)

Once you have a new firmware, you can use the MiniGPS program from the update package to test your module, and you can set default parameters (Just press Shift+ctrl+S to display additional tabs). Make sure that your GPS receivers default settings are match with the expected settings above.

Configuring and using Multiwii GPS functions

Setting GPS connection for serial GPS

```
#define GPS_SERIAL 2                    // Sets the serial port used for GPS

#define GPS_BAUD 115200                 // Sets the serial port speed

//#define I2C_GPS                       //Comment out I2C_GPS since using Serial GPS
```

Settings for I2C_GPS

```
//#define GPS_SERIAL 2                  // Comment out serial gps

//#define GPS_BAUD 115200               // Comment out serial gps speed

#define I2C_GPS                         // Enable I2C-GPS
```

Magnetic declination

Magnetic declination is the angle between magnetic north (the direction the north end of a compass needle points) and true north. For a precise navigation we have to know this variance and use it during our calculations. You can get your location's magnetic declination from this website : http://magnetic-declination.com/ The value must be converted to a decimal degree by the degree+minutes*(1/60) formula. For example the magnetic declination for Budapest is 3° 58' EAST so the value that you have to enter into config.h is 3+58*(1/60) which is **3.96.** EAST means that the value is positive, WEST means it's negative (at Pittsburg it is 9° 13' WEST which is **-9.21**). Enter the calculated value to config.h, in the #define MAG_DECLINIATION line, please add an f after the number to indicate a decimal number.

```
#define MAG_DECLINIATION  3.96f
```

Tail control

There are three settings that control the copter's heading during RTH. DO NOT COMMENT OUT THESE DEFINES just change the value true/false.

```
#define NAV_CONTROLS_HEADING        true
```

If this is true then during RTH the copter will always turns its head towards the home point. The heading control is governed by the MAG P value, lower P means the copter will turn slow, higher P equals quick turn. MAGHOLD must be active for this function. (and for all heading control functions)

```
#define NAV_TAIL_FIRST          false
```

If this is true and NAV_CONTROLS_HEADING is true too, then the copter will turn its TAIL toward the home point during RTH.

#define NAV_SET_TAKEOFF_HEADING   true

If this is true, then the copter, once it arrived to home point, will turn its head to the same direction where it was pointed at arming,

Additional GPS settings

GPS Filtering

Since GPS positioning is not 100% precise, there is always a small random error in the coordinates that are given by the GPS receiver. This "noise" can make PosHold twitchy. To overcome this twitch and sudden attitude changes caused by GPS inaccuracy, there is a 5 element moving average filter. Which filters the coordinates that are given by the GPS receiver. Use it only if you have a 10Hz update rate.

To enable GPS filtering, set the #define GPS_FILTERING true

Low speed D term cancelation

During position hold, the control loop's D term can induce errors when copter is moving slowly. To cancel D term when speed is below .5m/s set the #define GPS_LOW_SPEED_F_FILTER true.

Waypoint radius

If the distance between the copter and the active waypoint is less than this value, then the waypoint (home point) is considered reached.

You can set the waypoint radius at #define GPS_WP_RADIUS, the value is in cm.

The following four defines are currently (r33) is in the MultiWii.ino. They will be moved to config.h in the upcoming releases.

Minimum navigational speed

You can set the minimum speed what is used while navigating. This will be the target speed when copter is arrives to the waypoint (home point). The value is in cm/sec

#define NAV_SPEED_MIN   100

Maximum navigational speed

You can set the maximum speed for navigation. The value is in cm/sec. I recommend to lower it to 200cm/sec first, then increase it later.

#define NAV_SPEED_MAX   400

Slow navigation

When the slow navigation is set, then It allows the copter to reduce its speed to zero when approaching the next waypoint.

```
#define NAV_SLOW_NAV            true
```

Maximum allowed banking

This parameter limits the banking output from the navigation routines. By default the maximum banking that can be induced by the navigation routines is 30deg (1deg = 100). For testing, or developing you can decrease it to avoid runaways.

```
#define NAV_BANK_MAX 3000
```

PID controls

As many other flight characteristics, the Position hold and the RTH is controlled by PID control loops.

Position Hold

The location error is calculated in centimeters for X (Longitude) and Y(Latitude) which is fed to a PI controller. The first stage of the controller takes the XY position error and decides how fast the copter should go to reach the correct location.

PI Controler - Poshold

Parameter: POSHOL_P is by default .4 or 30 cm/s for a 1m error. The desired rate maxes out at 150cm/s. This doesn't limit the pitch of the copter, The rate controller will do what it takes to maintain this speed as it approaches the location.

Parameter: POSHOLD_I is used to overcome wind that may be pushing us away from our target. The higher the number the faster the copter will compensate. If your number is too high it will cause oscillations and overshoot.

PID Poshold Rate Control¶

Now that we have a desired rate, we need to change the copter pitch and roll to give us that rate of travel.

Parameter: POSHOLD_RATE_P are the proportional response. Your copter’s optimal setting will depend on the weight and thrust of your engines. If your copter overshoots the target, lower this value.

Parameter: POSHOLD_RATE_I is set to 0 by default. Use this value to maintain tight control of the speed of the copter. If the copter is not achieving the speed it needs, this term will make up the difference by tilting the copter more or less.

Parameter: POSHOLD_RATE_D is the dampening part. If the value is too high you will see small oscillations in pitch or roll. Once the POSHOLD_RATE_P value is dialed in, start at 0, and increment slowly. You should see oscillations die down.

RTH and Navigation
The navigation (RTH) has only one PID control, the desired speed is directly calculated from the distance to the target and the defined speed constrains. The NAV_P, NAV_I and NAV_D parameters have the same functions as the POSHOLD_RATE.

Getting ready for flight

Clearing the EEPROM
It's not a necessary step, but some people ran into weird problems after uploading the code. The root cause of these problems was a corrupted EEPROM content. To avoid such a problem, it's recommended to erase the EEPROM content before upload the new code at the first time.

Start your Arduino IDE, Select File->Examples->EEPROM->eeprom_clear

Upload the sketch and wait. When the status led is on, it's finished. Now you can compile and upload the new code.

Compiling the code
There are some extra steps are needed to get the code compiled in the arduino IDE. The GPS functions are using two libraries for PI and PID controllers. These libraries are in the Arduino-PID-libs folder and you have to copy them into the libraries folder in your arduino folder. If you planning to use i2c_gps, then an extra step is needed. The i2c communication library that comes with arduino needs a patch. You have to take twi.c and twi.h from the Arduino-twi-lib-patch and copy them to the libraries\wire\utilities folder.

First test and tuning

Pre flight tests
Before testing any GPS functions make sure that your copter perfectly working in level mode and your mag is precise. Mag should not be influenced by motor power wires. If necessary try to move your mag away from the power wires. Always make a ground test with a new copter, tie it down, open throttle and check MAG graph in GUI, it should not change when you increase throttle.

During the initial GPS flights I recommend to use HEADFREE mode, in case you lost orientation, or something goes wrong. (Thanks to HEADFREE mode, during the development I was able to bring home my tri, which continuously rotated.)

For GPS functions, you will also need a good GPS reception, and a solid GPS lock. Your GPS receiver must see at least 5 satellites for acceptable precision. There are two visual indicators that show that your GPS acquired lock.

If copter is disarmed, when a solid lock is achieved, the Flight Controller's status led will start blink.

If you use i2c-gps, you can watch the status led on the i2c-board. (this is the blue led on Navigatron).

* 3 short flash at startup
* blinks once per second if GPS is sending data but no position lock yet
* blinks twice fast, then off for a second if GPS 2D position is available
* blinks 3 times fast, then off for a second if GPS 3D position is available
* or goes on for a second, off for a second, (long pulse) if not getting any NEMA sentences for over 5 seconds

In the GUI the gray donut around the heading indicator should blink when GPS (either serial os i2c) is communicating with the Flight Controller.

In GUI poshold and rth box should change color when copter is armed and a solid gps lock is achieved (numSat>=5).

First test Position Hold

Once a solid GPS lock is achieved, home position is set when copter is armed. So plug your battery, wait for GPS lock and fly to your home position (I recommend at least 5/10 meters away from you) land, disarm and arm again. This sets the home position. Copter must be well trimmed and in level mode.

Now takeoff, and try PosHold. It works best if there is minimal horizontal speed when activating. If copter is moving when you activating PosHold, you will experience a couple of swings before it settles. You can try increasing poshold_rate D only by 0.001-0.002 at a time. If hold is not good enough you can increase P terms, also by 0.01-0.02 at a time. If copter swings or circles then decrease it, or increase D.

Once Poshold is ok, then you can try RTH.

Fly away some distance from your home location, and first activate poshold, let the copter settle for a couple of seconds, and then activate RTH. With the default tail control settings your copter should turn towards the home point and start approaching. Once it's arrived it will switch to a poshold and rotate it's head to the same direction as it was when armed. Please note, tail control works only when magHold is enabled.

Additional information

There is an excellent PID tuning simulator (http://diydrones.com/profiles/blogs/ac2-loiter-tuner-sim), which can help to understand the role of the POSHOLD and POSHOLD_RATE settings It's made by Jason Short from DIYDrones who is the genius behind Arducopter loiter code. The simulator is using the same algorithm that the current code use. You cannot take values from the simulator and enter them into your MultiWii, but you can experiment with different settings, and learn what value does what.