Laporan Proyek Akhir

Sistem Mikroprosesor EL3014

Multirotor UAV Flight Controller- Auto Levelling (MUFC-AL)

Kontroller terbang multicopter dengan Arduino

Disusun oleh :

Adi Trisna Nur Wijaya (13214122)

Naufalino Fadel Hutomo (13214138)

Teknik Elektro

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2017

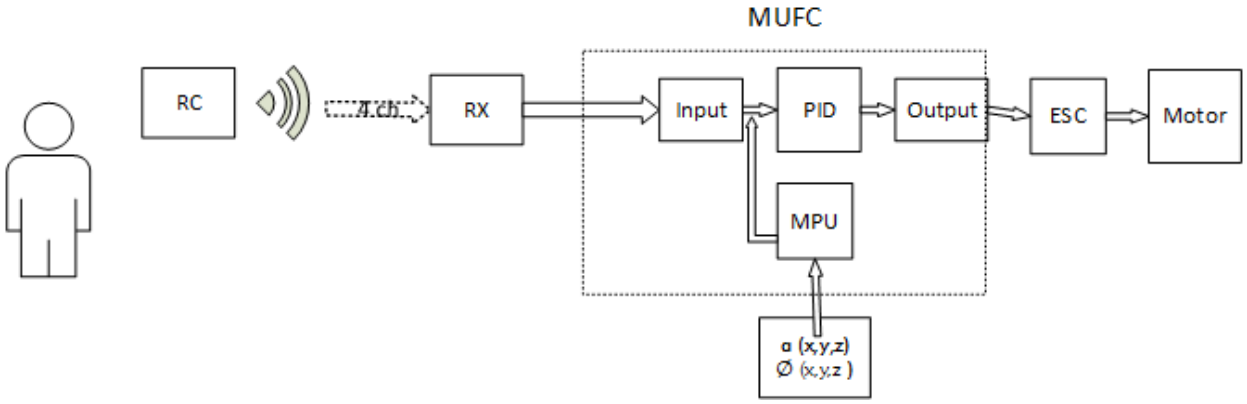
1. Pendahuluan

Saat ini, penggunaan teknologi pesawat tanpa awak (unmanned aerial vehicle atau UAV) sudah sangat marak dilakukan dalam rangka menyelesaikan berbagai macam masalah di berbagai bidang. Dari berbagai masalah tersebut sehingga perlu digunakan pesawat tanpa awak yang canggih. Sebagai contohnya implementasi pesawat tanpa awak yaitu dalam pemetaan suatu daerah. Dari pemetaan tersebut diperlukan pesawat tanpa awak yang stabil dalam suatu ketinggian tertentu. Berdasarkan permasalahan tersebut, sehingga dibuatlah MUFC-AL (Multirotor UAV Flight Controller- Auto Levelling).

2. Spesifikasi Sistem

2.1. Gambaran Umum

MUFC-AL merupakan pengendali pesawat tanpa awak (UAV) multirotor yang *auto-levelling*. Pengendali ini memanfaatkan mikrokontroller untuk mengatur sikap terbang dari UAV multirotor dengan mengkompensasi gangguan saat terbang karena angin, getaran, dan sebagainya serta error antara input dengan output yang dikeluarkan ke motor. Auto levelling merupakan mode terbang pada UAV ini, pada mode ini UAV akan mempertahankan ketinggiannya jika throttle berada di nilai tengah, akan naik saat throttle lebih dari nilai tengah dan akan turun jika lebih kecil. Pada mode ini UAV akan cendereung berada pada kondisi datar (level) pada ketinggian tertentu. Mode terbang ini berbeda dengan mode terbang manual yang input remote mengatur kecepatan motor secara langsung, namun tidak dengan ketinggian wahana. Sehingga pada mode manual diperlukan kompensasi dari pilot langsung untuk mengatur ketinggian dari UAV.



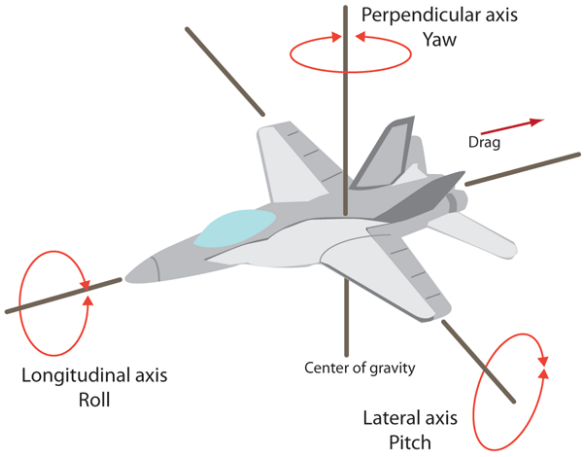
Gambar 1 Diagram Blok

Pesawat tanpa awak ini menerima input PWM dari *receiver*/ RX yang dikendalikan dengan menggunakan *remote control. Remote control* berfungsi sebagai pemeberi sinyal input berupa sinyal PWM pada frekuensi 2.4 GHz ke receiver. Remote control juga melakukan mixing sinyal, yaitu proses pengondisian besaran sinyal menjadi sinyal yang lebih dimengerti dengan keinginan pengguna. Sinyal ini terdiri dari 4 kanal yang mengatur nilai throttle (gerak vertical, naik dan turun), aileron (roll), rudder (yaw), serta elevator (pitch). Sistem control dari alat ini yaitu berupa pengendali PID dengan nilai parameter yang diperoleh dari IMU (Inertia Mesurement Unit). Nilai yang diperoleh berupa data posisi dari MUFC-AL yang kemudian dilakukan pengendalian dengan pengendali PID. Hal ini bertujuan untuk mengkompensasi

gangguan saat terbang menjadi sesuai dengan set point yang diinginkan. Output dari system ini yaitu berupa motor yang dapat menerbangkan alat dengan ketinggian tertentu dan konstan.

Diagram blok yang digunakan dapat dilihat pada gambar 1.

2.2. Derajat Kebebasan UAV

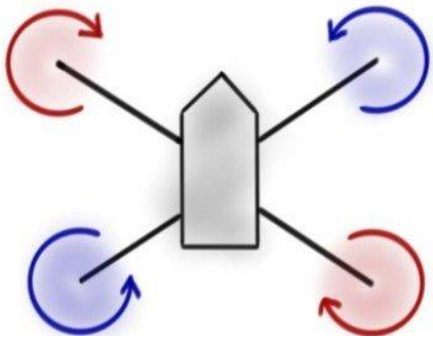


Gambar 2 Roll, Pitch, dan Yaw

UAV multirotor yang dirancang memiliki 6 derajat kebebasan, yaitu 3 derajat untuk gerak lateral atau gerak lurus sumbu x, y, dan z serta 3 derajat gerakan rotasi berupa roll, pitch, dan yaw. Gerakan ini dapat ditimbulkan dari kombinasi 4 channel dari remote control yaitu throttle, elevator, aileron dan rudder. Throttle mengatur gerak naik atau turun (sumbu y), elevator mengatur gerak rotasi pitch, aileron mengatur roll, sementara rudder mengatur yaw.

2.3. Putaran Motor

Untuk menerbangkan MUFC-AL ini diperlukan pengaturan arah putaran dari setiap motor yang digunakan. Setiap motor mempunyai arah putaran yang berbeda-beda untuk menghasilkan momen yang dapat memberikan gaya angkat kepada UAV. Untuk memberikan gaya gerak yang menghasilkan gerakan roll, pitch, dan yaw, kombinasi 2 dari 4 motor yang ada akan diberi kecepatan yang berbeda untuk memberikan gerakan yang diinginkan. Misal untuk melakukan pitch nose up, maka kedua motor di depan akan diatur untuk menjadi lebih cepat sementara motor belakang lebih lambat sehingga ada momen untuk mengangkat bagian depan UAV. Motor depan kanan serta motor kiri belakang berputar counter clockwise, sementara kedua motor lainnya yaitu motor kiri depan dan motor kanan blakang berputar searah clockwise.



Gambar 3 Arah putaran Motor

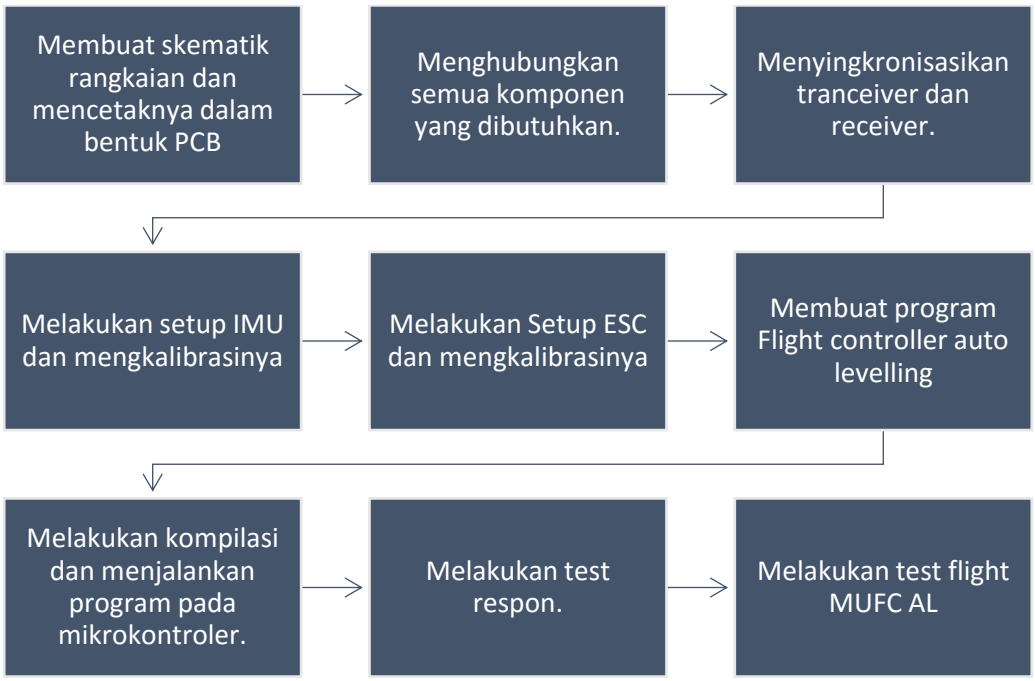
2.4. Komponen Elektrik

Komponen yang digunakan dalam perancangan MUFC-AL sebagai berikut.

1.	Airframe	S500
2.	Battery	3S, 25C
3.	ESC	Simonk max current 30A
4.	Motor	Ready To Sky 920kv
5.	Receiver	Fr Sky X8R
6.	Transmitter	Fr Sky DJT
7.	Remote Control	Turnigy 9XR Pro
8.	Power Module	
9.	PCB	FR4, Masking
10.	Mikroprosesor	Arduino Uno
11.	Propeller	Plastic, DJI 9x4.5
12.	Regulator	LM7808
13.	Dioda	14N001
14.	Header Male & Female	

3. Implementasi dan Hasil

3.1. Metodologi



Gambar 4 Alur Kerja

3.2. Hasil Analisis

3.2.1. Implementasi Mekanik

Pada MUFC-AL ini menggunakan beberapa komponen mekanik sebagai berikut.

1. Airframe

   MUFC-AL menggunakan kerangka airframe S500. Bentuk fisik dari airframe ini yaitu terdapat 4 buah kaki sebagai pijakan untuk lepas landas dan mendarat. Terdapat juga frame untuk menaruh 4 buah motor yang membentuk kotak. Airframe ini terbuat dari carbon fiber sehingga sangat kuat dan cukup dapat meredam getaran yang diakibatkan putaran motor. Airframe ini juga terdapat lapisan PCB untuk mendistribusikan daya dari baterai ke 4 motor yang ada. Keluaran power module disolder ke airframe ini lalu power port ESC juga disambungkan melalui airframe.

Gambar 5 Airframe S500

2. Propeller

   Propeller yang digunakan ialah propeller 9x4,5. Angka tersebut menunjukkan panjang propeller (9 cm) serta 4,5 untuk pitch propeller atau kelengkungan propeller. Propeller terbuat dari bahan plastik. Propeller yang panjang dan pitch yang besar, membutuhkan daya yang lebih besar untuk memutarnya namun juga memberikan gaya angkat yang lebih besar juga. Pemilihan propeller tersebut karena telah cukup untuk mengangkat beban UAV serta cocok dengan putaran motor yang dipakai. Bahan propeller dipilih plastic karena merupakan opsi palng murah serta ringan, walaupun mudah patah jika mengenai halangan.

   Namun saat pengujian terbang, terjadi sedikit kecelakaan sehingga propeller menabrak sisi aspal sehingga propeller patah yang mengakibatkan pengujian terbang saat itu dihentikan.

Gambar 6 Propeller

3.2.2. Implementasi Elektrik

Dalam implementasinya, MUFC-AL terdapat bagian elektrik sebagai berikut.

1. Power Supply

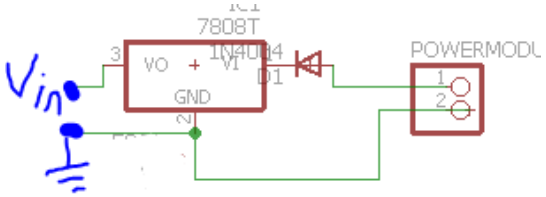
   Pada MUFC-AL diperlukan system power supply yang digunakan untuk menyuplay daya mikrokontroler, remote control, ESC dan motor. Kapasitas power supply yang dibutuhkan dari masing-masing komponen juga berbeda-beda.

   Power supply yang digunakan untuk menyuply daya motor yaitu berupa baterai Li-Po 3S.

Gambar 7 Baterai Li Po 3S untuk Motor

   Jumlah motor yang disupply yaitu sebanyak 4 buah motor yang perlu disupply untuk menjalankannya. Pemilihan baterai menyesuaikan dengan tegangan yang mampu diterima oleh motor, arus maksimal yang dikeluarkan, serta gaya angkat yang akan dihasilkan. Baterai terhubung ke system melalui power module. Power module berfungsi sebagai pengaman untuk memutus rangkaian jika terdeteksi arus berlebih yang bisa merusak komponen maupun baterai.

Power supply yang untuk menyuplai mikrokontroler juga berasal dari baterai Li-Po 3S yang digunakan untuk menyuplai daya motor. Untuk menyesuaikan dengan kebutuhan mikrokontroler maka diperlukan rangkaian yang digunakan untuk mengkonversi ke tegangan yang sesuai. Dalam hal ini digunakan komponen regulator dan diode. Dengan rangkaian sebagai berikut.



Gambar 8 Rangkaian Regulator

Power modul pada rangkaian diatas dihubungkan ke baterai Li-Po 3S. sedangkan Vin dihubungkan ke mikrokontroler.

Power supply yang digunakan untuk menyuplai sinyal ESC dan Receiver berasal dari tegangan keluaran mikrokontroler.

Power supply yang digunakan untuk menyuplai remote control berupa Li-Po 3S dengan gambar sebagai berikut.

Gambar 9 Baterai Li-Po 3S untuk Remote Control

2. Sistem komunikasi (Rx dan Tx)

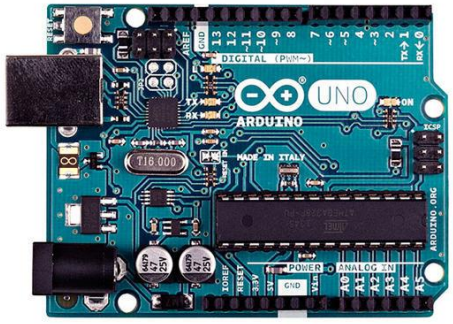
   Sistem komunikasi yaitu dengan menggunakan transmitter yang dihubungkan ke remote control dan receiver yang dihubungkan ke mikrokontroler.

   Remote kontrol berfungsi sebagai perangkat input dari pilot dan juga melakukan mixing sinyal. Pada remote kontrol diatur channel sinyal, limit sinyal, inverse sinyal pada suatu channel. Sinyal yang dimixing akan dikirim melalui transmitter. Transmitter yang digunakan ialah Fr Sky DJT dengan frekuensi 2.4 GHz

   Sinyal akan ditransmisikan secara wireless pada frekuensi 2.4 GHz lalu diterima oleh receiver. Sebelum receiver dan transmitter dapat berkomunikasi, diperlukan proses binding (dijelaskan di bagian pengujian). Sinyal yang diterima receiver berupa sinyal PWM tiap channel. Terdapat antenna baik pada Tx maupun Rx sebagai pemancar dan penerima gelombang.

Gambar 10 Module Rx Fr Sky DJT (kiri) dan Modul Rx Fr Sky X8R

3. Sistem mikroprosesor
   Sistem mikroprosesor yang digunakan yaitu ATMega 328p dalam board Arduino Uno.



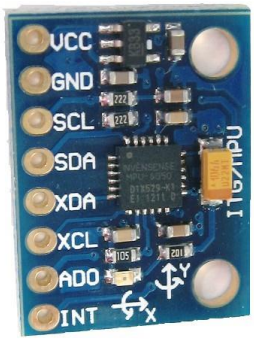
Gambar 11 Arduino Uno dengan mikroprosesor ATMega 328p

4. ESC(Electronics Speed Controller)
   Output dari mikrokontroller yaitu ke ESC yang dapat mengatur kecepatan motor. ESC dapat diatur dengan memberikan sinyal PWM lalu akan memberikan daya yang diinginkan ke motor yang terhubung dengannya. ESC yang digunakan dapat menghantarkan arus hingga 30 A. Batas arus ini sudah melebihi dari batas arus yang dilewati ke motor saat full throttle.
5. Motor
   Output yang menghasilkan gerakan ke wahana yaitu empat buah motor yang dilengkapi propeller sehingga wahana dapat terbang. Motor yang digunakan berupa outrunner ready to sky 920kv.

Gambar 12 Motor beserta ESC

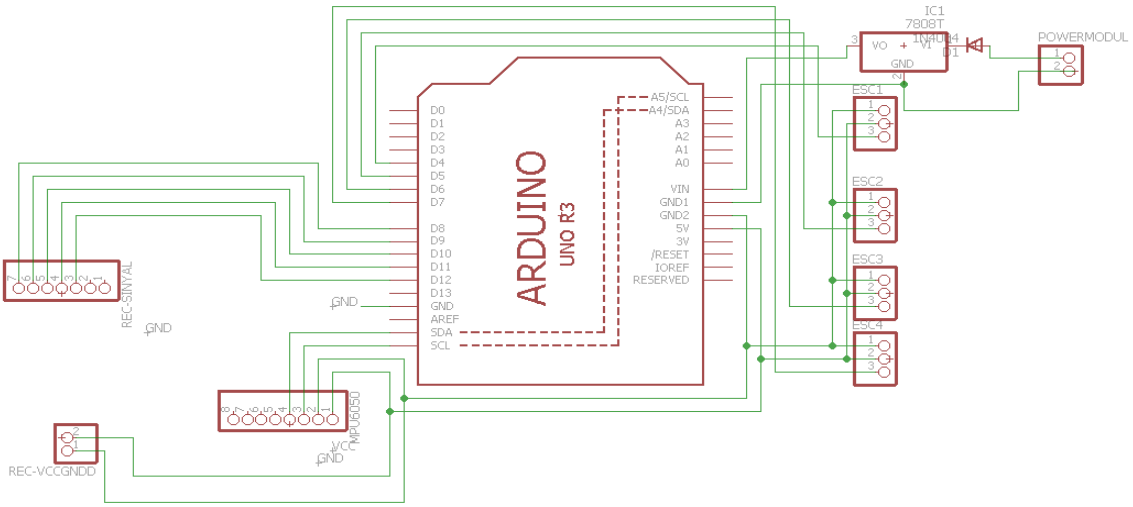
6. IMU

   Inertia measurement unit terdiri dari accelerometer untuk mengetahui percepatan serta gyroscope untuk mengetahui kecepatan sudut system. Pada pengujian ini digunakan IMU jenis MPU 6050 gy 52 yang merupakan MEMS (Micro Electro-mechanical System) dalam satu chip. Percepatan dan kecepatan sudut yang diambil berguna untuk mengkompensasi dan sebagai input pengendali. IMU diatur dengan integrated circuit tersendiri sehingga sinyal yang diterima oleh mikrokontroller sudah merupakan hasil sinyal conditioning dan langsung bisa digunakan. IMU berkomunikasi dengan mikrokontroller secara serial melalui protocol I2C (Inter Integrated Circuit). Untuk mengurangi getaran saat terbang, IMU dilem dengan rangkaian PCB serta mikrokontroller dan rangkaian PCB juga dilem dengan airframe.



Gambar 13 MPU 6050

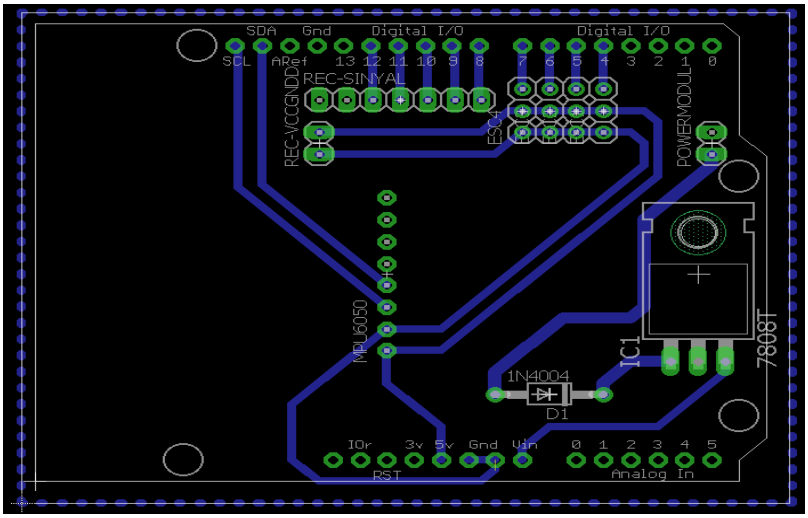
3.2.3. Skematik



Gambar 14 Skematik

Skematik rangkaian yang dibuat dapat dilihat pada gambar di atas. Rangkaian yang dirancang termasuk koneksi ke receiver, koneksi ke ESC untuk mengirim sinyal PWM, koneksi ke IMU, serta voltage regulator untuk memberi daya masuk ke mikrokontroller dari baterai.

Untuk koneksi ke ESC, digunakan pin 4,5,6, dan 7 atau PD4, PD5, PD6, dan PD7. Pin ini dikonfigurasikan sebagai pin output untuk mengirim sinyal PWM. Koneksi dari receiver menggunakan pin 8,9,10,11,dan 12 atau PB0 hingga PB4. Pin yang digunakan hanya 4 pin, kelebihan koneksi pin yaitu ke pin 12 diberikan untuk pengembangan lebih lanjut jika diinginkan penambahan channel suatu hari nanti, namun tidak diperlukan untuk MUFC-AL. Sementara mikrokontroller terhubung dengan IMU melalui pin SDA dan SCL yang dapat bertukar data secara serial dan memberi clock serial kepada IMU. Pin SDA dan SCL ini terhubung secara default ke pin analog 5 dan analog 4 atau PC4 dan PC5. Receiver dan ESC dihubungkan ke mikrokontroller melalui header.



Gambar 15 Desain PCB

Sementara voltage regulator menggunakan IC LM7808 yang memberikan tegangan 8 volt ke pin Vin Arduino. Ditambahkan diode sebagai pengaman bagi regulatornya.

Daya untuk ESC dan motor didapatkan langsung dari baterai melalui power module yang didistribusikan di airframe.

3.3. Hasil Pengujian Dan Analisis

3.3.1. Setup

Pada proses setup ini yaitu bertujuan memastikan bahwa semua alat dan komponen sudah terhubung dan berjalan dengan baik.

Pertama yang dilakukan yaitu melakukan sinkronisasi (binding) transmitter dan receiver. Dilakukan dengan cara menghubungkan pin 7 dan pin 8 receiver. Kemudian menekan dan menahan reset button dari receiver sampai indicator berwarna kuning. Kemudian menyalakan remote yang sudah terhubung dengan Tx, danTx pun akan mencari receiver terdekat dan transmitter dan receiver sudah sinkron. Tx dapat terhubung ke berbagai jenis Rx, sehingga terdapat mode binding pada Tx, untuk proses binding dengan receiver yang dipakai, mode binding ialah '00'. Setelah tx dan rx terhubung, maka dilakukan kalibrasi remote untuk mengetahui nilai tengah, nilai batas atas dan batas bawah serta mengecek apakah nilai channel inverse atau tidak dari masing-masing channel. Setelah proses sampling ini dilakukan, maka sinyal yang diterima dinormalisasi oleh mikrokontroller. Nilai-nilai ini disimpan ke mikrokontroller di dalam EEPROM.

Setelah transceiver sinkron, selanjutnya dilakukan setup IMU. Pertama dilakukan pengecekan alamat IMU untuk mengetahui jenis IMU yang terhubung dengan mikrokontroller. Setelah mengetahui jenis IMU yang terpakai, mikrokontroller akan mengambil alamat gyro lalu menyimpannya ke EEPROM mikrokontroller. Setelah itu dilakukan kalibrasi untuk mengetahui posisi datar dari IMU. Pada kalibrasi ini, letakkan UAV pada tempat yang datar dan pastikan juga IMU terpasang secara mendatar. Setelah kalibrasi, memastikan respon dari IMU dengan memiringkan wahana. Dari proses ini dapat diketahui sumbu-sumbu pada IMU untuk diset ulang serta mengetahui apakah respon terbalik atau tidak. Setelah proses tes respon IMU, maka dilakukan pengesetan kembali sumbu-sumbu IMU dan hasil ini disimpan ke dalam mikrokontroller di dalam EEPROM.

Setup ESC dilakukan dengan mengupload file yang berbeda dengan file setup sebelumnya. Tujuan dari pengaturan setup ESC ini ialah untuk memastikan arah putaran motor, serta memastikan apakah semua motor dapat bergerak secara sinkron. Dalam setup ESC, proses dimulai dengan mengecek apakah remote terkoneksi, input tidak terbalik dan mengkalibrasinya. Mengecek gyro dari hasil register yang menyimpan alamat gyro melalui I2C. Kemudian dilakukan pengecekan koneksi IMU, respon IMU dan melakukan kalibrasi IMU. Kemudian langkah terakhir setup yaitu melakukan pengecekan putaran motor apakah sudah sesuai dari putaran tiap motor dan mensinkron keempat ESC dan motor tanpa kontroler. Setelah semua berhasil dilakukan maka setup selesai.

Proses setup menggunakan Arduino IDE. Hal ini dikarenakan pada proses setup dibutuhkan serial monitor untuk melihat kondisi dari MUFC-AL apakah sudah terkoneksi antar komponennya dan untuk melakukan kalibrasi perlu menggunakan fitur serial monitor pada Arduino IDE.

Gambar 16 Alur Pengujian

3.3.2. Flight Controller-Auto Levelling

Pada proses ini yaitu dimulai dengan membuat program auto levelling dalam Bahasa C. Bahasa C yang sudah selesai dibuat dilakukan kompilasi dan dijalankan. Source code terlampir. Setelah kompilasi, program ini memakan memori program sebesar 11,38 kB dan memori data sebesar 517 bytes.

Penjelasan mengenai isi program flight controller sebagai berikut:

1. Library

Program ini menggunakan beberapa library bawaan dari Win AVR, library dasar C serta library twi.h untuk komunikasi I2C ke MPU. Library twi ini merupakan library dasar di Arduino IDE, sehingga hanya perlu menyalin library ini ke directory program MUFC. Library yang digunakan antara lain <util/delay.h> untuk memberi delay, <avr/eeprom> untuk menulis dan membaca register EEPROM. <avr/io.h> untuk mengatur input output, <avr/interrupt> untuk mengaktifkan dan mengatur interrupt routine, serta <stdlib.h> untuk untuk beberapa operas dalam bahasa C, <string.h> untuk operasi string, <inttypes.h> untuk mengetahui beberapa bentukan tipe data, <math.h> untuk operasi matematika seperti trigonometri, akar, dsb.

2. Konfigurasi Interrupt

Program ini membutuhkan 2 interrupt untuk menghitung waktu dengan timer interrupt serta untuk menerima sinyal input dari receiver dengan external interrupt.

Timer interrupt menggunakan timer0 dengan metode overflow. Pemilihan timer0 disebabkan karena hanya dibutuhkan periode yang sebentar untuk interrupt ini sehingga timer0 yang merupakan timer 8 bit sudah mencukupi keinginan. Timer interrupt ini digunakan untuk mengukur waktu yang telah berjalan semenjak program dinyalakan dan

menyimpan nilai ini pada variable micros, yaitu waktu dalam mikrodetik. Saat interrupt menyala, maka micros akan increment sebanyak 16 us. Periode didapatkan dengan perhitungan $T_{timer} = \frac{2^8}{16MHz} = 16\ \mu s$. Interrupt ini diaktifkan dengan mengatur register TCCR0B untuk mengatur prescaler (prescaler 1), serta register TIMSK untuk mengaktifkan interrupt.

External interrupt dipakai untuk mendapatkan respon yang cepat terhadap perubahan input dari pilot. Interrupt ini diaktifkan dengan mengatur register PCICR (enable scanning PCMSK), PCMSK0 untuk mengaktifkan interrupt. Pin yang digunakan ialah pin pada PORTD 8 hingga PORTD 11 yang merupakan pin PCINT0 hingga PCINT3. Interrupt routine pada bagian ini memberi tanda waktu perubahan sinyal pada masing-masing channel untuk mengetahui nilai PWM yang diterima. Variable waktu ini disimpan dalam variable timer1, timer2, timer3, dan timer4 yang menandakan waktu saat terjadi perubahan change state pada sinyal yang diterima.

3. Konfigurasi setup

Bagian setup hanya dijalankan sekali saja yaitu saat system baru dinyalakan. Bagian ini menjalankan beberapa prosedur penting. Salah satu bagian penting dalam setup ialah konfigurasi PORT sebagai PORT input atau output. Lalu pada setup juga terdapat prosedur pembacaan register EEPROM yang berisikan nilai-nilai parameter system MUFC seperti batas sinyal, kalibrasi MPU, kalibrasi ESC, ke dalam suatu array. Pada bagian ini juga diinisiasi komunikasi I2C dengan gyro, mengatur register gyro dengan hasil kalibrasi sebelumnya melalui fungsi set_gyro_registers(). Setelah itu dilakukan pengecekan nilai offset pada gyro, karena hal ini berbeda-beda di tiap tempat dan tidak cukup jika hanya dilakukan pada kalibrasi sebelumnya. Bagian ini juga terdapat pengaturan arming, disarming serta setup interrupt.

4. Arming dan Disarming

Fungsi arming ialah memberi tanda pada UAV bahwa pilot sudah siap untuk menerbangkan UAV agar lebih aman sehingga UAV tidak langsung terbang jika tiba-tiba remote memberi sinyal input pada saat baru menyalakan UAV. Arming dilakukan dengan memberi nilai throttle minimum dan yaw kiri maksimum. Saat arming telah dilakukan, motor baru akan menerima sinyal dari controller.

Disarming ialah kebalikan dari arming, yaitu memberi tanda bahwa UAV selesai diterbangkan. Disarming dilakukan dengan memeberi nilai minimum pada throttle dan yaw kanan maksimum.

5. Penghitungan Sudut

Penghitungan sudut tidak murni merupakan hasil pengukuran gyro, karen diperlukan beberapa pengolahan terlebih dahulu disebabkan beberapa hal. Pertama dilakukan penyamaan satuan sehingga pengukuran sudut dilakukan dalan satuan yang sama yaitu derajat/s dengan mengalikan dengan konstanta yang didapat dari frekuensi looping dan datasheet MPU 6050. Lalu juga ditambahkan efek yaw yang dapat memberikan sudut tambahan pada roll atau pitch jika yaw dilakukan pada posisi yang tidak mendatar. Hal

ini melibatkan penghitungan geometri secara sederhana. Sudut juga mempertimbangkan nilai percepatan yang terukur pada accelerometer.

UAV ini dijalankan pada mode terbang auto level, sehingga terdapat level adjust sebagai koreksi pada gerakan roll dan pitch.

6. Penghitungan PID

Pada bagian ini, program melakukan penghitungan PID untuk masing-masing roll, pitch dan yaw. Throtlle tidak dikendalikan PID karena dalam mode auto level, sehingga set point throttle bukan dari sinyal throttle dari remote control. Nilai konstanta PID diatur sembarang dan lalu dilakukan koreksi berdasar sikap terbang yang diamati, pada percobaan pertama kami menggunakan konstanta PID berdasarkan referensi yang didapat. Nilai setpoint ialah nilai sinyal yang diterima receiver dari transmitter, sementara feedback berdasar dari nilai sudut yang telah dihitung sebelumnya.

7. Sinyal PWM ke ESC

Sinyal PWM yang diberikan disimpan pada variable esc1, esc2, esc3, dan esc4 untuk masing-masing esc yang terhubung ke motor. Nilai ini bergantung pada hasil perhitungan PID lalu akan dilakukan operasi untuk memberikan sinyal spesifik terhadap satu motor tertentu tergantung dengan kombinasi 4 channel yang ada yaitu throttle, roll, pitch, dan yaw. Untuk throttle, karena dalam auto level, maka nilai throttle yang diberikan dibatasi yaitu maksimum 1800 sehingga UAV tidak bisa mengangkat secara drastic. Sinyal juga dibatasi untuk masing-masing motor agar tidak terjadi gerakan yang terlalu ekstrem.

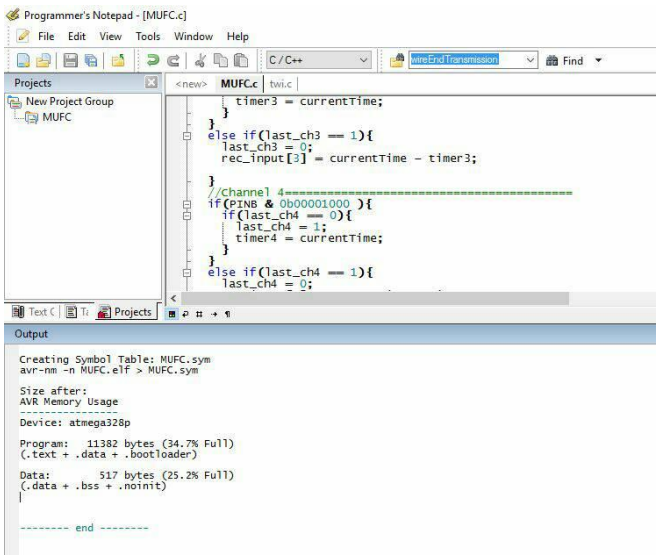
8. Set gyro Registers dan gyrolen

Pada bagian set gyro registers mikrokontroller berkomunikasi untuk mengatur nilai register pada MPU melalui protocol I2C dan memanfaatkan fungsi dengan awalan wire. Pertama mengambil nilai dari EEEPROM lalu mengatur register PWR_MGMT pada MPU untuk mengaktifkan gyroscope. Lalu juga dilakukan pengaturan nilai register GYRO_Config untuk mengatur sampling, ACCEL_Config untuk mengatur skala accelerometer dan register Config.

Sementara pada bagian gyrolen, dilakukan pembacaan nilai yang terukur dari gyro dan accelerometer dan menyimpannya dalam variable. Variable ini juga menyesuaikan dengan kalibrasi yang telah dilakukan sebelumnya pada setup apakah sinyal invert atau tidak, sehingga jika sinyal dalam kondisi invert, variable akan mengkompensasinya lagi. Bisa disimpulkan kompensasi sinyal yang invert dilakukan secara software pada program ini.

9. Wire

Rumpun fungsi wire yaitu wireWrite, wireRead, wireBeginTransmission, wireEndTransmission, wireBegin, wireRequestFrom, dan wireAvailable merupakan fungsi-fungsi yang menjalankan perintah komunikasi I2C untuk mengakses data, menulis data, mengecek, serta konfigurasi untuk komunikasi I2C antara mikrokontroller dengan MPU 6050.



Gambar 17 Hasil kompilasi program MUFC

3.3.3. **Tes Respon**

Setelah itu dilakukan tes respon MUFC-AL untuk melihat apakah program sudah berjalan dengan baik atau belum. Tes respon yaitu dengan cara memegang MUFC-AL dengan ditahan oleh kedua tangan dan diletakkan di atas kepala. Kemudian dilakukan tes respon dengan memberikan sinyal input melalui remote control dan melihat apakah respon yang diberikan telah sesuai dengan masukan yang diinginkan atau belum. Jika tes respon telah berhasil, maka wahana siap untuk uji terbang. Jika belum lulus tes respon, maka perlu dilakukan perbaikan pada wahana, perbaikan bisa melalui software dengan upload program ulang, atau melalui remote control.

Tes respon meliputi throttle (lepas landas dan mendarat) yaitu mengecek apakah MUFC-AL dapat naik dan turun. Hasil yang diperoleh tes throttle berjalan dengan baik. Kemudian dilakukan beberapa tes respon lainnya seperti roll ke kanan dan kekiri, pitch ke atas dan bawah, yaw ke kanan dan kiri. Dari hasil tes respon MUFC-AL dapat berjalan dengan baik untuk semua gerakan. Namun terdeteksi ada sinyal yang terbalik untuk gerakan pitch pada tes respon pertama. Ini bisa dikompensasi dengan mengatur mixing sinyal melalui remote control.

3.3.4. **Tes Terbang**

Setelah dilakukan tes respons dan berhasil, kemudian dilakukan tes terbang. Saat dilakukan tes terbang, MUFC-AL menabrak tembok dan propeller patah sehingga tidak Tes terbang dilakukan pertama kali pada tanggal 18 Mei 2017 di kawasan Monumen Perjuangan Bandung. Pada tes terbang pertama ini, UAV berhasil terbang dan mengangkat. Wahana juga terlihat telah melakukan fitur auto level dengan baik. Saat throttle di bawah nilai tengah, maka UAV cenderung mempertahankan ketiggiannya, sementara jika melebihi nilai tengah, UAV baru akan mengangkat. Saat diberikan roll dan pitch, UAV juga mengkompensasi gerakan tersebut layaknya power steering pada mobil, yaitu seelah roll atau pitch dikembalikan ke nilai tengah, maka UAV akan merespon dengan memebri roll atau pitch kebalikan hingga AUV berada di posisi mendatar lagi.

Beberapa saat kemudian, akibat angin dan kurang berpengalamannya pilot dalam mengendalikan UAV, orientasi UAV miring dan karena tidak dikompensasi dengan cepat oleh pilot, propeller tertahan oleh pembatas jalan. Hal ini menyebabkan propeller patah dan bengkok. Akibat hal ini uji terbang dihentikan dan diperlukan propeller baru karena akan membahayakan jika dilanjutkan karena tidak berimbangnya propeller pada semua motor. Namun secara keseluruhan MUFC-AL sudah berjalan dengan baik.

4. Kesimpulan

Dari proses perancangan, pengujian serta analisis yang dilakukan, diperoleh beberapa kesimpulan sebagai berikut.

1. MUFC-AL ialah pengendali terbang multirotor quadcopter auto level. Auto level yaitu wahana cenderung mempertahankan ketinggiannya.
2. Pengendali menggunakan implementasi PID digital pada mikrokontroller.
3. MUFC-AL hanya mengatur pengendali terbang berupa perhitungan PID, pemberian sinyal, pemrosesan sinyal input dari receiver, serta memroses pengukuran IMU untuk dijadikan feedback pada pengendali. Sementara transmisi sinyal, mixing sinyal, pengendali kecepatan motor diatur komponen lain seperti receiver, transmitter, remote control dan ESC.
4. Pengendali terbang dapat diimplementasikan secara sederhana dengan Bahasa C dan deprogram ke mikroprosessor ATMega.
5. Diperlukan kalibrasi sebelum dilakukan pengujian terbang untuk memastikan standar lingkungan saat itu. Kalibrasi berupa kalibrasi koneksi rx dan tx, sinyal input per channel, batas sinyal input, level atau permukaan datar, sumbu gyro dan accelerometer, ESC, serta putaran motor.
6. Dari hasil implementasi, mikrokontroller telah berfungsi dengan baik mulai dari proses pengaturan, kalibrasi, hingga pengujian. UAV dapat merespon dan terbang dengan cukup baik dan memuaskan.

Daftar Pustaka

www.brokking.net/

http://www.atmel.com/webdoc/avrlibcreferencemanual

http://www.avrfreaks.net/forum/

Sumber Gambar

http://www.machinedesign.com/

https://www.quora.com/Why-cant-all-the-motors-of-a-quadcopter-rotate-in-the-same-direction

http://playground.arduino.cc/Main/MPU-6050

Lampiran

Dokumentasi

Video : tinyurl.com/mufcAutoLevelling

Foto

Gambar 18 Airframe yang telah dipasang system lengkap

Gambar 19 PCB dengan MPU yang dilem (kiri). Pemasangan PCB ke Arduino Board (kanan)

Gambar 20 Propeller yang patah saat tes terbang

Source code yang digunakan.

Setup.ino

```
#include <Wire.h>    //Include the Wire.h library so we can communicate with the gyro
#include <EEPROM.h>//include EEPROM.h so we can store information onto the EEPROM

//Declaring Global Variables
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte lowByte, highByte, type, gyro_address, error, clockspeed_ok;
byte channel_1_assign, channel_2_assign, channel_3_assign, channel_4_assign;
byte roll_axis, pitch_axis, yaw_axis;
byte receiver_check_byte, gyro_check_byte;
volatile        int        receiver_input_channel_1,        receiver_input_channel_2,
receiver_input_channel_3, receiver_input_channel_4;
int center_channel_1, center_channel_2, center_channel_3, center_channel_4;
int high_channel_1, high_channel_2, high_channel_3, high_channel_4;
int low_channel_1, low_channel_2, low_channel_3, low_channel_4;
int address, cal_int;
unsigned long timer, timer_1, timer_2, timer_3, timer_4, current_time;
float gyro_pitch, gyro_roll, gyro_yaw;
float gyro_roll_cal, gyro_pitch_cal, gyro_yaw_cal;

//function declaration
void check_receiver_inputs(byte movement);
void register_min_max();
void check_gyro_axes(byte movement);
void check_to_continue();
void gyro_signalen();
void start_gyro();
byte search_gyro(int gyro_address, int who_am_i);
void wait_sticks_zero();

//Setup routine
void setup(){
  pinMode(12, OUTPUT);
    PCICR |= (1 << PCIE0);    // set PCIE0 to enable PCMSK0 scan
  PCMSK0 |= (1 << PCINT0);  // set PCINT0 (digital input 8) to trigger an interrupt
on state change
  PCMSK0 |= (1 << PCINT1);  // set PCINT1 (digital input 9)to trigger an interrupt on
state change
  PCMSK0 |= (1 << PCINT2);  // set PCINT2 (digital input 10)to trigger an interrupt
on state change
  PCMSK0 |= (1 << PCINT3);  // set PCINT3 (digital input 11)to trigger an interrupt
on state change
  Wire.begin();             //Start the I2C as master
  Serial.begin(57600);      //Start the serial connetion @ 57600bps
  delay(250);               //Give the gyro time to start
}
//Main program
void loop(){
  //Show the YMFC-3D V2 intro
//  intro();

  Serial.println(F(""));
```

```
  Serial.println(F("====================================================="));
  Serial.println(F("System check"));
  Serial.println(F("====================================================="));
  delay(1000);
  Serial.println(F("Checking I2C clock speed."));
  delay(1000);

  TWBR = 12;                        //Set the I2C clock speed to 400kHz.

  #if F_CPU == 16000000L           //If the clock speed is 16MHz include the next code
line when compiling
    clockspeed_ok = 1;             //Set clockspeed_ok to 1
  #endif                           //End of if statement

  if(TWBR == 12 && clockspeed_ok){
    Serial.println(F("I2C clock speed is correctly set to 400kHz."));
  }
  else{
    Serial.println(F("I2C clock speed is not set to 400kHz. (ERROR 8)"));
    error = 1;
  }

  if(error == 0){
    Serial.println(F(""));
    Serial.println(F("====================================================="));
    Serial.println(F("Transmitter setup"));
    Serial.println(F("====================================================="));
    delay(1000);
    Serial.print(F("Checking for valid receiver signals."));
    //Wait 10 seconds until all receiver inputs are valid
   // wait_for_receiver();
    Serial.println(F(""));
  }
  //Quit the program in case of an error
  if(error == 0){
    delay(2000);
    Serial.println(F("Place all sticks and subtrims in the center position within 10
seconds."));
    for(int i = 9;i > 0;i--){
      delay(1000);
      Serial.print(i);
      Serial.print(" ");
    }
    Serial.println(" ");
    //Store the central stick positions
    center_channel_1 = receiver_input_channel_1;
    center_channel_2 = receiver_input_channel_2;
    center_channel_3 = receiver_input_channel_3;
    center_channel_4 = receiver_input_channel_4;
    Serial.println(F(""));
    Serial.println(F("Center positions stored."));
    Serial.print(F("Digital input 08 = "));
    Serial.println(receiver_input_channel_1);
    Serial.print(F("Digital input 09 = "));
    Serial.println(receiver_input_channel_2);
```

```
    Serial.print(F("Digital input 10 = "));
    Serial.println(receiver_input_channel_3);
    Serial.print(F("Digital input 11 = "));
    Serial.println(receiver_input_channel_4);
    Serial.println(F(""));
    Serial.println(F(""));
  }
  if(error == 0){
    Serial.println(F("Move the throttle stick to full throttle and back to center"));
    //Check for throttle movement
    check_receiver_inputs(1);
    Serial.print(F("Throttle is connected to digital input "));
    Serial.println((channel_3_assign & 0b00000111) + 7);
    if(channel_3_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();

    Serial.println(F(""));
    Serial.println(F(""));
    Serial.println(F("Move the roll stick to simulate left wing up and back to
center"));
    //Check for throttle movement
    check_receiver_inputs(2);
    Serial.print(F("Roll is connected to digital input "));
    Serial.println((channel_1_assign & 0b00000111) + 7);
    if(channel_1_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();
  }
  if(error == 0){
    Serial.println(F(""));
    Serial.println(F(""));
    Serial.println(F("Move the pitch stick to simulate nose up and back to center"));
    //Check for throttle movement
    check_receiver_inputs(3);
    Serial.print(F("Pitch is connected to digital input "));
    Serial.println((channel_2_assign & 0b00000111) + 7);
    if(channel_2_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();
  }
  if(error == 0){
    Serial.println(F(""));
    Serial.println(F(""));
    Serial.println(F("Move the yaw stick to simulate nose right and back to center"));
    //Check for throttle movement
    check_receiver_inputs(4);
    Serial.print(F("Yaw is connected to digital input "));
    Serial.println((channel_4_assign & 0b00000111) + 7);
    if(channel_4_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();
  }
  if(error == 0){
    Serial.println(F(""));
```

```
    Serial.println(F(""));
    Serial.println(F("Gently move all the sticks simultaneously to their extends"));
    Serial.println(F("When ready put the sticks back in their center positions"));
    //Register the min and max values of the receiver channels
    register_min_max();
    Serial.println(F(""));
    Serial.println(F(""));
    Serial.println(F("High, low and center values found during setup"));
    Serial.print(F("Digital input 08 values:"));
    Serial.print(low_channel_1);
    Serial.print(F(" - "));
    Serial.print(center_channel_1);
    Serial.print(F(" - "));
    Serial.println(high_channel_1);
    Serial.print(F("Digital input 09 values:"));
    Serial.print(low_channel_2);
    Serial.print(F(" - "));
    Serial.print(center_channel_2);
    Serial.print(F(" - "));
    Serial.println(high_channel_2);
    Serial.print(F("Digital input 10 values:"));
    Serial.print(low_channel_3);
    Serial.print(F(" - "));
    Serial.print(center_channel_3);
    Serial.print(F(" - "));
    Serial.println(high_channel_3);
    Serial.print(F("Digital input 11 values:"));
    Serial.print(low_channel_4);
    Serial.print(F(" - "));
    Serial.print(center_channel_4);
    Serial.print(F(" - "));
    Serial.println(high_channel_4);
    Serial.println(F("Move stick 'nose up' and back to center to continue"));
    check_to_continue();
  }

    if(error == 0){
    //What gyro is connected
    Serial.println(F(""));
    Serial.println(F("======================================================"));
    Serial.println(F("Gyro search"));
    Serial.println(F("======================================================"));
    delay(2000);

    Serial.println(F("Searching for MPU-6050 on address 0x68/104"));
    delay(1000);
    if(search_gyro(0x68, 0x75) == 0x68){
      Serial.println(F("MPU-6050 found on address 0x68"));
      type = 1;
      gyro_address = 0x68;
    }

    if(type == 0){
      Serial.println(F("Searching for MPU-6050 on address 0x69/105"));
      delay(1000);
```

```
    if(search_gyro(0x69, 0x75) == 0x68){
      Serial.println(F("MPU-6050 found on address 0x69"));
      type = 1;
      gyro_address = 0x69;
    }
  }

  if(type == 0){
    Serial.println(F("Searching for L3G4200D on address 0x68/104"));
    delay(1000);
    if(search_gyro(0x68, 0x0F) == 0xD3){
      Serial.println(F("L3G4200D found on address 0x68"));
      type = 2;
      gyro_address = 0x68;
    }
  }

  if(type == 0){
    Serial.println(F("Searching for L3G4200D on address 0x69/105"));
    delay(1000);
    if(search_gyro(0x69, 0x0F) == 0xD3){
      Serial.println(F("L3G4200D found on address 0x69"));
      type = 2;
      gyro_address = 0x69;
    }
  }

  if(type == 0){
    Serial.println(F("Searching for L3GD20H on address 0x6A/106"));
    delay(1000);
    if(search_gyro(0x6A, 0x0F) == 0xD7){
      Serial.println(F("L3GD20H found on address 0x6A"));
      type = 3;
      gyro_address = 0x6A;
    }
  }

  if(type == 0){
   Serial.println(F("Searching for L3GD20H on address 0x6B/107"));
    delay(1000);
    if(search_gyro(0x6B, 0x0F) == 0xD7){
      Serial.println(F("L3GD20H found on address 0x6B"));
      type = 3;
      gyro_address = 0x6B;
    }
  }

  if(type == 0){
    Serial.println(F("No gyro device found!!! (ERROR 3)"));
    error = 1;
  }

  else{
    delay(3000);
    Serial.println(F(""));
```

```
      Serial.println(F("====================================================="));
      Serial.println(F("Gyro register settings"));
      Serial.println(F("====================================================="));
      start_gyro(); //Setup the gyro for further use
    }
   }

  //If the gyro is found we can setup the correct gyro axes.
  if(error == 0){
    delay(3000);
    Serial.println(F(""));
    Serial.println(F("====================================================="));
    Serial.println(F("Gyro calibration"));
    Serial.println(F("====================================================="));
    Serial.println(F("Don't move the quadcopter!! Calibration starts in 3 seconds"));
    delay(3000);
    Serial.println(F("Calibrating the gyro, this will take +/- 8 seconds"));
    Serial.print(F("Please wait"));
    //Let's take multiple gyro data samples so we can determine the average gyro offset
(calibration).
    for (cal_int = 0; cal_int < 2000 ; cal_int ++){              //Take 2000 readings
for calibration.
      if(cal_int % 100 == 0)Serial.print(F("."));                    //Print dot to
indicate calibration.
      gyro_signalen();                                             //Read the gyro
output.
      gyro_roll_cal += gyro_roll;                                  //Ad roll value to
gyro_roll_cal.
      gyro_pitch_cal += gyro_pitch;                               //Ad pitch value to
gyro_pitch_cal.
      gyro_yaw_cal += gyro_yaw;                                    //Ad yaw value to
gyro_yaw_cal.
      delay(4);                                                   //Wait 3 milliseconds
before the next loop.
    }
    //Now that we have 2000 measures, we need to devide by 2000 to get the average
gyro offset.
    gyro_roll_cal /= 2000;                                         //Divide the roll
total by 2000.
    gyro_pitch_cal /= 2000;                                       //Divide the pitch
total by 2000.
    gyro_yaw_cal /= 2000;                                          //Divide the yaw
total by 2000.

    //Show the calibration results
    Serial.println(F(""));
    Serial.print(F("Axis 1 offset="));
    Serial.println(gyro_roll_cal);
    Serial.print(F("Axis 2 offset="));
    Serial.println(gyro_pitch_cal);
    Serial.print(F("Axis 3 offset="));
    Serial.println(gyro_yaw_cal);
    Serial.println(F(""));

    Serial.println(F("====================================================="));
```

```
    Serial.println(F("Gyro axes configuration"));
    Serial.println(F("================================================="));

    //Detect the left wing up movement
    Serial.println(F("Lift the left side of the quadcopter to a 45 degree angle within
10 seconds"));
    //Check axis movement
    check_gyro_axes(1);
    if(error == 0){
      Serial.println(F("OK!"));
      Serial.print(F("Angle detection = "));
      Serial.println(roll_axis & 0b00000011);
      if(roll_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));
      else Serial.println(F("Axis inverted = no"));
      Serial.println(F("Put the quadcopter back in its original position"));
      Serial.println(F("Move stick 'nose up' and back to center to continue"));
      check_to_continue();

      //Detect the nose up movement
      Serial.println(F(""));
      Serial.println(F(""));
      Serial.println(F("Lift the nose of the quadcopter to a 45 degree angle within
10 seconds"));
      //Check axis movement
      check_gyro_axes(2);
    }
    if(error == 0){
      Serial.println(F("OK!"));
      Serial.print(F("Angle detection = "));
      Serial.println(pitch_axis & 0b00000011);
      if(pitch_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));
      else Serial.println(F("Axis inverted = no"));
      Serial.println(F("Put the quadcopter back in its original position"));
      Serial.println(F("Move stick 'nose up' and back to center to continue"));
      check_to_continue();

      //Detect the nose right movement
      Serial.println(F(""));
      Serial.println(F(""));
      Serial.println(F("Rotate the nose of the quadcopter 45 degree to the right within
10 seconds"));
      //Check axis movement
      check_gyro_axes(3);
    }
    if(error == 0){
      Serial.println(F("OK!"));
      Serial.print(F("Angle detection = "));
      Serial.println(yaw_axis & 0b00000011);
      if(yaw_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));
      else Serial.println(F("Axis inverted = no"));
      Serial.println(F("Put the quadcopter back in its original position"));
      Serial.println(F("Move stick 'nose up' and back to center to continue"));
      check_to_continue();
    }
  }
```

```
  if(error == 0){
    Serial.println(F(""));
    Serial.println(F("======================================================"));
    Serial.println(F("LED test"));
    Serial.println(F("======================================================"));
    digitalWrite(12, HIGH);
    Serial.println(F("The LED should now be lit"));
    Serial.println(F("Move stick 'nose up' and back to center to continue"));
    check_to_continue();
    digitalWrite(12, LOW);
  }

  Serial.println(F(""));

  if(error == 0){
    Serial.println(F("======================================================"));
    Serial.println(F("Final setup check"));
    Serial.println(F("======================================================"));
    delay(1000);
    if(receiver_check_byte == 0b00001111){
      Serial.println(F("Receiver channels ok"));
    }
    else{
      Serial.println(F("Receiver channel verification failed!!! (ERROR 6)"));
      error = 1;
    }
    delay(1000);
    /*
    if(gyro_check_byte == 0b00000111){
      Serial.println(F("Gyro axes ok"));
    }
    else{
      Serial.println(F("Gyro exes verification failed!!! (ERROR 7)"));
      error = 1;
    }
    */
  }

  if(error == 0){
    //If all is good, store the information in the EEPROM
    Serial.println(F(""));
    Serial.println(F("======================================================"));
    Serial.println(F("Storing EEPROM information"));
    Serial.println(F("======================================================"));
    Serial.println(F("Writing EEPROM"));
    delay(1000);
    Serial.println(F("Done!"));
    EEPROM.write(0, center_channel_1 & 0b11111111);
    EEPROM.write(1, center_channel_1 >> 8);
    EEPROM.write(2, center_channel_2 & 0b11111111);
    EEPROM.write(3, center_channel_2 >> 8);
    EEPROM.write(4, center_channel_3 & 0b11111111);
    EEPROM.write(5, center_channel_3 >> 8);
    EEPROM.write(6, center_channel_4 & 0b11111111);
```

```
    EEPROM.write(7, center_channel_4 >> 8);
    EEPROM.write(8, high_channel_1 & 0b11111111);
    EEPROM.write(9, high_channel_1 >> 8);
    EEPROM.write(10, high_channel_2 & 0b11111111);
    EEPROM.write(11, high_channel_2 >> 8);
    EEPROM.write(12, high_channel_3 & 0b11111111);
    EEPROM.write(13, high_channel_3 >> 8);
    EEPROM.write(14, high_channel_4 & 0b11111111);
    EEPROM.write(15, high_channel_4 >> 8);
    EEPROM.write(16, low_channel_1 & 0b11111111);
    EEPROM.write(17, low_channel_1 >> 8);
    EEPROM.write(18, low_channel_2 & 0b11111111);
    EEPROM.write(19, low_channel_2 >> 8);
    EEPROM.write(20, low_channel_3 & 0b11111111);
    EEPROM.write(21, low_channel_3 >> 8);
    EEPROM.write(22, low_channel_4 & 0b11111111);
    EEPROM.write(23, low_channel_4 >> 8);
    EEPROM.write(24, channel_1_assign);
    EEPROM.write(25, channel_2_assign);
    EEPROM.write(26, channel_3_assign);
    EEPROM.write(27, channel_4_assign);
    EEPROM.write(28, roll_axis);
    EEPROM.write(29, pitch_axis);
    EEPROM.write(30, yaw_axis);
    EEPROM.write(31, type);
    EEPROM.write(32, gyro_address);
    //Write the EEPROM signature
    EEPROM.write(33, 'J');
    EEPROM.write(34, 'M');
    EEPROM.write(35, 'B');


    //To make sure evrything is ok, verify the EEPROM data.
    Serial.println(F("Verify EEPROM data"));
    delay(1000);
    if(center_channel_1 != ((EEPROM.read(1) << 8) | EEPROM.read(0)))error = 1;
    if(center_channel_2 != ((EEPROM.read(3) << 8) | EEPROM.read(2)))error = 1;
    if(center_channel_3 != ((EEPROM.read(5) << 8) | EEPROM.read(4)))error = 1;
    if(center_channel_4 != ((EEPROM.read(7) << 8) | EEPROM.read(6)))error = 1;

    if(high_channel_1 != ((EEPROM.read(9) << 8) | EEPROM.read(8)))error = 1;
    if(high_channel_2 != ((EEPROM.read(11) << 8) | EEPROM.read(10)))error = 1;
    if(high_channel_3 != ((EEPROM.read(13) << 8) | EEPROM.read(12)))error = 1;
    if(high_channel_4 != ((EEPROM.read(15) << 8) | EEPROM.read(14)))error = 1;

    if(low_channel_1 != ((EEPROM.read(17) << 8) | EEPROM.read(16)))error = 1;
    if(low_channel_2 != ((EEPROM.read(19) << 8) | EEPROM.read(18)))error = 1;
    if(low_channel_3 != ((EEPROM.read(21) << 8) | EEPROM.read(20)))error = 1;
    if(low_channel_4 != ((EEPROM.read(23) << 8) | EEPROM.read(22)))error = 1;

    if(channel_1_assign != EEPROM.read(24))error = 1;
    if(channel_2_assign != EEPROM.read(25))error = 1;
    if(channel_3_assign != EEPROM.read(26))error = 1;
    if(channel_4_assign != EEPROM.read(27))error = 1;
```

```
    if(roll_axis != EEPROM.read(28))error = 1;
    if(pitch_axis != EEPROM.read(29))error = 1;
    if(yaw_axis != EEPROM.read(30))error = 1;
    if(type != EEPROM.read(31))error = 1;
    if(gyro_address != EEPROM.read(32))error = 1;

    if('J' != EEPROM.read(33))error = 1;
    if('M' != EEPROM.read(34))error = 1;
    if('B' != EEPROM.read(35))error = 1;

    if(error == 1)Serial.println(F("EEPROM verification failed!!! (ERROR 5)"));
    else Serial.println(F("Verification done"));
  }


  if(error == 0){
    Serial.println(F("Setup is finished."));
    Serial.println(F("You can now calibrate the esc's and upload the YMFC-AL code."));
  }
  else{
   Serial.println(F("The setup is aborted due to an error."));
   Serial.println(F("Check the Q and A page of the YMFC-AL project on:"));
   Serial.println(F("www.brokking.net for more information about this error."));
  }
  while(1);
}

//Search for the gyro and check the Who_am_I register
byte search_gyro(int gyro_address, int who_am_i){
  Wire.beginTransmission(gyro_address);
  Wire.write(who_am_i);
  Wire.endTransmission();
  Wire.requestFrom(gyro_address, 1);
  timer = millis() + 100;
  while(Wire.available() < 1 && timer > millis());
  lowByte = Wire.read();
  address = gyro_address;
  return lowByte;
}

void start_gyro(void){
  //Setup the L3G4200D or L3GD20H
  if(type == 2 || type == 3){
    Wire.beginTransmission(address);                              //Start communication
with the gyro with the address found during search
    Wire.write(0x20);                                               //We want to write
to register 1 (20 hex)
    Wire.write(0x0F);                                               //Set the register
bits as 00001111 (Turn on the gyro and enable all axis)
    Wire.endTransmission();                                       //End the transmission
with the gyro

    Wire.beginTransmission(address);                              //Start communication
with the gyro (adress 1101001)
```

```
    Wire.write(0x20);                                              //Start reading @
register 28h and auto increment with every read
    Wire.endTransmission();                                       //End the transmission
    Wire.requestFrom(address, 1);                                      //Request 6 bytes
from the gyro
    while(Wire.available() < 1);                                     //Wait until the 1
byte is received
    Serial.print(F("Register 0x20 is set to:"));
    Serial.println(Wire.read(),BIN);

    Wire.beginTransmission(address);                               //Start communication
with the gyro  with the address found during search
    Wire.write(0x23);                                                 //We want to write
to register 4 (23 hex)
    Wire.write(0x90);                                                 //Set the register
bits as 10010000 (Block Data Update active & 500dps full scale)
    Wire.endTransmission();                                       //End the transmission
with the gyro

    Wire.beginTransmission(address);                               //Start communication
with the gyro (adress 1101001)
    Wire.write(0x23);                                                  //Start reading @
register 28h and auto increment with every read
    Wire.endTransmission();                                       //End the transmission
    Wire.requestFrom(address, 1);                                      //Request 6 bytes
from the gyro
    while(Wire.available() < 1);                                     //Wait until the 1
byte is received
    Serial.print(F("Register 0x23 is set to:"));
    Serial.println(Wire.read(),BIN);

  }
  //Setup the MPU-6050
  if(type == 1){

    Wire.beginTransmission(address);                               //Start communication
with the gyro
    Wire.write(0x6B);                                             //PWR_MGMT_1 register
    Wire.write(0x00);                                             //Set to zero to turn
on the gyro
    Wire.endTransmission();                                       //End the transmission

    Wire.beginTransmission(address);                               //Start communication
with the gyro
    Wire.write(0x6B);                                                  //Start reading @
register 28h and auto increment with every read
    Wire.endTransmission();                                       //End the transmission
    Wire.requestFrom(address, 1);                                      //Request 1 bytes
from the gyro
    while(Wire.available() < 1);                                     //Wait until the 1
byte is received
    Serial.print(F("Register 0x6B is set to:"));
    Serial.println(Wire.read(),BIN);

    Wire.beginTransmission(address);  //Start communication with the gyro
```

```
    Wire.write(0x1B);                                                //GYRO_CONFIG register
    Wire.write(0x08);         //Set the register bits as 00001000 (500dps full scale)
    Wire.endTransmission();                                          //End the transmission

    Wire.beginTransmission(address); // communication with the gyro (adress 1101001)
    Wire.write(0x1B);//Start reading @ register 28h and auto increment with every read
    Wire.endTransmission();                                          //End the transmission
    Wire.requestFrom(address, 1);    //Request 1 bytes from the gyro
    while(Wire.available() < 1); //Wait until the 1 byte is received
    Serial.print(F("Register 0x1B is set to:"));
    Serial.println(Wire.read(),BIN);

  }
}

void gyro_signalen(){
  if(type == 2 || type == 3){
    Wire.beginTransmission(address);            //Start communication with the gyro
    Wire.write(168);                           //Start reading @ register 28h and auto
increment with every read
    Wire.endTransmission();                     //End the transmission
    Wire.requestFrom(address, 6);              //Request 6 bytes from the gyro
    while(Wire.available() < 6);              //Wait until the 6 bytes are received
    lowByte = Wire.read();  //First received byte is the low part of the angular data
    highByte = Wire.read();//Second received byte is the high part of the angular data
    gyro_roll = ((highByte<<8)|lowByte); //Multiply highByte by 256  and add lowByte
    if(cal_int  ==  2000)gyro_roll  -=  gyro_roll_cal;  //Only  compensate  after  the
calibration
    lowByte = Wire.read(); //First received byte is the low part of the angular data
    highByte = Wire.read();//Second received byte is the high part of the angular data
    gyro_pitch = ((highByte<<8)|lowByte ; //Multiply highByte by 256 (shift left by
8) and ad lowByte
    if(cal_int  ==  2000)gyro_pitch  -=  gyro_pitch_cal;  //Only  compensate  after  the
calibration
    lowByte = Wire.read();  //First received byte is the low part of the angular data
    highByte = Wire.read();//Second received byte is the high part of the angular data
    gyro_yaw = ((highByte<<8)|lowByte); //Multiply highByte by 256 (shift left by 8)
and ad lowByte
    if(cal_int  ==  2000)gyro_yaw  -=  gyro_yaw_cal;  //Only  compensate  after  the
calibration
  }
  if(type == 1){
    Wire.beginTransmission(address);    //Start communication with the gyro
    Wire.write(0x43);                       //Start  reading  @  register  43h  and  auto
increment with every read
    Wire.endTransmission();                  //End the transmission
    Wire.requestFrom(address,6);                                        //Request 6 bytes
from the gyro
    while(Wire.available() < 6);                                       //Wait until the 6
bytes are received
    gyro_roll=Wire.read()<<8|Wire.read(); //Read high and low part of the angular data
    if(cal_int  ==  2000)gyro_roll  -=  gyro_roll_cal;  //Only  compensate  after  the
calibration
    gyro_pitch=Wire.read()<<8|Wire.read();//Read high and low part of the angular data
```

```
    if(cal_int == 2000)gyro_pitch -= gyro_pitch_cal;//Only compensate after the
calibration
    gyro_yaw=Wire.read()<<8|Wire.read();  //Read high and low part of the angular data
    if(cal_int == 2000)gyro_yaw -= gyro_yaw_cal;//Only compensate after the
calibration
  }
}

//Check if a receiver input value is changing within 30 seconds
void check_receiver_inputs(byte movement){
  byte trigger = 0;
  int pulse_length;
  timer = millis() + 30000;
  while(timer > millis() && trigger == 0){
    delay(250);
    if(receiver_input_channel_1 > 1750 || receiver_input_channel_1 < 1250){
      trigger = 1;
      receiver_check_byte |= 0b00000001;
      pulse_length = receiver_input_channel_1;
    }
    if(receiver_input_channel_2 > 1750 || receiver_input_channel_2 < 1250){
      trigger = 2;
      receiver_check_byte |= 0b00000010;
      pulse_length = receiver_input_channel_2;
    }
    if(receiver_input_channel_3 > 1750 || receiver_input_channel_3 < 1250){
      trigger = 3;
      receiver_check_byte |= 0b00000100;
      pulse_length = receiver_input_channel_3;
    }
    if(receiver_input_channel_4 > 1750 || receiver_input_channel_4 < 1250){
      trigger = 4;
      receiver_check_byte |= 0b00001000;
      pulse_length = receiver_input_channel_4;
    }
  }
  if(trigger == 0){
    error = 1;
    Serial.println(F("No stick movement detected in the last 30 seconds!!! (ERROR
2)"));
  }
  //Assign the stick to the function.
  else{
    if(movement == 1){
      channel_3_assign = trigger;
      if(pulse_length < 1250)channel_3_assign += 0b10000000;
    }
    if(movement == 2){
      channel_1_assign = trigger;
      if(pulse_length < 1250)channel_1_assign += 0b10000000;
    }
    if(movement == 3){
      channel_2_assign = trigger;
      if(pulse_length < 1250)channel_2_assign += 0b10000000;
    }
```

```
    if(movement == 4){
      channel_4_assign = trigger;
      if(pulse_length < 1250)channel_4_assign += 0b10000000;
    }
  }
}

void check_to_continue(){
  byte continue_byte = 0;
  while(continue_byte == 0){
    if(channel_2_assign == 0b00000001 && receiver_input_channel_1 > center_channel_1
+ 150)continue_byte = 1;
    if(channel_2_assign == 0b10000001 && receiver_input_channel_1 < center_channel_1
- 150)continue_byte = 1;
    if(channel_2_assign == 0b00000010 && receiver_input_channel_2 > center_channel_2
+ 150)continue_byte = 1;
    if(channel_2_assign == 0b10000010 && receiver_input_channel_2 < center_channel_2
- 150)continue_byte = 1;
    if(channel_2_assign == 0b00000011 && receiver_input_channel_3 > center_channel_3
+ 150)continue_byte = 1;
    if(channel_2_assign == 0b10000011 && receiver_input_channel_3 < center_channel_3
- 150)continue_byte = 1;
    if(channel_2_assign == 0b00000100 && receiver_input_channel_4 > center_channel_4
+ 150)continue_byte = 1;
    if(channel_2_assign == 0b10000100 && receiver_input_channel_4 < center_channel_4
- 150)continue_byte = 1;
    delay(100);
  }
  wait_sticks_zero();
}

//Check if the transmitter sticks are in the neutral position
void wait_sticks_zero(){
  byte zero = 0;
  while(zero < 15){
    if(receiver_input_channel_1 < center_channel_1 + 20 && receiver_input_channel_1 >
center_channel_1 - 20)zero |= 0b00000001;
    if(receiver_input_channel_2 < center_channel_2 + 20 && receiver_input_channel_2 >
center_channel_2 - 20)zero |= 0b00000010;
    if(receiver_input_channel_3 < center_channel_3 + 20 && receiver_input_channel_3 >
center_channel_3 - 20)zero |= 0b00000100;
    if(receiver_input_channel_4 < center_channel_4 + 20 && receiver_input_channel_4 >
center_channel_4 - 20)zero |= 0b00001000;
    delay(100);
  }
}

//Checck if the receiver values are valid within 10 seconds
void wait_for_receiver(){
  byte zero = 0;
  timer = millis() + 10000;
  while(timer > millis() && zero < 15){
    if(receiver_input_channel_1 < 2100 && receiver_input_channel_1 > 900)zero |=
0b00000001;
```

```
    if(receiver_input_channel_2 < 2100 && receiver_input_channel_2 > 900)zero |=
0b00000010;
    if(receiver_input_channel_3 < 2100 && receiver_input_channel_3 > 900)zero |=
0b00000100;
    if(receiver_input_channel_4 < 2100 && receiver_input_channel_4 > 900)zero |=
0b00001000;
    delay(500);
    Serial.print(F("."));
  }
  if(zero == 0){
    error = 1;
    Serial.println(F("."));
    Serial.println(F("No valid receiver signals found!!! (ERROR 1)"));
  }
  else Serial.println(F(" OK"));
}

//Register the min and max receiver values and exit when the sticks are back in the
neutral position
void register_min_max(){
  byte zero = 0;
  low_channel_1 = receiver_input_channel_1;
  low_channel_2 = receiver_input_channel_2;
  low_channel_3 = receiver_input_channel_3;
  low_channel_4 = receiver_input_channel_4;
  while(receiver_input_channel_1 < center_channel_1 + 20 && receiver_input_channel_1
> center_channel_1 - 20)delay(250);
  Serial.println(F("Measuring endpoints...."));
  while(zero < 15){
    if(receiver_input_channel_1 < center_channel_1 + 20 && receiver_input_channel_1 >
center_channel_1 - 20)zero |= 0b00000001;
    if(receiver_input_channel_2 < center_channel_2 + 20 && receiver_input_channel_2 >
center_channel_2 - 20)zero |= 0b00000010;
    if(receiver_input_channel_3 < center_channel_3 + 20 && receiver_input_channel_3 >
center_channel_3 - 20)zero |= 0b00000100;
    if(receiver_input_channel_4 < center_channel_4 + 20 && receiver_input_channel_4 >
center_channel_4 - 20)zero |= 0b00001000;
    if(receiver_input_channel_1        <        low_channel_1)low_channel_1        =
receiver_input_channel_1;
    if(receiver_input_channel_2        <        low_channel_2)low_channel_2        =
receiver_input_channel_2;
    if(receiver_input_channel_3        <        low_channel_3)low_channel_3        =
receiver_input_channel_3;
    if(receiver_input_channel_4        <        low_channel_4)low_channel_4        =
receiver_input_channel_4;
    if(receiver_input_channel_1       >       high_channel_1)high_channel_1       =
receiver_input_channel_1;
    if(receiver_input_channel_2       >       high_channel_2)high_channel_2       =
receiver_input_channel_2;
    if(receiver_input_channel_3       >       high_channel_3)high_channel_3       =
receiver_input_channel_3;
    if(receiver_input_channel_4       >       high_channel_4)high_channel_4       =
receiver_input_channel_4;
    delay(100);
  }
```

```
}

//Check if the angular position of a gyro axis is changing within 10 seconds
void check_gyro_axes(byte movement){
  byte trigger_axis = 0;
  float gyro_angle_roll, gyro_angle_pitch, gyro_angle_yaw;
  //Reset all axes
  gyro_angle_roll = 0;
  gyro_angle_pitch = 0;
  gyro_angle_yaw = 0;
  gyro_signalen();
  timer = millis() + 10000;
  while(timer > millis() && gyro_angle_roll > -30 && gyro_angle_roll < 30 &&
gyro_angle_pitch > -30 && gyro_angle_pitch < 30 && gyro_angle_yaw > -30 &&
gyro_angle_yaw < 30){
    gyro_signalen();
    if(type == 2 || type == 3){
      gyro_angle_roll += gyro_roll * 0.00007;              //0.00007 = 17.5 (md/s) /
250(Hz)
      gyro_angle_pitch += gyro_pitch * 0.00007;
      gyro_angle_yaw += gyro_yaw * 0.00007;
    }
    if(type == 1){
      gyro_angle_roll += gyro_roll * 0.0000611;          // 0.0000611 = 1 / 65.5 (LSB
degr/s) / 250(Hz)
      gyro_angle_pitch += gyro_pitch * 0.0000611;
      gyro_angle_yaw += gyro_yaw * 0.0000611;
    }

    delayMicroseconds(3700);  //Loop is running @ 250Hz. +/-300us is used for
communication with the gyro
  }
  //Assign the moved axis to the orresponding function (pitch, roll, yaw)
  if((gyro_angle_roll < -30 || gyro_angle_roll > 30) && gyro_angle_pitch > -30 &&
gyro_angle_pitch < 30 && gyro_angle_yaw > -30 && gyro_angle_yaw < 30){
    gyro_check_byte |= 0b00000001;
    if(gyro_angle_roll < 0)trigger_axis = 0b10000001;
    else trigger_axis = 0b00000001;
  }
  if((gyro_angle_pitch < -30 || gyro_angle_pitch > 30) && gyro_angle_roll > -30 &&
gyro_angle_roll < 30 && gyro_angle_yaw > -30 && gyro_angle_yaw < 30){
    gyro_check_byte |= 0b00000010;
    if(gyro_angle_pitch < 0)trigger_axis = 0b10000010;
    else trigger_axis = 0b00000010;
  }
  if((gyro_angle_yaw < -30 || gyro_angle_yaw > 30) && gyro_angle_roll > -30 &&
gyro_angle_roll < 30 && gyro_angle_pitch > -30 && gyro_angle_pitch < 30){
    gyro_check_byte |= 0b00000100;
    if(gyro_angle_yaw < 0)trigger_axis = 0b10000011;
    else trigger_axis = 0b00000011;
  }

  if(trigger_axis == 0){
    error = 1;
```

```
    Serial.println(F("No angular motion is detected in the last 10 seconds!!! (ERROR
4)"));
  }
  else
  if(movement == 1)roll_axis = trigger_axis;
  if(movement == 2)pitch_axis = trigger_axis;
  if(movement == 3)yaw_axis = trigger_axis;

}

//This routine is called every time input 8, 9, 10 or 11 changed state
ISR(PCINT0_vect){
  current_time = micros();
  //Channel 1=========================================
  if(PINB & B00000001){                                        //Is input 8 high?
    if(last_channel_1 == 0){                                   //Input 8 changed from
0 to 1
      last_channel_1 = 1;                                         //Remember current
input state
      timer_1 = current_time;                                      //Set timer_1 to
current_time
    }
  }
  else if(last_channel_1 == 1){                                //Input 8 is not high
and changed from 1 to 0
    last_channel_1 = 0;                                         //Remember current
input state
    receiver_input_channel_1 = current_time - timer_1;             //Channel 1 is
current_time - timer_1
  }
  //Channel 2=========================================
  if(PINB & B00000010 ){                                       //Is input 9 high?
    if(last_channel_2 == 0){                                   //Input 9 changed from
0 to 1
      last_channel_2 = 1;                                         //Remember current
input state
      timer_2 = current_time;                                      //Set timer_2 to
current_time
    }
  }
  else if(last_channel_2 == 1){                                //Input 9 is not high
and changed from 1 to 0
    last_channel_2 = 0;                                         //Remember current
input state
    receiver_input_channel_2 = current_time - timer_2;             //Channel 2 is
current_time - timer_2
  }
  //Channel 3=========================================
  if(PINB & B00000100 ){                                       //Is input 10 high?
    if(last_channel_3 == 0){                                  //Input 10 changed from
0 to 1
      last_channel_3 = 1;                                         //Remember current
input state
      timer_3 = current_time;                                      //Set timer_3 to
current_time
```

```
    }
  }
  else if(last_channel_3 == 1){                                    //Input 10 is not high
and changed from 1 to 0
    last_channel_3 = 0;                                                //Remember current
input state
    receiver_input_channel_3 = current_time - timer_3;              //Channel 3 is
current_time - timer_3

  }
  //Channel 4=======================================
  if(PINB & B00001000 ){                                            //Is input 11 high?
    if(last_channel_4 == 0){                        //Input 11 changed from 0 to 1
      last_channel_4 = 1;                          //Remember current input state
      timer_4 = current_time;                 //Set timer_4 to current_time
    }
  }
  else if(last_channel_4 == 1){       //Input 11 is not high and changed from 1 to 0
    last_channel_4 = 0;              //Remember current input state
    receiver_input_channel_4 = current_time - timer_4;
  }
}

//Intro subroutine
void intro(){
  Serial.println(F("====================================================="));
  delay(1500);
  Serial.println(F(""));
  Serial.println(F("Your"));
  delay(500);
  Serial.println(F("  Multicopter"));
  delay(500);
  Serial.println(F("    Flight"));
  delay(500);
  Serial.println(F("      Controller"));
  delay(1000);
  Serial.println(F(""));
  Serial.println(F("FC-AL Setup Program"));
  Serial.println(F(""));
  Serial.println(F("====================================================="));
  delay(1500);
 }
```

ESC Callibrate.ino

```
//The program will start in calibration mode.
//Send the following characters / numbers via the serial monitor to change the mode
//
//r = print receiver signals.
//a = print quadcopter angles.
//1 = check rotation / vibrations for motor 1 (right front CCW).
//2 = check rotation / vibrations for motor 2 (right rear CW).
//3 = check rotation / vibrations for motor 3 (left rear CCW).
//4 = check rotation / vibrations for motor 4 (left front CW).
```

```
//5 = check vibrations for all motors together.


#include <Wire.h>                                        //Include the Wire.h library so
we can communicate with the gyro.
#include <EEPROM.h>                                       //Include the EEPROM.h library
so we can store information onto the EEPROM

//Declaring global variables
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte eeprom_data[36], start, data;
boolean new_function_request,first_angle;
volatile        int        receiver_input_channel_1,        receiver_input_channel_2,
receiver_input_channel_3, receiver_input_channel_4;
int esc_1, esc_2, esc_3, esc_4;
int counter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4;
int receiver_input[5];
int loop_counter, gyro_address, vibration_counter;
int temperature;
long acc_x, acc_y, acc_z, acc_total_vector[20], acc_av_vector, vibration_total_result;
unsigned  long  timer_channel_1,  timer_channel_2,  timer_channel_3,  timer_channel_4,
esc_timer, esc_loop_timer;
unsigned long zero_timer, timer_1, timer_2, timer_3, timer_4, current_time;

int acc_axis[4], gyro_axis[4];
double gyro_pitch, gyro_roll, gyro_yaw;
float angle_roll_acc, angle_pitch_acc, angle_pitch, angle_roll;
int cal_int;
double gyro_axis_cal[4];

//Setup routine
void setup(){
  Serial.begin(57600);                                                          //Start
the serial port.
  Wire.begin();                                                                 //Start
the wire library as master
  TWBR = 12;                                                                     //Set
the I2C clock speed to 400kHz.
  Serial.println("Setup");
  //Arduino Uno pins default to inputs, so they don't need to be explicitly declared
as inputs.
  DDRD |= B11110000;                                                          //Configure
digital poort 4, 5, 6 and 7 as output.
  DDRB |= B00010000;                                                          //Configure
digital poort 12 as output.

  PCICR |= (1 << PCIE0);                                                          //
set PCIE0 to enable PCMSK0 scan.
  PCMSK0 |= (1 << PCINT0);                                                        //
set PCINT0 (digital input 8) to trigger an interrupt on state change.
  PCMSK0 |= (1 << PCINT1);                                                        //
set PCINT1 (digital input 9)to trigger an interrupt on state change.
  PCMSK0 |= (1 << PCINT2);                                                        //
set PCINT2 (digital input 10)to trigger an interrupt on state change.
```

```
  PCMSK0 |= (1 << PCINT3);                                                    //
set PCINT3 (digital input 11)to trigger an interrupt on state change.

  for(data = 0; data <= 35; data++)eeprom_data[data] = EEPROM.read(data);
//Read EEPROM for faster data access

  gyro_address = eeprom_data[32];                                            //Store
the gyro address in the variable.

  set_gyro_registers();                                                       //Set
the specific gyro registers.

  //Check the EEPROM signature to make sure that the setup program is executed.
  while(eeprom_data[33] != 'J' || eeprom_data[34] != 'M' || eeprom_data[35] != 'B'){
    delay(500);                                                              //Wait
for 500ms.
    digitalWrite(12, !digitalRead(12));                                      //Change
the led status to indicate error.
  }
  wait_for_receiver();                                                       //Wait
until the receiver is active.
  zero_timer = micros();                                                     //Set
the zero_timer for the first loop.

  while(Serial.available())data = Serial.read();                             //Empty
the serial buffer.
  data = 0;                                                                  //Set
the data variable back to zero.
}

//Main program loop
void loop(){
  while(zero_timer + 4000 > micros());                                       //Start
the pulse after 4000 micro seconds.
  zero_timer = micros();                                                     //Reset
the zero timer.

  if(Serial.available() > 0){
    data = Serial.read();                                                    //Read
the incomming byte.
    new_function_request = true;                                             //Set
the new request flag.
    loop_counter = 0;                                                        //Reset
the loop_counter variable.
    cal_int = 0;                                                             //Reset
the cal_int variable to undo the calibration.
    start = 0;                                                               //Set
start to 0.
    first_angle = false;                                                     //Set
first_angle to false.
    //Confirm the choice on the serial monitor.
    if(data == 'r')Serial.println("Reading receiver signals.");
    if(data == 'a')Serial.println("Print the quadcopter angles.");
    if(data == 'a')Serial.println("Gyro calibration starts in 2 seconds (don't move
the quadcopter).");
```

```
    if(data == '1')Serial.println("Test motor 1 (right front CCW.)");
    if(data == '2')Serial.println("Test motor 2 (right rear CW.)");
    if(data == '3')Serial.println("Test motor 3 (left rear CCW.)");
    if(data == '4')Serial.println("Test motor 4 (left front CW.)");
    if(data == '5')Serial.println("Test all motors together");

    //Let's create a small delay so the message stays visible for 2.5 seconds.
    //We don't want the ESC's to beep and have to send a 1000us pulse to the ESC's.
    for(vibration_counter = 0; vibration_counter < 625; vibration_counter++){
//Do this loop 625 times
      delay(3);                                                                 //Wait
3000us.
      esc_1 = 1000;                                                              //Set
the pulse for ESC 1 to 1000us.
      esc_2 = 1000;                                                              //Set
the pulse for ESC 1 to 1000us.
      esc_3 = 1000;                                                              //Set
the pulse for ESC 1 to 1000us.
      esc_4 = 1000;                                                              //Set
the pulse for ESC 1 to 1000us.
      esc_pulse_output();                                                       //Send
the ESC control pulses.
    }
    vibration_counter = 0;                                                     //Reset
the vibration_counter variable.
  }

  receiver_input_channel_3              =              convert_receiver_channel(3);
//Convert the actual receiver signals for throttle to the standard 1000 - 2000us.
  if(receiver_input_channel_3     <     1025)new_function_request     =     false;
//If the throttle is in the lowest position set the request flag to false.



////////////////////////////////////////////////////////////////////////////////
////////
  //Run the ESC calibration program to start with.

////////////////////////////////////////////////////////////////////////////////
////////
  if(data == 0 && new_function_request == false){                             //Only
start the calibration mode at first start.
    receiver_input_channel_3             =             convert_receiver_channel(3);
//Convert the actual receiver signals for throttle to the standard 1000 - 2000us.
    esc_1 = receiver_input_channel_3;                                          //Set
the pulse for motor 1 equal to the throttle channel.
    esc_2 = receiver_input_channel_3;                                          //Set
the pulse for motor 2 equal to the throttle channel.
    esc_3 = receiver_input_channel_3;                                          //Set
the pulse for motor 3 equal to the throttle channel.
    esc_4 = receiver_input_channel_3;                                          //Set
the pulse for motor 4 equal to the throttle channel.
    esc_pulse_output();                                                       //Send
the ESC control pulses.
  }
```

```
////////////////////////////////////////////////////////////////////////////////
////////
  //When user sends a 'r' print the receiver signals.

////////////////////////////////////////////////////////////////////////////////
////////
  if(data == 'r'){
    loop_counter ++;                                                    //Increase
the loop_counter variable.
    receiver_input_channel_1             =             convert_receiver_channel(1);
//Convert the actual receiver signals for pitch to the standard 1000 - 2000us.
    receiver_input_channel_2             =             convert_receiver_channel(2);
//Convert the actual receiver signals for roll to the standard 1000 - 2000us.
    receiver_input_channel_3             =             convert_receiver_channel(3);
//Convert the actual receiver signals for throttle to the standard 1000 - 2000us.
    receiver_input_channel_4             =             convert_receiver_channel(4);
//Convert the actual receiver signals for yaw to the standard 1000 - 2000us.

    if(loop_counter == 125){                                                  //Print
the receiver values when the loop_counter variable equals 250.
      print_signals();                                                        //Print
the receiver values on the serial monitor.
      loop_counter = 0;                                                       //Reset
the loop_counter variable.
    }

    //For starting the motors: throttle low and yaw left (step 1).
    if(receiver_input_channel_3 < 1050 && receiver_input_channel_4 < 1050)start = 1;
    //When yaw stick is back in the center position start the motors (step 2).
    if(start == 1 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 >
1450)start = 2;
    //Stopping the motors: throttle low and yaw right.
    if(start == 2 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 >
1950)start = 0;

    esc_1 = 1000;                                                              //Set
the pulse for ESC 1 to 1000us.
    esc_2 = 1000;                                                              //Set
the pulse for ESC 1 to 1000us.
    esc_3 = 1000;                                                              //Set
the pulse for ESC 1 to 1000us.
    esc_4 = 1000;                                                              //Set
the pulse for ESC 1 to 1000us.
    esc_pulse_output();                                                       //Send
the ESC control pulses.
  }


////////////////////////////////////////////////////////////////////////////////
///////
  //When user sends a '1, 2, 3, 4 or 5 test the motors.
```

```
////////////////////////////////////////////////////////////////////////////////
////////
  if(data == '1' || data == '2' || data == '3' || data == '4' || data == '5'){
//If motor 1, 2, 3 or 4 is selected by the user.
    loop_counter ++;                                                          //Add
1 to the loop_counter variable.
    if(new_function_request     ==     true     &&     loop_counter     ==     250){
//Wait for the throttle to be set to 0.
      Serial.print("Set   throttle   to   1000   (low).   It's   now   set   to:   ");
//Print message on the serial monitor.
      Serial.println(receiver_input_channel_3);
//Print the actual throttle position.
      loop_counter = 0;                                                      //Reset
the loop_counter variable.
    }
    if(new_function_request == false){                                        //When
the throttle was in the lowest position do this.
      receiver_input_channel_3            =            convert_receiver_channel(3);
//Convert the actual receiver signals for throttle to the standard 1000 - 2000us.
      if(data   ==   '1'   ||   data   ==   '5')esc_1   =   receiver_input_channel_3;
//If motor 1 is requested set the pulse for motor 1 equal to the throttle channel.
      else esc_1 = 1000;                                                      //If
motor 1 is not requested set the pulse for the ESC to 1000us (off).
      if(data   ==   '2'   ||   data   ==   '5')esc_2   =   receiver_input_channel_3;
//If motor 2 is requested set the pulse for motor 1 equal to the throttle channel.
      else esc_2 = 1000;                                                      //If
motor 2 is not requested set the pulse for the ESC to 1000us (off).
      if(data   ==   '3'   ||   data   ==   '5')esc_3   =   receiver_input_channel_3;
//If motor 3 is requested set the pulse for motor 1 equal to the throttle channel.
      else esc_3 = 1000;                                                      //If
motor 3 is not requested set the pulse for the ESC to 1000us (off).
      if(data   ==   '4'   ||   data   ==   '5')esc_4   =   receiver_input_channel_3;
//If motor 4 is requested set the pulse for motor 1 equal to the throttle channel.
      else esc_4 = 1000;                                                      //If
motor 4 is not requested set the pulse for the ESC to 1000us (off).

      esc_pulse_output();                                                     //Send
the ESC control pulses.

      //For balancing the propellors it's possible to use the accelerometer to measure
the vibrations.
      if(eeprom_data[31] == 1){                                               //The
MPU-6050 is installed
        Wire.beginTransmission(gyro_address);                                 //Start
communication with the gyro.
        Wire.write(0x3B);                                                     //Start
reading @ register 43h and auto increment with every read.
        Wire.endTransmission();                                               //End
the transmission.
        Wire.requestFrom(gyro_address,6);
//Request 6 bytes from the gyro.
        while(Wire.available() < 6);                                          //Wait
until the 6 bytes are received.
```

```
        acc_x = Wire.read()<<8|Wire.read();                                  //Add
the low and high byte to the acc_x variable.
        acc_y = Wire.read()<<8|Wire.read();                                  //Add
the low and high byte to the acc_y variable.
        acc_z = Wire.read()<<8|Wire.read();                                  //Add
the low and high byte to the acc_z variable.

        acc_total_vector[0]    =    sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));
//Calculate the total accelerometer vector.

        acc_av_vector = acc_total_vector[0];                                 //Copy
the total vector to the accelerometer average vector variable.

        for(start = 16; start > 0; start--){                                  //Do
this loop 16 times to create an array of accelrometer vectors.
          acc_total_vector[start]      =      acc_total_vector[start      -      1];
//Shift every variable one position up in the array.
          acc_av_vector += acc_total_vector[start];                          //Add
the array value to the acc_av_vector variable.
        }

        acc_av_vector /= 17;                                                //Divide
the acc_av_vector by 17 to get the avarage total accelerometer vector.

        if(vibration_counter < 20){                                          //If
the vibration_counter is less than 20 do this.
          vibration_counter ++;                                            //Increment
the vibration_counter variable.
          vibration_total_result   +=   abs(acc_total_vector[0]   -   acc_av_vector);
//Add the absolute difference between the avarage vector and current vector to the
vibration_total_result variable.
        }
        else{
          vibration_counter = 0;                                             //If
the vibration_counter is equal or larger than 20 do this.
          Serial.println(vibration_total_result/50);
//Print the total accelerometer vector divided by 50 on the serial monitor.
          vibration_total_result = 0;                                       //Reset
the vibration_total_result variable.
        }
      }
    }
  }

////////////////////////////////////////////////////////////////////////////////
///////
  //When user sends a 'a' display the quadcopter angles.

////////////////////////////////////////////////////////////////////////////////
////////
  if(data == 'a'){

    if(cal_int != 2000){
      Serial.print("Calibrating the gyro");
```

```
      //Let's take multiple gyro data samples so we can determine the average gyro
offset (calibration).
      for (cal_int = 0; cal_int < 2000 ; cal_int ++){                           //Take
2000 readings for calibration.
        if(cal_int % 125 == 0){
          digitalWrite(12, !digitalRead(12));   //Change the led status to indicate
calibration.
          Serial.print(".");
        }
        gyro_signalen();                                                        //Read
the gyro output.
        gyro_axis_cal[1] += gyro_axis[1];                                        //Ad
roll value to gyro_roll_cal.
        gyro_axis_cal[2] += gyro_axis[2];                                        //Ad
pitch value to gyro_pitch_cal.
        gyro_axis_cal[3] += gyro_axis[3];                                        //Ad
yaw value to gyro_yaw_cal.
        //We don't want the esc's to be beeping annoyingly. So let's give them a 1000us
puls while calibrating the gyro.
        PORTD |= B11110000;                                                      //Set
digital poort 4, 5, 6 and 7 high.
        delayMicroseconds(1000);                                                //Wait
1000us.
        PORTD &= B00001111;                                                      //Set
digital poort 4, 5, 6 and 7 low.
        delay(3);                                                               //Wait
3 milliseconds before the next loop.
      }
      Serial.println(".");
      //Now that we have 2000 measures, we need to devide by 2000 to get the average
gyro offset.
      gyro_axis_cal[1] /= 2000;                                                 //Divide
the roll total by 2000.
      gyro_axis_cal[2] /= 2000;                                                 //Divide
the pitch total by 2000.
      gyro_axis_cal[3] /= 2000;                                                 //Divide
the yaw total by 2000.
    }
    else{
      ///We don't want the esc's to be beeping annoyingly. So let's give them a 1000us
puls while calibrating the gyro.
      PORTD |= B11110000;                                                       //Set
digital poort 4, 5, 6 and 7 high.
      delayMicroseconds(1000);                                                  //Wait
1000us.
      PORTD &= B00001111;                                                       //Set
digital poort 4, 5, 6 and 7 low.

      //Let's get the current gyro data.
      gyro_signalen();

      //Gyro angle calculations
      //0.0000611 = 1 / (250Hz / 65.5)
      angle_pitch           +=           gyro_pitch           *           0.0000611;
//Calculate the traveled pitch angle and add this to the angle_pitch variable.
```

```
      angle_roll           +=           gyro_roll           *           0.0000611;
//Calculate the traveled roll angle and add this to the angle_roll variable.

      //0.000001066 = 0.0000611 * (3.142(PI) / 180degr) The Arduino sin function is
in radians
      angle_pitch -= angle_roll * sin(gyro_yaw * 0.000001066);                    //If
the IMU has yawed transfer the roll angle to the pitch angel.
      angle_roll += angle_pitch * sin(gyro_yaw * 0.000001066);                    //If
the IMU has yawed transfer the pitch angle to the roll angel.

      //Accelerometer angle calculations
      acc_total_vector[0]     =     sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));
//Calculate the total accelerometer vector.

      //57.296 = 1 / (3.142 / 180) The Arduino asin function is in radians
      angle_pitch_acc      =      asin((float)acc_y/acc_total_vector[0])*      57.296;
//Calculate the pitch angle.
      angle_roll_acc      =      asin((float)acc_x/acc_total_vector[0])*      -57.296;
//Calculate the roll angle.

      if(!first_angle){
        angle_pitch = angle_pitch_acc;                                            //Set
the pitch angle to the accelerometer angle.
        angle_roll = angle_roll_acc;                                              //Set
the roll angle to the accelerometer angle.
        first_angle = true;
      }
      else{
        angle_pitch   =   angle_pitch   *   0.9996   +   angle_pitch_acc   *   0.0004;
//Correct the drift of the gyro pitch angle with the accelerometer pitch angle.
        angle_roll   =   angle_roll   *   0.9996   +   angle_roll_acc   *   0.0004;
//Correct the drift of the gyro roll angle with the accelerometer roll angle.
      }

      //We can't print all the data at once. This takes to long and the angular
readings will be off.
      if(loop_counter == 0)Serial.print("Pitch: ");
      if(loop_counter == 1)Serial.print(angle_pitch ,0);
      if(loop_counter == 2)Serial.print(" Roll: ");
      if(loop_counter == 3)Serial.print(angle_roll ,0);
      if(loop_counter == 4)Serial.print(" Yaw: ");
      if(loop_counter == 5)Serial.println(gyro_yaw / 65.5 ,0);

      loop_counter ++;
      if(loop_counter == 60)loop_counter = 0;
    }
  }
}



//This routine is called every time input 8, 9, 10 or 11 changed state.
ISR(PCINT0_vect){
  current_time = micros();
  //Channel 1=========================================
```

```
  if(PINB & B00000001){                                          //Is input 8 high?
    if(last_channel_1 == 0){                                     //Input 8 changed from
0 to 1.
      last_channel_1 = 1;                                           //Remember current
input state.
      timer_1 = current_time;                                        //Set timer_1 to
current_time.
    }
  }
  else if(last_channel_1 == 1){                                   //Input 8 is not high
and changed from 1 to 0.
    last_channel_1 = 0;                                          //Remember current
input state.
    receiver_input[1] = current_time - timer_1;                      //Channel 1 is
current_time - timer_1.
  }
  //Channel 2=======================================
  if(PINB & B00000010 ){                                         //Is input 9 high?
    if(last_channel_2 == 0){                                     //Input 9 changed from
0 to 1.
      last_channel_2 = 1;                                           //Remember current
input state.
      timer_2 = current_time;                                        //Set timer_2 to
current_time.
    }
  }
  else if(last_channel_2 == 1){                                   //Input 9 is not high
and changed from 1 to 0.
    last_channel_2 = 0;                                          //Remember current
input state.
    receiver_input[2] = current_time - timer_2;                      //Channel 2 is
current_time - timer_2.
  }
  //Channel 3=======================================
  if(PINB & B00000100 ){                                         //Is input 10 high?
    if(last_channel_3 == 0){                                    //Input 10 changed from
0 to 1.
      last_channel_3 = 1;                                           //Remember current
input state.
      timer_3 = current_time;                                        //Set timer_3 to
current_time.
    }
  }
  else if(last_channel_3 == 1){                                  //Input 10 is not high
and changed from 1 to 0.
    last_channel_3 = 0;                                          //Remember current
input state.
    receiver_input[3] = current_time - timer_3;                      //Channel 3 is
current_time - timer_3.
  }
  //Channel 4=======================================
  if(PINB & B00001000 ){                                         //Is input 11 high?
    if(last_channel_4 == 0){                                    //Input 11 changed from
0 to 1.
```

```
      last_channel_4 = 1;                                          //Remember current
input state.
      timer_4 = current_time;                                       //Set timer_4 to
current_time.
    }
  }
  else if(last_channel_4 == 1){                                  //Input 11 is not high
and changed from 1 to 0.
    last_channel_4 = 0;                                          //Remember current
input state.
    receiver_input[4] = current_time - timer_4;                    //Channel 4 is
current_time - timer_4.
  }
}

//Checck if the receiver values are valid within 10 seconds
void wait_for_receiver(){
  byte zero = 0;                                                              //Set
all bits in the variable zero to 0
  while(zero < 15){                                                          //Stay
in this loop until the 4 lowest bits are set
    if(receiver_input[1] < 2100 && receiver_input[1] > 900)zero |= 0b00000001;  //Set
bit 0 if the receiver pulse 1 is within the 900 - 2100 range
    if(receiver_input[2] < 2100 && receiver_input[2] > 900)zero |= 0b00000010;  //Set
bit 1 if the receiver pulse 2 is within the 900 - 2100 range
    if(receiver_input[3] < 2100 && receiver_input[3] > 900)zero |= 0b00000100;  //Set
bit 2 if the receiver pulse 3 is within the 900 - 2100 range
    if(receiver_input[4] < 2100 && receiver_input[4] > 900)zero |= 0b00001000;  //Set
bit 3 if the receiver pulse 4 is within the 900 - 2100 range
    delay(500);                                                              //Wait
500 milliseconds
  }
}

//This part converts the actual receiver signals to a standardized 1000 – 1500 – 2000
microsecond value.
//The stored data in the EEPROM is used.
int convert_receiver_channel(byte function){
  byte channel, reverse;                                                    //First
we declare some local variables
  int low, center, high, actual;
  int difference;

  channel = eeprom_data[function + 23] & 0b00000111;                         //What
channel corresponds with the specific function
  if(eeprom_data[function + 23] & 0b10000000)reverse = 1;                   //Reverse
channel when most significant bit is set
  else reverse = 0;                                                          //If
the most significant is not set there is no reverse

  actual = receiver_input[channel];                                         //Read
the actual receiver value for the corresponding function
  low = (eeprom_data[channel * 2 + 15] << 8) | eeprom_data[channel * 2 + 14];  //Store
the low value for the specific receiver input channel
```

```
  center = (eeprom_data[channel * 2 - 1] << 8) | eeprom_data[channel * 2 - 2]; //Store
the center value for the specific receiver input channel
  high = (eeprom_data[channel * 2 + 7] << 8) | eeprom_data[channel * 2 + 6];   //Store
the high value for the specific receiver input channel

  if(actual < center){                                                          //The
actual receiver value is lower than the center value
    if(actual < low)actual = low;                                              //Limit
the lowest value to the value that was detected during setup
    difference  =  ((long)(center  -  actual)  *  (long)500)  /  (center  -  low);
//Calculate and scale the actual value to a 1000 - 2000us value
    if(reverse == 1)return 1500 + difference;                                   //If
the channel is reversed
    else return 1500 - difference;                                              //If
the channel is not reversed
  }
  else                    if(actual                     >                     center){
//The actual receiver value is higher than the center value
    if(actual > high)actual = high;                                             //Limit
the lowest value to the value that was detected during setup
    difference  =  ((long)(actual  -  center)  *  (long)500)  /  (high  -  center);
//Calculate and scale the actual value to a 1000 - 2000us value
    if(reverse == 1)return 1500 - difference;                                   //If
the channel is reversed
    else return 1500 + difference;                                              //If
the channel is not reversed
  }
  else return 1500;
}

void print_signals(){
  Serial.print("Start:");
  Serial.print(start);

  Serial.print("  Roll:");
  if(receiver_input_channel_1 - 1480 < 0)Serial.print("<<<");
  else if(receiver_input_channel_1 - 1520 > 0)Serial.print(">>>");
  else Serial.print("-+-");
  Serial.print(receiver_input_channel_1);

  Serial.print("  Pitch:");
  if(receiver_input_channel_2 - 1480 < 0)Serial.print("^^^");
  else if(receiver_input_channel_2 - 1520 > 0)Serial.print("vvv");
  else Serial.print("-+-");
  Serial.print(receiver_input_channel_2);

  Serial.print("  Throttle:");
  if(receiver_input_channel_3 - 1480 < 0)Serial.print("vvv");
  else if(receiver_input_channel_3 - 1520 > 0)Serial.print("^^^");
  else Serial.print("-+-");
  Serial.print(receiver_input_channel_3);

  Serial.print("  Yaw:");
  if(receiver_input_channel_4 - 1480 < 0)Serial.print("<<<");
  else if(receiver_input_channel_4 - 1520 > 0)Serial.print(">>>");
```

```
  else Serial.print("-+-");
  Serial.println(receiver_input_channel_4);
}

void esc_pulse_output(){
  zero_timer = micros();
  PORTD |= B11110000;                                          //Set port 4, 5, 6
and 7 high at once
  timer_channel_1 = esc_1 + zero_timer;                       //Calculate the time
when digital port 4 is set low.
  timer_channel_2 = esc_2 + zero_timer;                       //Calculate the time
when digital port 5 is set low.
  timer_channel_3 = esc_3 + zero_timer;                       //Calculate the time
when digital port 6 is set low.
  timer_channel_4 = esc_4 + zero_timer;                       //Calculate the time
when digital port 7 is set low.

  while(PORTD >= 16){                                          //Execute the loop
until digital port 4 to 7 is low.
    esc_loop_timer = micros();                                 //Check the current
time.
    if(timer_channel_1 <= esc_loop_timer)PORTD &= B11101111;    //When the delay time
is expired, digital port 4 is set low.
    if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;    //When the delay time
is expired, digital port 5 is set low.
    if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;    //When the delay time
is expired, digital port 6 is set low.
    if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;    //When the delay time
is expired, digital port 7 is set low.
  }
}

void set_gyro_registers(){
  //Setup the MPU-6050
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);                      //Start communication
with the address found during search.
    Wire.write(0x6B);                                          //We want to write
to the PWR_MGMT_1 register (6B hex)
    Wire.write(0x00);                                          //Set the register
bits as 00000000 to activate the gyro
    Wire.endTransmission();                                    //End the transmission
with the gyro.

    Wire.beginTransmission(gyro_address);                      //Start communication
with the address found during search.
    Wire.write(0x1B);                                          //We want to write
to the GYRO_CONFIG register (1B hex)
    Wire.write(0x08);                                          //Set the register
bits as 00001000 (500dps full scale)
    Wire.endTransmission();                                    //End the transmission
with the gyro

    Wire.beginTransmission(gyro_address);                      //Start communication
with the address found during search.
```

```
    Wire.write(0x1C);                                          //We want to write
to the ACCEL_CONFIG register (1A hex)
    Wire.write(0x10);                                          //Set the register
bits as 00010000 (+/- 8g full scale range)
    Wire.endTransmission();                                   //End the transmission
with the gyro

    //Let's perform a random register check to see if the values are written correct
    Wire.beginTransmission(gyro_address);                      //Start communication
with the address found during search
    Wire.write(0x1B);                                           //Start reading @
register 0x1B
    Wire.endTransmission();                                   //End the transmission
    Wire.requestFrom(gyro_address, 1);                          //Request 1 bytes
from the gyro
    while(Wire.available() < 1);                               //Wait until the 6
bytes are received
    if(Wire.read() != 0x08){                                    //Check if the value
is 0x08
      digitalWrite(12,HIGH);                                   //Turn on the warning
led
      while(1)delay(10);                                        //Stay in this loop
for ever
    }

    Wire.beginTransmission(gyro_address);                      //Start communication
with the address found during search
    Wire.write(0x1A);                                          //We want to write
to the CONFIG register (1A hex)
    Wire.write(0x03);                                          //Set the register
bits as 00000011 (Set Digital Low Pass Filter to ~43Hz)
    Wire.endTransmission();                                   //End the transmission
with the gyro

  }
}

void gyro_signalen(){
  //Read the MPU-6050
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);                      //Start communication
with the gyro.
    Wire.write(0x3B);                                           //Start reading @
register 43h and auto increment with every read.
    Wire.endTransmission();                                   //End the transmission.
    Wire.requestFrom(gyro_address,14);                          //Request 14 bytes
from the gyro.
    while(Wire.available() < 14);                              //Wait until the 14
bytes are received.
    acc_axis[1] = Wire.read()<<8|Wire.read();                   //Add the low and
high byte to the acc_x variable.
    acc_axis[2] = Wire.read()<<8|Wire.read();                   //Add the low and
high byte to the acc_y variable.
    acc_axis[3] = Wire.read()<<8|Wire.read();                   //Add the low and
high byte to the acc_z variable.
```

```
    temperature = Wire.read()<<8|Wire.read();                        //Add the low and
high byte to the temperature variable.
    gyro_axis[1] = Wire.read()<<8|Wire.read();                      //Read high and low
part of the angular data.
    gyro_axis[2] = Wire.read()<<8|Wire.read();                      //Read high and low
part of the angular data.
    gyro_axis[3] = Wire.read()<<8|Wire.read();                      //Read high and low
part of the angular data.
  }

  if(cal_int == 2000){
    gyro_axis[1] -= gyro_axis_cal[1];                               //Only compensate
after the calibration.
    gyro_axis[2] -= gyro_axis_cal[2];                               //Only compensate
after the calibration.
    gyro_axis[3] -= gyro_axis_cal[3];                               //Only compensate
after the calibration.
  }
  gyro_roll = gyro_axis[eeprom_data[28] & 0b00000011];             //Set gyro_roll to
the correct axis that was stored in the EEPROM.
  if(eeprom_data[28] & 0b10000000)gyro_roll *= -1;                 //Invert gyro_roll
if the MSB of EEPROM bit 28 is set.
  gyro_pitch = gyro_axis[eeprom_data[29] & 0b00000011];           //Set gyro_pitch to
the correct axis that was stored in the EEPROM.
  if(eeprom_data[29] & 0b10000000)gyro_pitch *= -1;               //Invert gyro_pitch
if the MSB of EEPROM bit 29 is set.
  gyro_yaw = gyro_axis[eeprom_data[30] & 0b00000011];              //Set gyro_yaw to
the correct axis that was stored in the EEPROM.
  if(eeprom_data[30] & 0b10000000)gyro_yaw *= -1;                 //Invert gyro_yaw if
the MSB of EEPROM bit 30 is set.

  acc_x = acc_axis[eeprom_data[29] & 0b00000011];                  //Set acc_x to the
correct axis that was stored in the EEPROM.
  if(eeprom_data[29] & 0b10000000)acc_x *= -1;                     //Invert acc_x if
the MSB of EEPROM bit 29 is set.
  acc_y = acc_axis[eeprom_data[28] & 0b00000011];                  //Set acc_y to the
correct axis that was stored in the EEPROM.
  if(eeprom_data[28] & 0b10000000)acc_y *= -1;                     //Invert acc_y if
the MSB of EEPROM bit 28 is set.
  acc_z = acc_axis[eeprom_data[30] & 0b00000011];                  //Set acc_z to the
correct axis that was stored in the EEPROM.
  if(eeprom_data[30] & 0b10000000)acc_z *= -1;                     //Invert acc_z if
the MSB of EEPROM bit 30 is set.
}
```

MUFC.c

```
/*MUFC-Multicopter UAV Flight Controller --Auto Levelling
 Modified by: Naufalino Fadel Hutomo and Adi Trisna
 Inspired By: J. Brokking, Oct 2016

*/
#define BUFFER_LENGTH 32         //for Wire function use
```

```
//library AVR
#include <util/delay.h>
#include <avr/eeprom.h>
#include <avr/io.h>
#include <avr/interrupt.h>

//library C
#include <stdlib.h>
#include <string.h>
#include <inttypes.h>
#include <math.h>
 #include "twi.h"

///PID constant and limit///
float kp_roll = 1.3;       //Proportional constant for roll
float ki_roll = 0.04;             //Integrative constant for roll
float kd_roll = 18.0;             //Diferentiative constant for roll
int pid_max_roll= 400;            //maximum output of PID for roll

float kp_pitch      = 1.3; //Pitch
float ki_pitch      = 0.04;
float kd_pitch      = 18.0;
int pid_max_pitch   = 400;

float kp_yaw  = 4.00;      //yaw
float ki_yaw  = 0.02;
float kd_yaw  = 0.0;
int pid_max_yaw     = 400;

int autoLevel = 1;                //Auto level 0 (false) ~0 (true)

///Variable for wire
uint8_t rxBuffer[BUFFER_LENGTH];
uint8_t rxBufferIndex = 0;
uint8_t rxBufferLength = 0;

uint8_t txAddress = 0;
uint8_t txBufferIndex = 0;
uint8_t txBufferLength = 0;
uint8_t txBuffer[BUFFER_LENGTH];
uint8_t transmitting =0;

uint8_t dumm;
int dummInt;
/// variable for micros
unsigned long micros=0;

///Global variable///
uint8_t last_ch1, last_ch2, last_ch3, last_ch4;
uint8_t eeprom_data[36];
uint8_t highByte, lowByte;
volatile int rec_input_ch1, rec_input_ch2, rec_input_ch3, rec_input_ch4;
int counter_ch1, counter_ch2, counter_ch3, counter_ch4;
int esc1, esc2, esc3, esc4;
int throttle, batteryVolt;
```

```
int cal_int, start, gyroAdrr;
int rec_input[5];
int temperature;
int acc_axis[4], gyro_axis[4];
float roll_level_adjust, pitch_level_adjust;

long accX, accY, accZ, accTotalVector;
unsigned long timerCh1, timerCh2, timerCh3, timerCh4, escTimer, escLoopTimer;
unsigned long timer1, timer2, timer3, timer4, currentTime;
unsigned long loopTimer;
double gyroPitch, gyroRoll, gyroYaw;
double gyro_axis_call[4];
float pid_error_temp;
float   pid_i_mem_roll,   pid_roll_setpoint,   gyro_roll_input,   pid_output_roll,
pid_last_roll_d_error;
float  pid_i_mem_pitch,  pid_pitch_setpoint,  gyro_pitch_input,  pid_output_pitch,
pid_last_pitch_d_error;
float    pid_i_mem_yaw,    pid_yaw_setpoint,    gyro_yaw_input,    pid_output_yaw,
pid_last_yaw_d_error;
float angle_roll_acc, angle_pitch_acc, angle_pitch, angle_roll;
int gyro_angle_set; //true or false


///Interrupt Routine to get micros
ISR(TIMER0_OVF_vect)
{
       micros+= 16;
}

///Interrupt whenever get signal from receiver
ISR(PCINT0_vect){
  currentTime = micros;
  //Channel 1=========================================
  if(PINB & 0b00000001){                                                    //Is
input 8 high?
    if(last_ch1 == 0)
       {                                                 //Input 8 changed from 0 to
1.
      last_ch1 = 1;                                                          //Remember
current input state.
      timer1 = currentTime;                                                //Set timer_1
to currentTime.

       }
  }
  else if(last_ch1 == 1){                                                   //Input 8 is
not high and changed from 1 to 0.
    last_ch1 = 0;                                                           //Remember
current input state.
    rec_input[1] = currentTime - timer1;                                   //Channel 1 is
currentTime - timer_1.
  }
  //Channel 2=========================================
  if(PINB & 0b00000010 ){                                                   //Is
input 9 high?
```

```
    if(last_ch2 == 0){                                                  //Input 9 changed
from 0 to 1.
      last_ch2 = 1;                                                       //Remember
current input state.
      timer2 = currentTime;                                             //Set timer_2
to currentTime.
    }
  }
  else if(last_ch2 == 1){                                               //Input 9 is
not high and changed from 1 to 0.
    last_ch2 = 0;                                                       //Remember
current input state.
    rec_input[2] = currentTime - timer2;                                //Channel 2 is
currentTime - timer_2.
  }
  //Channel 3=======================================
  if(PINB & 0b00000100 ){                                                 //Is
input 10 high?
    if(last_ch3 == 0){                                                  //Input 10
changed from 0 to 1.
      last_ch3 = 1;                                                     //Remember
current input state.
      timer3 = currentTime;                                             //Set timer_3
to currentTime.
    }
  }
  else if(last_ch3 == 1){                                               //Input 10 is
not high and changed from 1 to 0.
    last_ch3 = 0;                                                       //Remember
current input state.
    rec_input[3] = currentTime - timer3;                                //Channel 3 is
currentTime - timer_3.

  }
  //Channel 4=======================================
  if(PINB & 0b00001000 ){                                                 //Is
input 11 high?
    if(last_ch4 == 0){                                                  //Input 11
changed from 0 to 1.
      last_ch4 = 1;                                                     //Remember
current input state.
      timer4 = currentTime;                                             //Set timer_4
to currentTime.
    }
  }
  else if(last_ch4 == 1){                                               //Input 11 is
not high and changed from 1 to 0.
    last_ch4 = 0;                                                       //Remember
current input state.
    rec_input[4] = currentTime - timer4;                                //Channel 4 is
currentTime - timer_4.
  }
}

///Setup routine///
```

```
void setup( void)
{
	intSetup();
	//copy EEPROM data
	for(start = 0; start<=35; start++)
	{
		eeprom_data[start] = eepromRead(start);
	}

	start=0;			//set start back to zero
	gyroAdrr= eeprom_data[32];	//store gyro address from previous setup

	wireBegin();			//I2c as master

	TWBR = 12;		//I2C clock to 400kHz

	///IO config
	DDRD   |= 0b11110000;			//PD4, PD5, PD6, PD7, as output to ESC
	DDRB   |= 0b00110000;			// pin 8,9,10,11 or PB0-PB3 as input
from receiver, Pin 12 and 13 (PB4, PB5) as output

	//use LED in Arduino for indication
	PORTB  |= (1 << 5);  //arduino pin 13

	//check EEPROM signature to make sure the setup is executed
	while(eeprom_data[33]!='J' || eeprom_data[34] !='M' ||eeprom_data[35]!= 'B' )
_delay_ms(10);

	//if setup is completed without MPU, stop flight controller
	if (eeprom_data[31]==2 || eeprom_data[31]==3) _delay_ms(10);


	set_gyro_registers();		//Set the specific gyro registers

	for (cal_int=0; cal_int < 1250; cal_int ++)    //wait 5 seconds before continue
	{
		PORTD |= 0b11110000;
		_delay_ms(1);
		PORTD &= 0b00001111;
		_delay_ms(3);
	}

	///Determine gyro offset
	for (cal_int = 0; cal_int <2000; cal_int ++)
	{
		if (cal_int % 15 ==0)
		{
			PORTB ^= (0b00010000);     //blink LED
		}
		gyro_signalen();				//read gyro output
		gyro_axis_call[1]   += gyro_axis[1];		//add roll value
		gyro_axis_call[2]   += gyro_axis[2];		//add pitch value
		gyro_axis_call[3]   += gyro_axis[3];		// add yaw value
		// give pulse to esc, biar gak bunyi
		PORTD |= 0b11110000;
```

```
              _delay_ms(1);
              PORTD &= 0b00001111;
              _delay_ms(3);
       }
       //divide by 2000, from 2000 measurement to have gyro offset
       gyro_axis_call[1] /= 2000;              //roll value
       gyro_axis_call[2] /= 2000;              //pitch value
       gyro_axis_call[3] /= 2000;              //yaw value

       ///Interrupt setup
       PCICR  |= (1 << PCIE0);           //enable PCMSK0 scan
       PCMSK0       |= (1 << PCINT0);          //set PCINT0 / PD8 in Uno to trigger
interrupt
       PCMSK0       |= (1 << PCINT1);          //set PCINT1 / PD9 in Uno to trigger
interrupt
       PCMSK0       |= (1 << PCINT2);          //set PCINT2 / PD10 in Uno to trigger
interrupt
       PCMSK0       |= (1 << PCINT3);          //set PCINT3 / P11 in Uno to trigger
interrupt

       //Arming Mode. wait until receiver active and throttle in lowest value
       while (rec_input_ch3 < 990 || rec_input_ch3 > 1020 || rec_input_ch4 < 1400)
       {
              rec_input_ch3 = convert_receiver_ch(3);        //convert    the    ch3/
throttle to normalized value
              rec_input_ch4 = convert_receiver_ch(4);        // idem, ch4/yaw
              start++;            // while waiting, increment

              //ngilangin bunyi esc, kasih pulse
              PORTD |= 0b11110000;
              _delay_ms(1);
              PORTD &= 0b00001111;
              _delay_ms(3);
              if(start ==125)
              {
                     PORTB ^= 0b00010000;             //blink LED every 500 ms or 125
loop
                     start=0;            //start again at 0
              }
       }
       start=0;                  //set start to 0

       loopTimer = micros;        //set timer for next loop
       PORTB &= (0 << 5);         //set led to LOW when finished setup
}

int main ()
{
       setup();

       ///Main Loop
       while(1)
       {
              //65.5 = 1 deg/s, we get it from MPU 6050 datasheet
```

```
              gyro_roll_input  =  (gyro_roll_input  *  0.7)  +  ((gyroRoll/65.5)  *0.3);
              //convert the value, so gyro pid input is in deg/s
              gyro_pitch_input = (gyro_pitch_input * 0.7) + ((gyroPitch/65.5) *0.3);
              //convert the value, so gyro pid input is in deg/s
              gyro_yaw_input = (gyro_yaw_input * 0.7) + ((gyroYaw/65.5) *0.3);
       //convert the value, so gyro pid input is in deg/s

              //Calculate angle in gyro.       0.0000611 = 1 /(250 Hz / 65.5)
              angle_pitch += gyroPitch *       0.0000611;
              angle_roll   += gyroRoll *0.0000611;

              //because sin is in radian, convert to radian. 0.000001066 = 0.0000611
* 3.142 <pi> / 180 deg. Calculate angle if IMU has yaw effect, so it give side effect
in other angle. Untuk lebih jelas, harus melihat geometri dari Roll, yaw, dan pitch
              angle_pitch  -= angle_roll * sin(gyroYaw * 0.000001066);
              angle_roll   += angle_pitch * sin(gyroYaw * 0.000001066);

              //accel angle calculation
              accTotalVector =  sqrt((accX*accX) + (accY*accY) + (accZ*accZ));

              if (abs(accY) < accTotalVector)         //pitch   angle.   absolute   to
prevent NaN in asin function
              {
                     angle_pitch_acc = asin( (float) accY/accTotalVector) * 57.296;
              }

              if (abs(accX) < accTotalVector)         //roll angle
              {
                     angle_roll_acc = asin( (float) accX/accTotalVector) * -57.296;
              }

              // place MPU spirit level
              angle_pitch_acc -= 0.0;          //accel clibration angle pitch
              angle_roll_acc -= 0.0;

              angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;
       //correct the drift of gyro angle with accel angle
              angle_roll   = angle_roll  * 0.9996 + angle_roll_acc  * 0.0004;

              pitch_level_adjust = angle_pitch * 15;               //calculate    the
pitch angle correction
              roll_level_adjust = angle_roll * 15;                 //calculate    the
pitch angle correction

              if(!autoLevel)
              {
                     pitch_level_adjust = 0;
                     roll_level_adjust = 0;
              }

              ///Arming. Throttle low, yaw left
              if (rec_input_ch3 < 1050 && rec_input_ch4 <1050) start=1;
              //when yaw is back to center, motor start
              if (start==1 && rec_input_ch3 < 1050 && rec_input_ch4 > 1450 )
              {
```

```
                    start =2;

                    angle_pitch = angle_pitch_acc;          //gyro angle eq to accel
angle when the quad is started
                    angle_roll = angle_roll_acc;
                    gyro_angle_set = 1;

                    //Resete PID for bumpless
                    pid_i_mem_pitch = 0;
                    pid_i_mem_roll = 0;
                    pid_i_mem_yaw = 0;
                    pid_last_pitch_d_error = 0;
                    pid_last_roll_d_error = 0;
                    pid_last_yaw_d_error = 0;
             }

             /// DISARMING. Throttle low, yaw right
             if (start==2 && rec_input_ch3 < 1050 && rec_input_ch4 > 1950 ) start=0;

             //PID setpoint in deg/s.
             pid_roll_setpoint=0;
             //dead band of 16us
             if (rec_input_ch1 > 1508) pid_roll_setpoint=rec_input_ch1-1508;
             else if (rec_input_ch1 < 1492) pid_roll_setpoint = rec_input_ch1 -1492;

             pid_roll_setpoint -= roll_level_adjust;        //substract   the   angle
correction from the standardized value
             pid_roll_setpoint /= 3.0;                      // to get angle in deg/s

             //PID setpoint in deg/s.
             pid_pitch_setpoint=0;
             //dead band of 16us
             if (rec_input_ch2 > 1508) pid_pitch_setpoint=rec_input_ch2-1508;
             else if (rec_input_ch2 < 1492) pid_pitch_setpoint = rec_input_ch2 -1492;

             pid_pitch_setpoint -= pitch_level_adjust;            //substract    the
angle correction from the standardized value
             pid_pitch_setpoint /= 3.0;                     // to get angle in deg/s

             //PID setpoint in deg/s.
             pid_yaw_setpoint=0;
             //dead band of 16us
             if (rec_input_ch3 > 1050)
             {
                    if  (rec_input_ch4  >  1508)  pid_yaw_setpoint=  (rec_input_ch4-
1508)/3.0;
                    else if (rec_input_ch4 < 1492) pid_yaw_setpoint = (rec_input_ch4
-1492)/3.0;
             }

             calculate_pid();

             throttle = rec_input_ch3;        //as base signal

             if (start==2)
```

```
		{
			if (throttle > 1800) throttle =1800;
			esc1   =   throttle   -   pid_output_pitch   +   pid_output_roll   -
pid_output_yaw;
			esc2   =   throttle   +   pid_output_pitch   +   pid_output_roll   +
pid_output_yaw;
			esc3   =   throttle   +   pid_output_pitch   -   pid_output_roll   -
pid_output_yaw;
			esc4   =   throttle   -   pid_output_pitch   -   pid_output_roll   +
pid_output_yaw;

			//limiting the pulse signal
			if (esc1 <1100) esc1=1100;
			if (esc2 <1100) esc2=1100;
			if (esc3 <1100) esc3=1100;
			if (esc4 <1100) esc4=1100;

			if (esc1 > 2000) esc1=2000;
			if (esc2 > 2000) esc2=2000;
			if (esc3 > 2000) esc3=2000;
			if (esc4 > 2000) esc4=2000;
		}
		else   // still arming mode
		{
			esc1 = 1000;
			esc2 = 1000;
			esc3 = 1000;
			esc4 = 1000;
		}

		//looptime
		if(micros- loopTimer > 4050) PORTB &= (1 << 5);             //LED high

		//esc pulse every 4ms, because refresh rate is 250Hz
		while(micros-loopTimer < 4000);
		loopTimer = micros;

	PORTD |= 0b11110000;
	timer1 = esc1 + loopTimer;
	timer2 = esc2 + loopTimer;
	timer3 = esc3 + loopTimer;
	timer4 = esc4 + loopTimer;

	gyro_signalen();

	}
	return 0;
}

void set_gyro_registers(void)
{
	// Setup MPU 6050
	if (eeprom_data[31] == 1)
	{
		wireBeginTransmission(gyroAdrr);
```

```
              dummInt = wireWrite(0x6B);        //write to PWR_MGMT_1 register, 6B hex
              dummInt = wireWrite (0x00);              // acticvate gyro
              dumm = wireEndTransmission();

              wireBeginTransmission(gyroAdrr);
              dummInt = wireWrite(0x1B);        // GYRO_CONFIG register
              dummInt = wireWrite(0x08);        //set as 00001000 equal to 500 dps
              dumm = wireEndTransmission();

              wireBeginTransmission(gyroAdrr);
              dummInt = wireWrite(0x1C);        // ACCEL_CONFIG register
              dummInt = wireWrite(0x10);        //set as 00010000 equal to +- 8 g full
scale
              dumm = wireEndTransmission();

              //check
              wireBeginTransmission(gyroAdrr);
              dummInt = wireWrite(0x1B);
              dumm = wireEndTransmission();
              dumm = wireRequestFrom(gyroAdrr,1);            //request 1 byte
              while (wireAvailable() < 1);
              if (wireRead() != 0x08)
              {
                     PORTB |= (1 << 5);         // LED High
                     while(1) _delay_ms(10);
              }

              wireBeginTransmission(gyroAdrr);
              dummInt = wireWrite(0x1A);        //write CONFIG register
              dummInt = wireWrite(0x03);        // set bits to 00000011
              dumm = wireEndTransmission();

       }

}

void gyro_signalen  ()
{
       //Read MPU 6050
       if (eeprom_data[31] ==1)
       {
              wireBeginTransmission(gyroAdrr);
              dummInt = wireWrite(0x3B);
              dumm = wireEndTransmission();
              dumm = wireRequestFrom(gyroAdrr,14);           //request  14  bytes  from
gyro

              rec_input_ch1 = convert_receiver_ch(1);
              rec_input_ch2 = convert_receiver_ch(2);
              rec_input_ch3 = convert_receiver_ch(3);
              rec_input_ch4 = convert_receiver_ch(4);

              while (wireAvailable() < 14);
              acc_axis[1] = wireRead()<<8 | wireRead();                          //Add
the low and high byte to the acc_x variable.
```

```
              acc_axis[2] = wireRead() <<8| wireRead();                                  //Add
the low and high byte to the acc_y variable.
              acc_axis[3] = wireRead() <<8| wireRead();                                  //Add
the low and high byte to the acc_z variable.
              temperature = wireRead() <<8| wireRead();                                  //Add
the low and high byte to the temperature variable.
              gyro_axis[1] = wireRead() <<8| wireRead();                                //Read
high and low part of the angular data.
              gyro_axis[2] = wireRead() <<8| wireRead();                                //Read
high and low part of the angular data.
              gyro_axis[3] = wireRead() <<8| wireRead();                                //Read
high and low part of the angular data.
       }

       if (cal_int == 2000)       //only compensate after calibration
       {
              gyro_axis[1] -= gyro_axis_call[1];
              gyro_axis[2] -= gyro_axis_call[2];
              gyro_axis[3] -= gyro_axis_call[3];
       }

       gyroRoll     = gyro_axis[eeprom_data[28] & 0b00000011];
       if (eeprom_data[28] & 0b10000000)       gyroRoll *= -1;
       //invert if MSB of EEPROM bit 28 is set
       gyroPitch    = gyro_axis[eeprom_data[29] & 0b00000011];
       if (eeprom_data[29] & 0b10000000)       gyroPitch *= -1;           //invert if
MSB of EEPROM bit 28 is set
       gyroYaw      = gyro_axis[eeprom_data[30] & 0b00000011];
       if (eeprom_data[30] & 0b10000000)       gyroYaw *= -1;
       //invert if MSB of EEPROM bit 28 is set

       accX = acc_axis[eeprom_data[29] & 0b00000011];       //set accX to the correct
axis that was stored in EEPROM
       if (eeprom_data[29] & 0b10000000) accX *= -1;
       accY = acc_axis[eeprom_data[28] & 0b00000011];       //set accY to the correct
axis that was stored in EEPROM
       if (eeprom_data[28] & 0b10000000) accY *= -1;
       accZ = acc_axis[eeprom_data[30] & 0b00000011];       //set accZ to the correct
axis that was stored in EEPROM
       if (eeprom_data[30] & 0b10000000) accZ *= -1;
}

///convert the actual receiver signal to normalized 1000-2000 us. Useful to know
whether the channel is inversed, overvalued, or undervalued
int convert_receiver_ch (int x)
{
       uint8_t channel, reverse;
       int low, center, high, actual, dif;

       channel = eeprom_data[x+23] & 0b00000111;      //What channel corresponds with
the asked variable x
       if (eeprom_data[x+23] & 0b10000000) reverse =1;             //reverse  channel
when msb is set
       else reverse=0;
```

```
	actual = rec_input[channel];
	low = (eeprom_data[channel*2 + 15] <<8) | eeprom_data[channel *2 +14];
	//store low value
	center = (eeprom_data[channel*2 -1] <<8) | eeprom_data[channel *2 - 2];
	//store center value
	high = (eeprom_data[channel*2 + 7] <<8) | eeprom_data[channel *2 + 6];
	//store high value

	if (actual < center)
	{
		if (actual<low) actual=low;		//  limit  to  the  value  which
determined in setup configuration before
		dif = ((long) (center-actual) * (long) 500) / (center-low);	//
scaling to a 1000-2000 us
		if(reverse==1) return 1500+dif;
		else return 1500-dif;
	}
	else if (actual>center)
	{
		if(actual>high) actual=high;	// limit to the value shich determined
in setup configuration
		dif = ((long) (actual-center) * (long) 500) / (high-center);
		if(reverse==1) return 1500-dif;
		else return 1500+dif;
	}
	else return 1500;

}

///PID
void calculate_pid ()
{
	//roll
	pid_error_temp = gyro_roll_input - pid_roll_setpoint;
	pid_i_mem_roll += ki_roll*pid_error_temp;
	if (pid_i_mem_roll > pid_max_roll) pid_i_mem_roll = pid_max_roll;
	else if (pid_i_mem_roll < pid_max_roll *-1) pid_i_mem_roll = pid_max_roll *-1;

	pid_output_roll  =  kp_roll  *  pid_error_temp  +  pid_i_mem_roll  +  kd_roll  *
(pid_error_temp - pid_last_roll_d_error);
	if(pid_output_roll > pid_max_roll)pid_output_roll = pid_max_roll;
	else if(pid_output_roll < pid_max_roll * -1)pid_output_roll = pid_max_roll * -
1;

	pid_last_roll_d_error = pid_error_temp;

	///Pitch
	pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;
	pid_i_mem_pitch += ki_pitch * pid_error_temp;
	if(pid_i_mem_pitch > pid_max_pitch)pid_i_mem_pitch = pid_max_pitch;
	else if(pid_i_mem_pitch < pid_max_pitch * -1)pid_i_mem_pitch = pid_max_pitch *
-1;

	pid_output_pitch = kp_pitch * pid_error_temp + pid_i_mem_pitch + kd_pitch *
(pid_error_temp - pid_last_pitch_d_error);
```

```
	if(pid_output_pitch > pid_max_pitch)pid_output_pitch = pid_max_pitch;
	else if(pid_output_pitch < pid_max_pitch * -1)pid_output_pitch = pid_max_pitch
* -1;

	pid_last_pitch_d_error = pid_error_temp;

	///Yaw
	pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;
	pid_i_mem_yaw += ki_yaw * pid_error_temp;
	if(pid_i_mem_yaw > pid_max_yaw)pid_i_mem_yaw = pid_max_yaw;
	else if(pid_i_mem_yaw < pid_max_yaw * -1)pid_i_mem_yaw = pid_max_yaw * -1;

	pid_output_yaw = kp_yaw * pid_error_temp + pid_i_mem_yaw + kd_yaw *
(pid_error_temp - pid_last_yaw_d_error);
	if(pid_output_yaw > pid_max_yaw)pid_output_yaw = pid_max_yaw;
	else if(pid_output_yaw < pid_max_yaw * -1)pid_output_yaw = pid_max_yaw * -1;

	pid_last_yaw_d_error = pid_error_temp;
}

void wireBegin()
{
	rxBufferIndex = 0;
	rxBufferLength = 0;

	txBufferIndex = 0;
	txBufferLength = 0;

	twi_init();
}

int wireBeginTransmission(int address)
{
	// indicate that we are transmitting
  transmitting = 1;
  // set address of targeted slave
  txAddress = (uint8_t) address;
  // reset tx buffer iterator vars
  txBufferIndex = 0;
  txBufferLength = 0;

}

int wireWrite(int data)
{
	if(transmitting)	// in master transmitter mode
	{
		if(txBufferLength >= BUFFER_LENGTH)
		{
		  return 0;
		}
		// put byte in tx buffer
		txBuffer[txBufferIndex] = data;
		++txBufferIndex;
		// update amount in buffer
```

```
                txBufferLength = txBufferIndex;
        }else
        {
          // in slave send mode
                // reply to master
                twi_transmit(&data, 1);
        }
        return 1;
}

int wireEndTransmission()
{
        // transmit buffer (blocking)
  uint8_t ret = twi_writeTo(txAddress, txBuffer, txBufferLength, 1, 1);
  // reset tx buffer iterator vars
  txBufferIndex = 0;
  txBufferLength = 0;
  // indicate that we are done transmitting
  transmitting = 0;
  return ret;

}

int wireRequestFrom (int address, int quantity)
{
        if(quantity > BUFFER_LENGTH){
                quantity = BUFFER_LENGTH;
        }

        // perform blocking read into buffer
        uint8_t read = twi_readFrom(address, rxBuffer, quantity, 1);
        // set rx buffer iterator vars
        rxBufferIndex = 0;
        rxBufferLength = read;

        return read;

}

int wireAvailable()
{
        return rxBufferLength - rxBufferIndex;
}

int wireRead ()
{
        int value = -1;

        // get each successive byte on each call
        if(rxBufferIndex < rxBufferLength)
        {
                value = rxBuffer[rxBufferIndex];
                ++rxBufferIndex;
        }
        return value;
```

```
}

void intSetup (void)
{
       //Timer 0 for micros
       TCCR0B |= (1<<CS00); //prescaler 1. Interrupt when overflow, at 1/16MHz*2^8=16us
       TIMSK0 |= (1<<TOIE0); //overflow interrupt enable

    sei();
}
```